Semiconductors

UM10036_1

ISP1582 PC Eval Kit (PCI) User's Guide

July 2003

User's Guide

Rev. 1.0

Revision History:

Version	**Date**	**Description**	**Author**
1.0	July 2003	Initial version	Albert Goh

We welcome your feedback. Send it to wired.support@philips.com.

Philips Semiconductors - Asia Product Innovation Centre
Visit www.semiconductors.philips.com/buses/usb or www.flexiusb.com

PHILIPS

This is a legal agreement between you (either an individual or an entity) and Philips Semiconductors. By accepting this product, you indicate your agreement to the disclaimer specified as follows:

DISCLAIMER

PRODUCT IS DEEMED ACCEPTED BY RECIPIENT. THE PRODUCT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, PHILIPS SEMICONDUCTORS FURTHER DISCLAIMS ALL WARRANTIES, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANT ABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT. THE ENTIRE RISK ARISING OUT OF THE USE OR PERFORMANCE OF THE PRODUCT AND DOCUMENTATION REMAINS WITH THE RECIPIENT. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL PHILIPS SEMICONDUCTORS OR ITS SUPPLIERS BE LIABLE FOR ANY CONSEQUENTIAL, INCIDENTAL, DIRECT, INDIRECT, SPECIAL, PUNITIVE, OR OTHER DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR OTHER PECUNIARY LOSS) ARISING OUT OF THIS AGREEMENT OR THE USE OF OR INABILITY TO USE THE PRODUCT, EVEN IF PHILIPS SEMICONDUCTORS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

CONTENTS

FIGURES

TABLES

1. Introduction

The ISP1582 Hi-Speed USB Device PC evaluation kit (PCI) allows you to evaluate the features of the ISP1582—which is a Hi-Speed Universal Serial Bus (USB) interface device—in the Generic Processor Mode, that is, separate address and data bus operation. It helps you evaluate the ISP1582 as a generic slave direct memory access (DMA) device, that is, as a PC kit on PCI.

On these boards are the ISP1582, Xilinx® XCS30XL, Xilinx XC17S30XL serial PROM, PLX9054 PCI-to-local bus bridge, SRAM, and serial EEPROM. The PC1 eval kit can connect the ISP1582 to any generic processor. Figure 1-1 shows the ISP1582 PCI board.

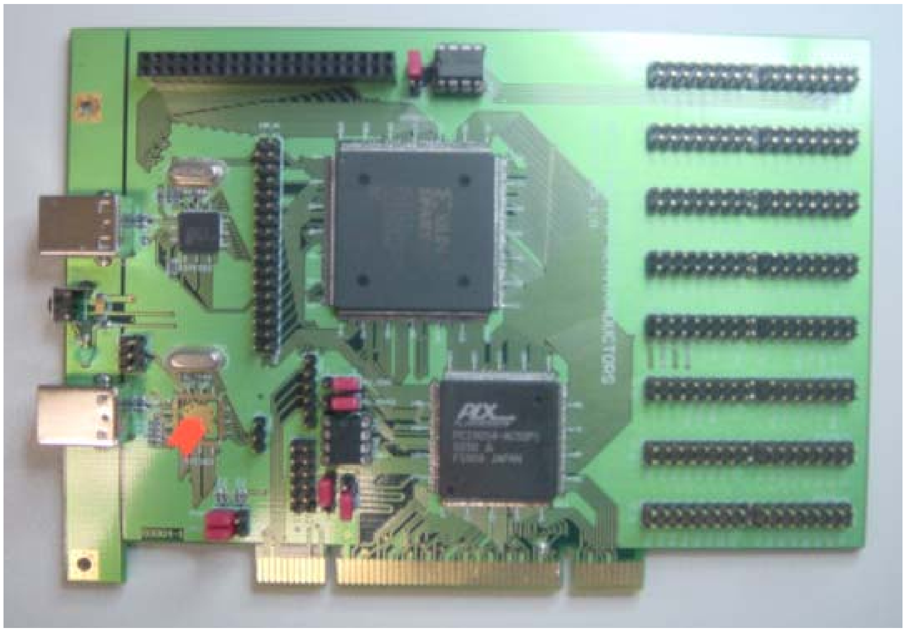


Figure 1-1 ISP1582 PCI Board

2. System Requirements

PC Host:

- Hi-Speed USB Host Controller add-on card*
- Ping pong application for GDMA for Microsoft® Windows® 2000 and Windows XP.

Device:

- Device PC for the PC kit; must be running on Windows 98 or Windows Millennium Edition (Me)*

Firmware:

- Turbo C*
- Firmware for the PC kit.

* —*Denotes that the item will not be included in the evaluation kit.*

3. Block Diagram

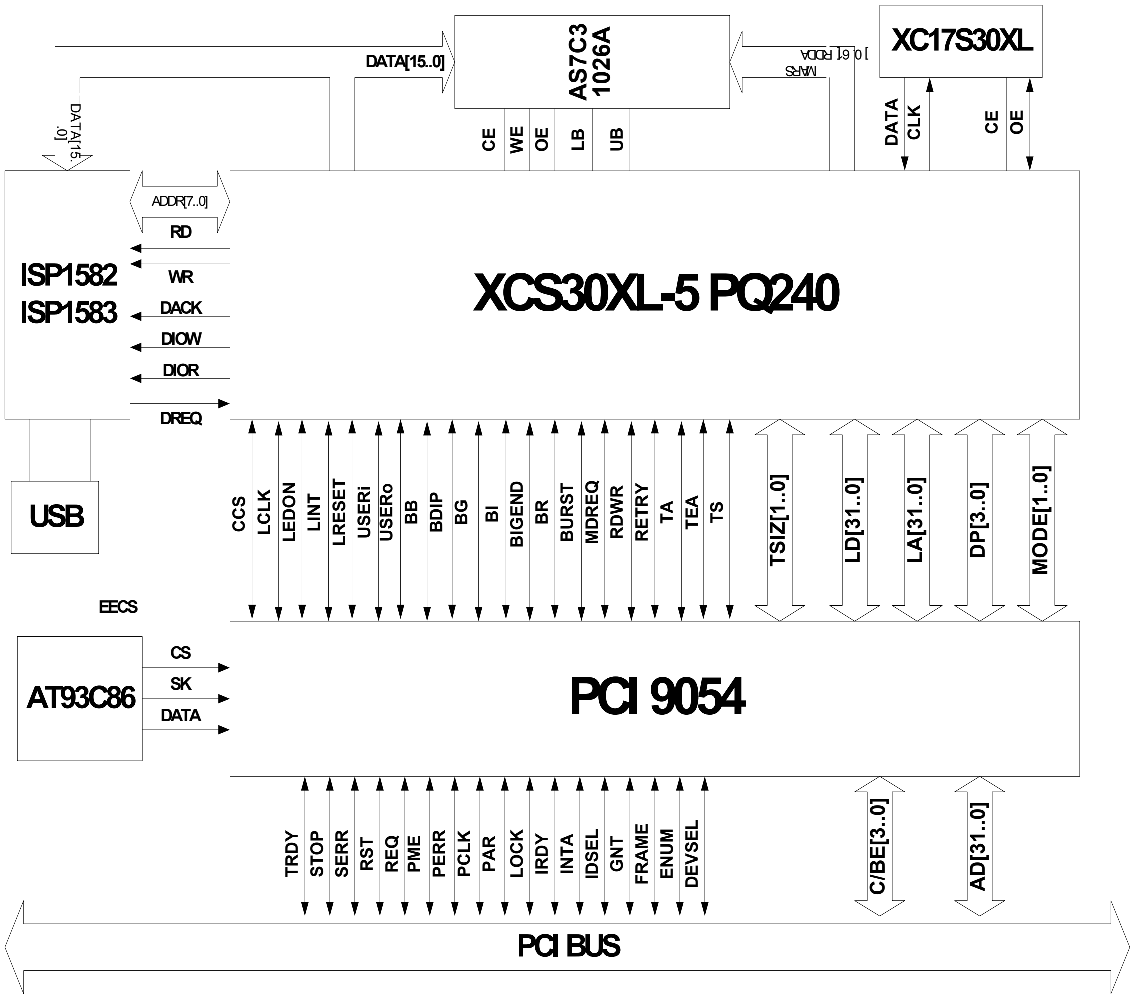


Figure 3-1: PCI Board Block Diagram

Figure 3-1 illustrates how the ISP1582 can be configured to work in the Generic Processor mode. The device PC acts as a processor that has the data bus shared between the DMA Controller and the processor. The device PC can access the ISP1582 registers with the help of the Xilinx XCS30XL, which converts the PLX9054 local bus to the ISP1582 generic processor bus. The PLX9054 converts PCI access to local bus access. On the PCI kit, data from DMA or PIO access is stored in the SRAM.

4. PCB Layout

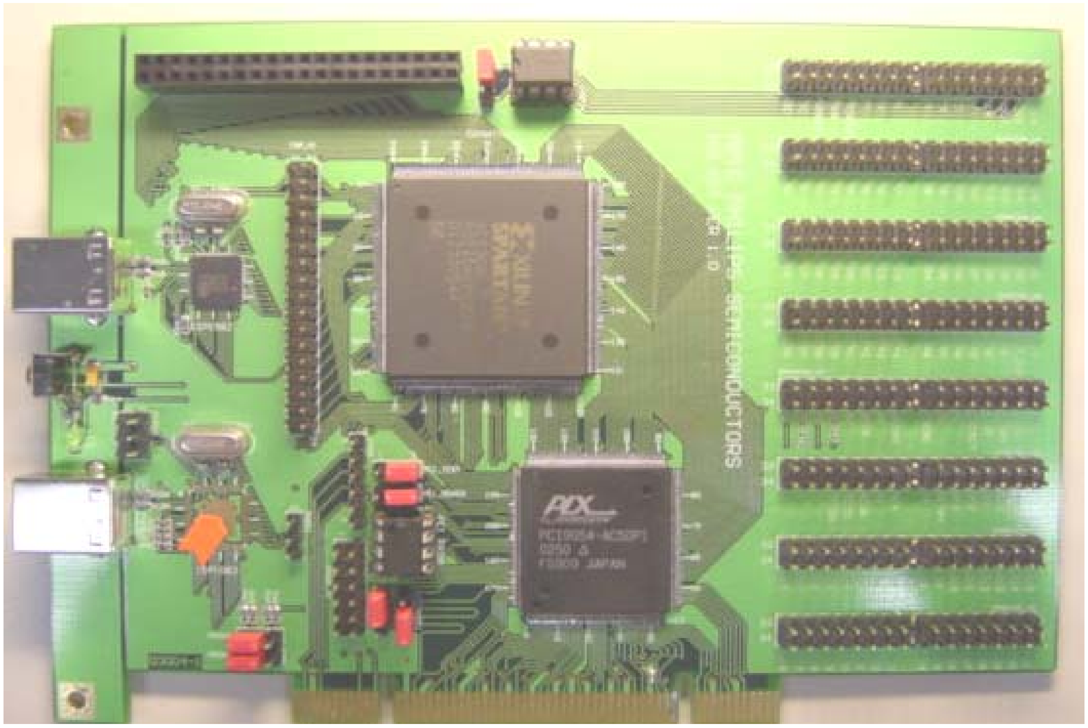
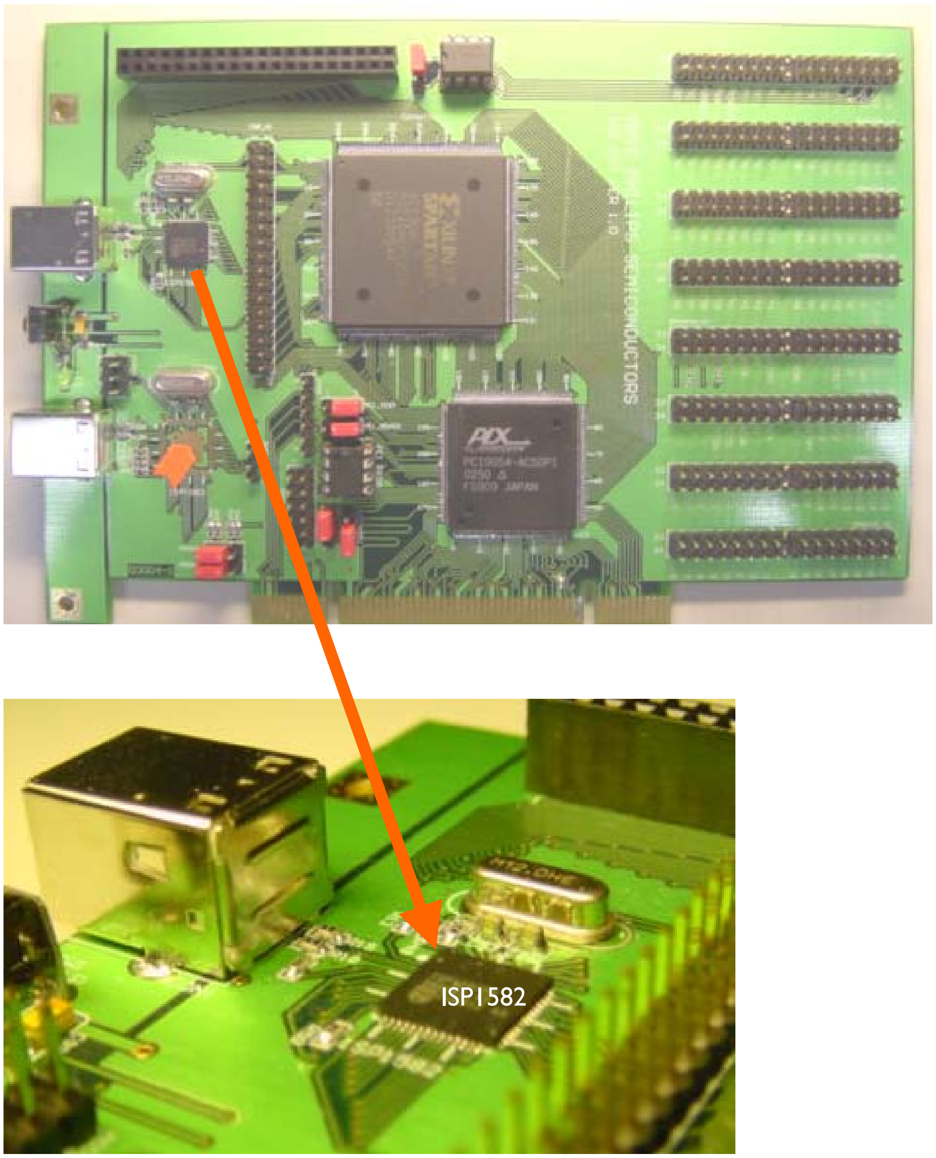


Figure 4-1: PCI Board PCB

Figure 4-1 shows the PCB layout and the placement of components on the ISP1582 PCI board. The PCB is designed for future expansion for ISP1583.

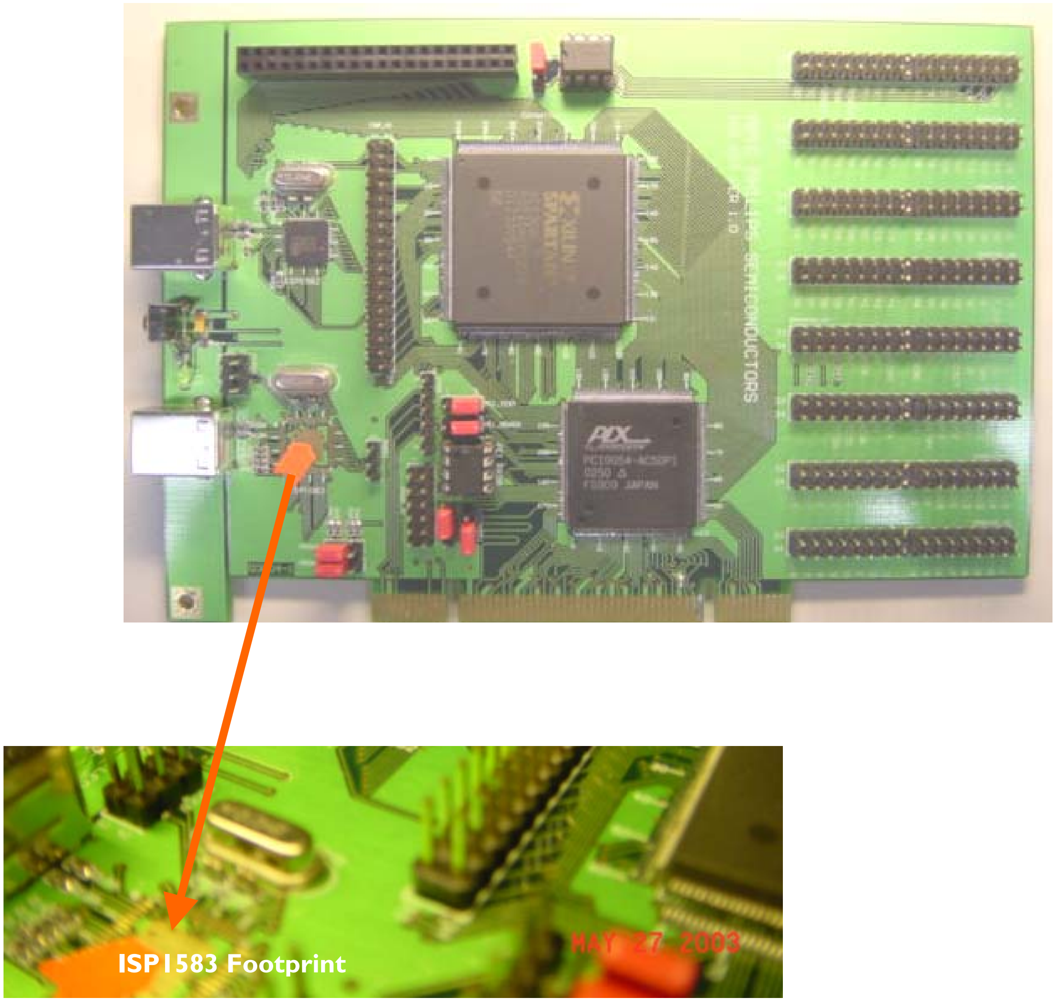
5. Component Placement

5.1. ISP1582



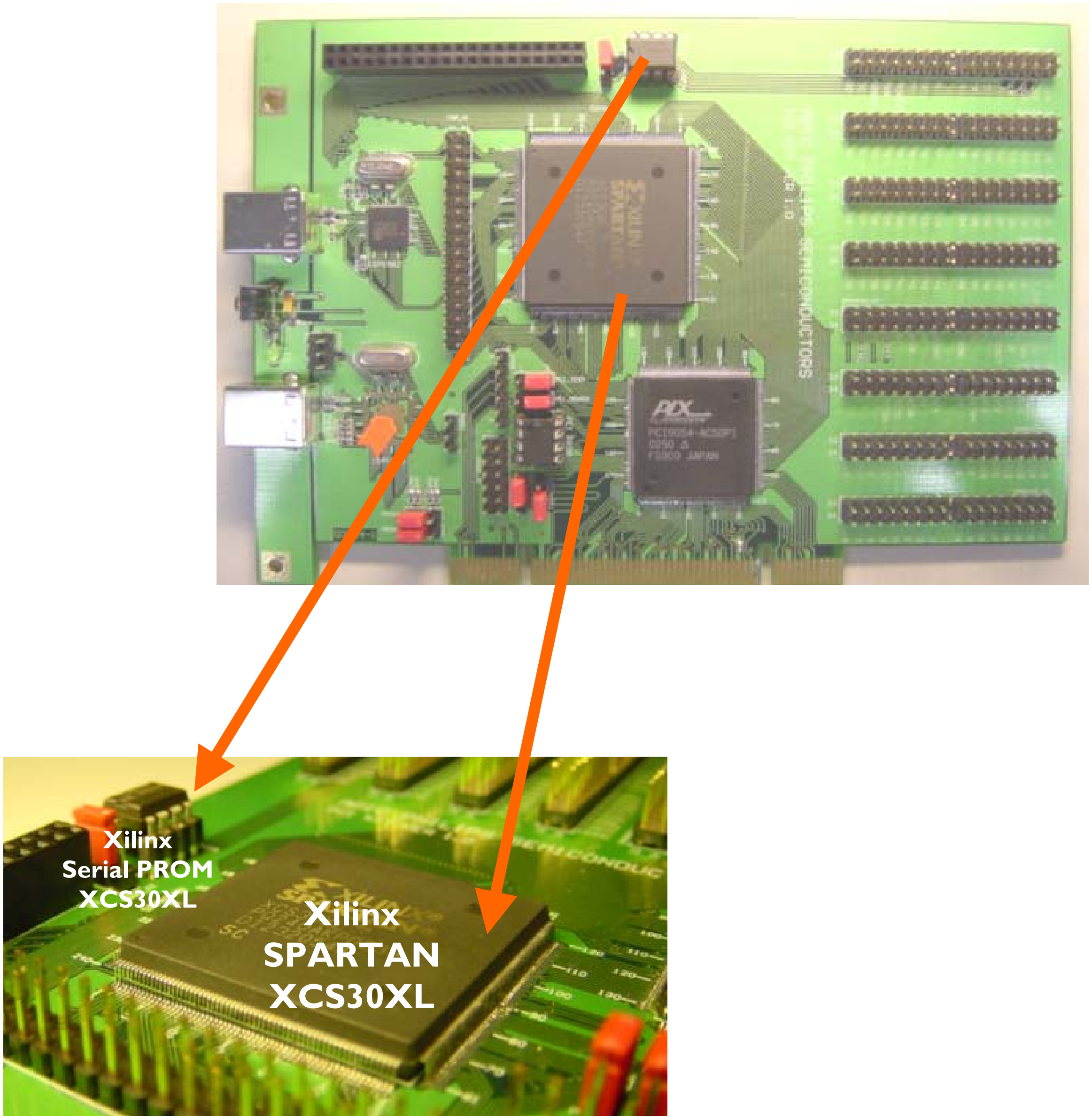
ISP1582 is at the upper-left corner of the PCI board.

5.2. ISP1583



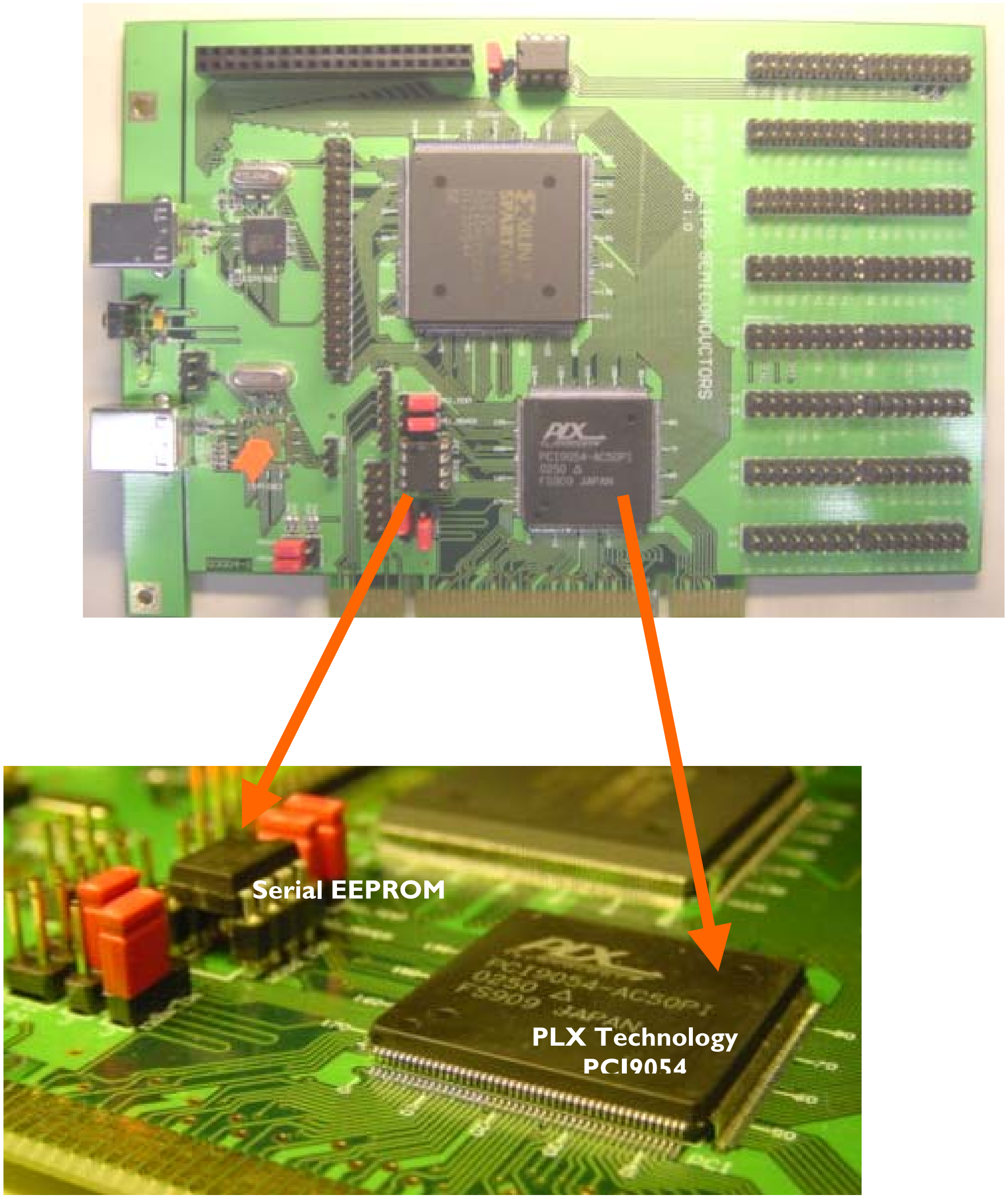
The PCI board also includes the footprint for ISP1583 to cater for future expansion of the PCI kit to ISP1583. ISP1583 is another Philips Hi-Speed USB interface device that in addition to supporting Generic Mode for CPU interface like the ISP1582, it also supports Split Bus mode and direct interface to any ATA/ATAPI device.

5.3. Xilinx XCS30XL FPGA and XC17S30XL Serial PROM



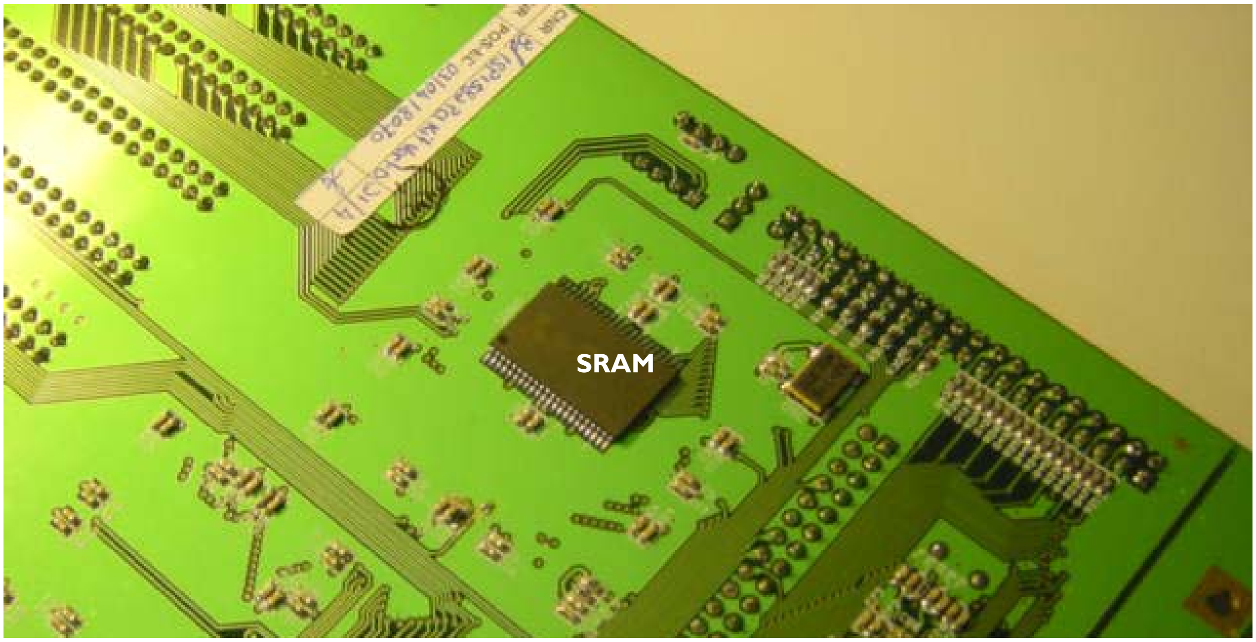
The Xilinx XCS30XL translates the PLX9054 PCI bridge local bus to the ISP1582 generic processor bus. It also acts as the Master DMA controller for the DMA transfer. Upon power on, the Xilinx XC17S30XL serial PROM configures the Xilinx XCS30XL to the local bus translator and the master DMA controller.

5.4. PLX Technology PCI9054 PCI Bridge and Serial EEPROM



The PLX technology PCI9054 is a PCI-to-local bus bridge. The PCI9054 has been configured to C mode local bus, which is a non-multiplexed address and data bus. Upon power on, the serial EEPROM configures the PCI9054 to the C mode local bus and interrupt set to INTA on the PCI bus.

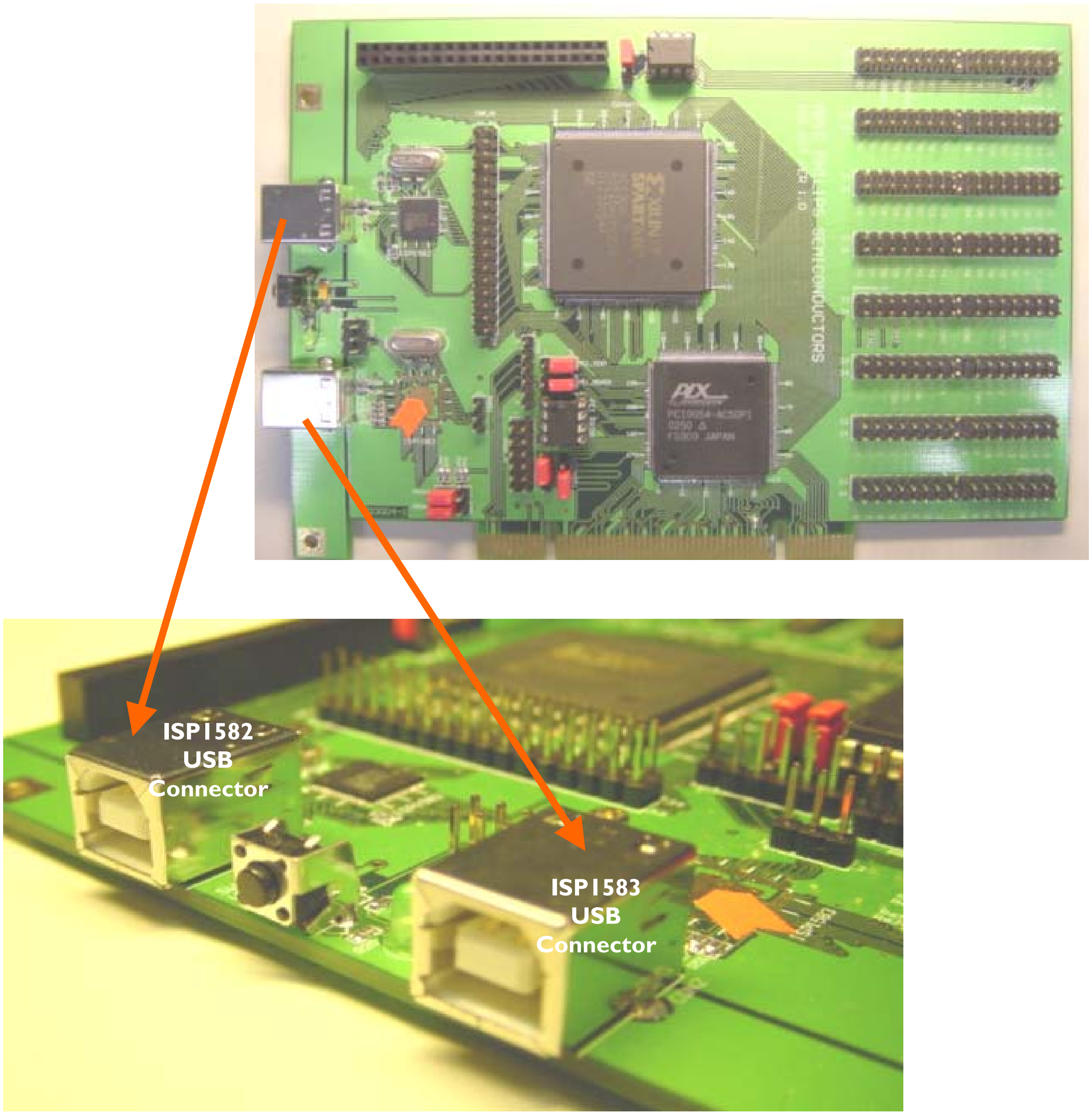
5.5. SRAM



The SRAM is at the bottom layer of the PCI board. The SRAM acts as storage space for the data when the device is configured to perform PIO or DMA data transfer. The size of the SRAM is 64K x 16.

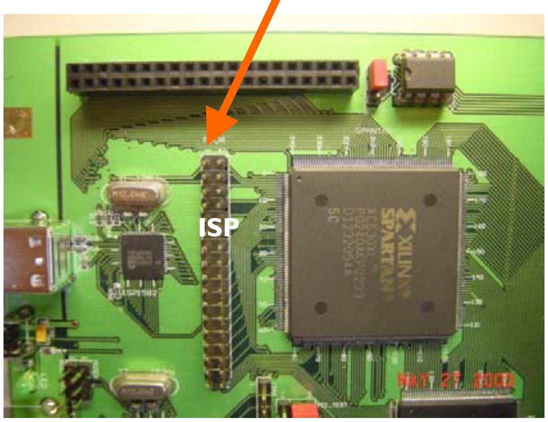
6. Header and Connector Placement

6.1. USB Connectors

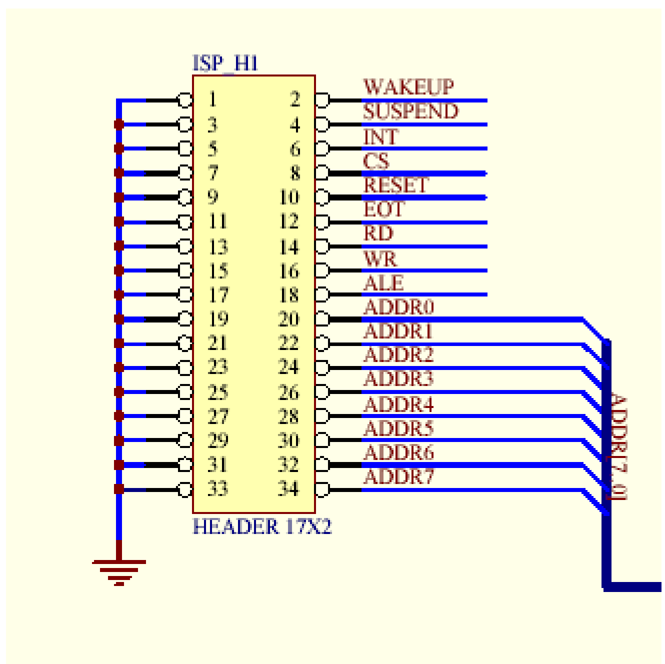


The ISP1582 USB connector is at the upper section of the PCI board. The ISP1583 USB connector at the lower section is for future expansion of the PCI board to accommodate the ISP1583.

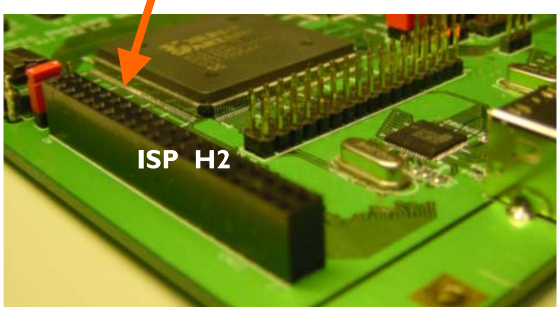
6.2. ISP1582 Generic Processor Expansion Bus

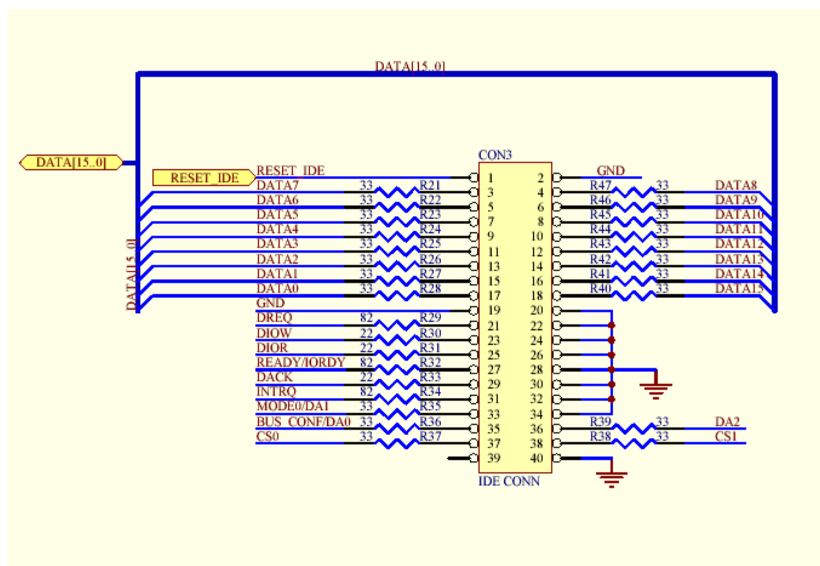


Acts as an expansion bus for connection to the other non-multiplexed bus processor



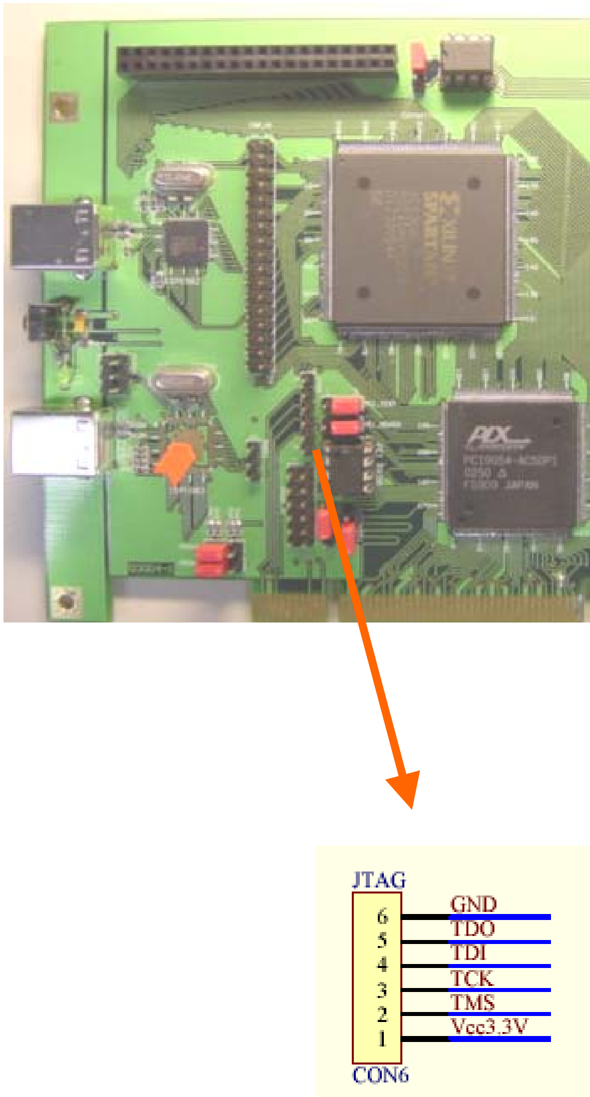
6.3. ISP1582 DMA Expansion Bus





Acts as a DMA expansion bus for connection to an external DMA controller and the ISP1582's 16-bit data bus.

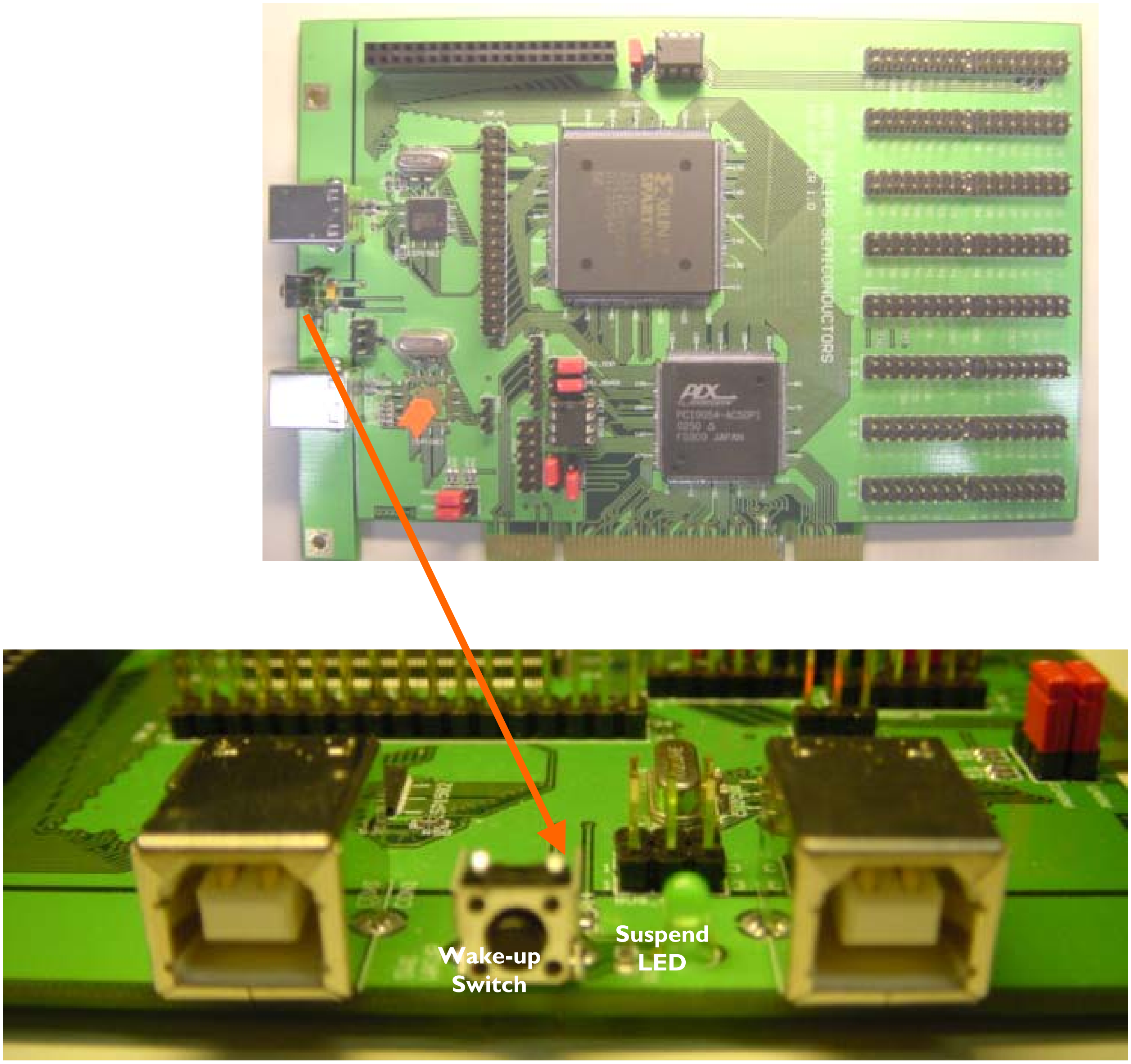
6.4. JTAG Header



The JTAG header allows the end-user to reprogram the Xilinx® XCS30XL to perform a user-defined operation.

7. ISP1582 PCI Kit Switch and LED Placement

7.1. Wake-Up Switch and Suspend LED Placement



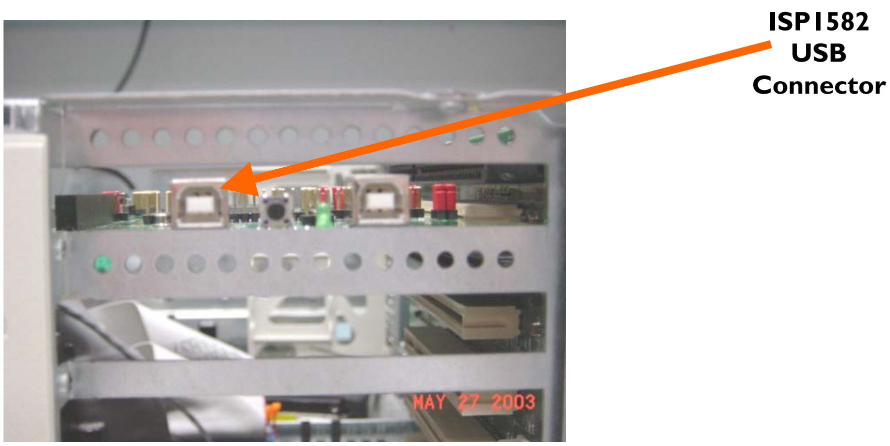
The wake-up switch is tied to the wake-up pin of the ISP1582, which will wake up the ISP1582 when it is in the suspend mode. The suspend LED indicates the suspend state of ISP1582. When turned on, ISP1582 is in the suspend mode. Currently, this version of the PCI kit does not support the wake-up switch and the suspend LED.

8. ISP1582 PCI Eval Kit Setup Procedure

8.1. PCI Kit Setup Procedure

1. Switch off the device PC and allocate an empty PCI slot.
2. Insert the ISP1582 PCI board into an available PCI slot, and switch on the device PC.

3. Run the PCIKit.exe file on the device PC.
4. Plug in the host USB cable to the ISP1582 USB connector.



5. After successful enumeration, run the USB Device applet on the host PC.

8.2. PCI Kit Host PC Setup and Bus Enumeration Procedure

If the evaluation board is connected for the first time to the host PC, the host OS Device Manager will prompt for the installation of the INF and the driver. Select the location of Phkit.inf and Phkit.sys from the ISP1582 evaluation diskette and complete the installation procedure.
On successful installation, you will see the device added in the Computer Management window under Device Manager as shown in Figure 8-1.

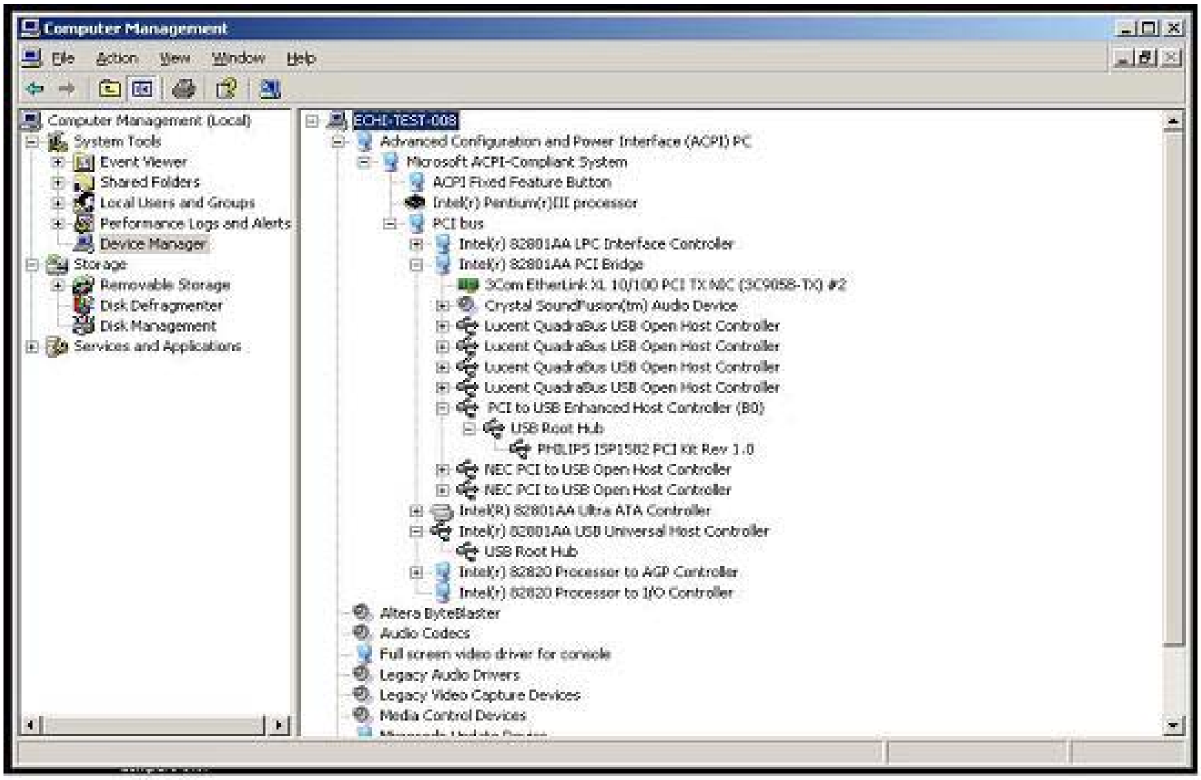


Figure 8-1: Hi-Speed USB Device on Philips ISP1561 EHCI USB 2.0 Host Controller

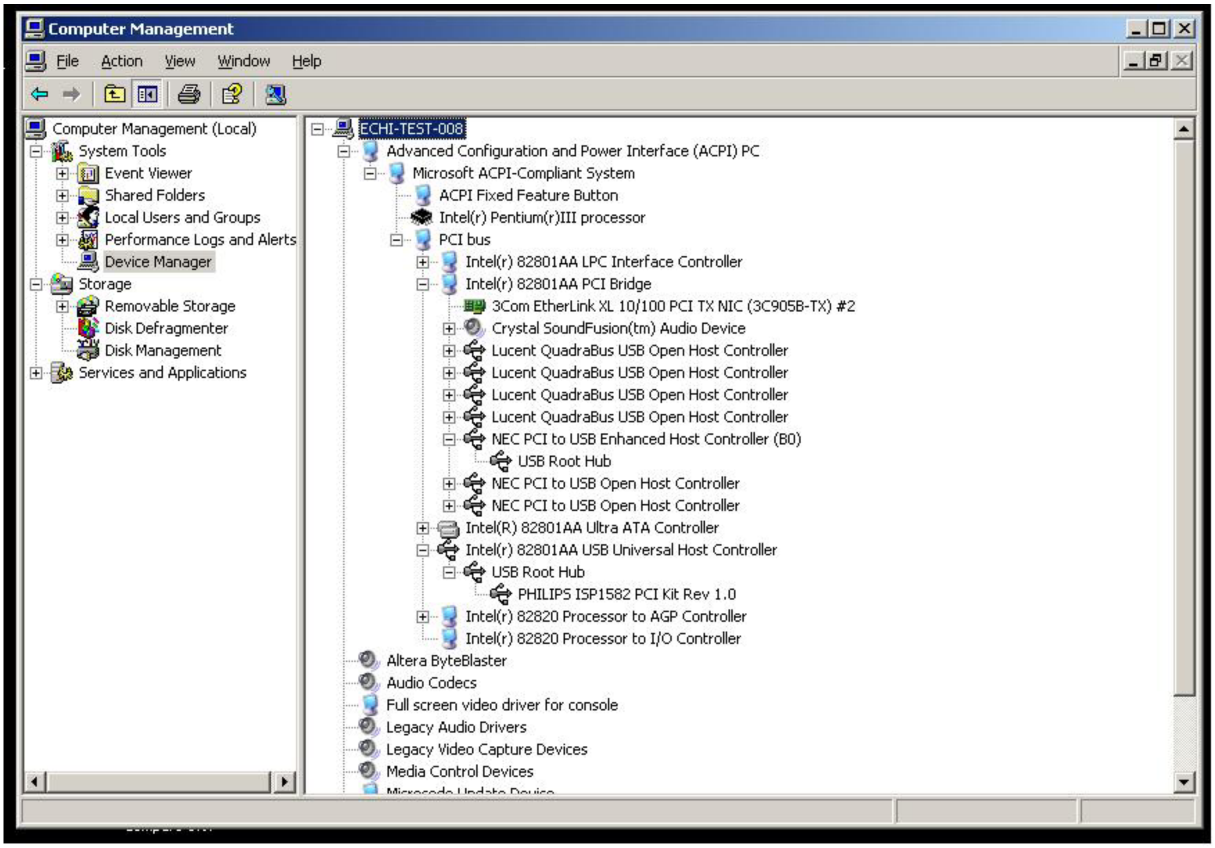
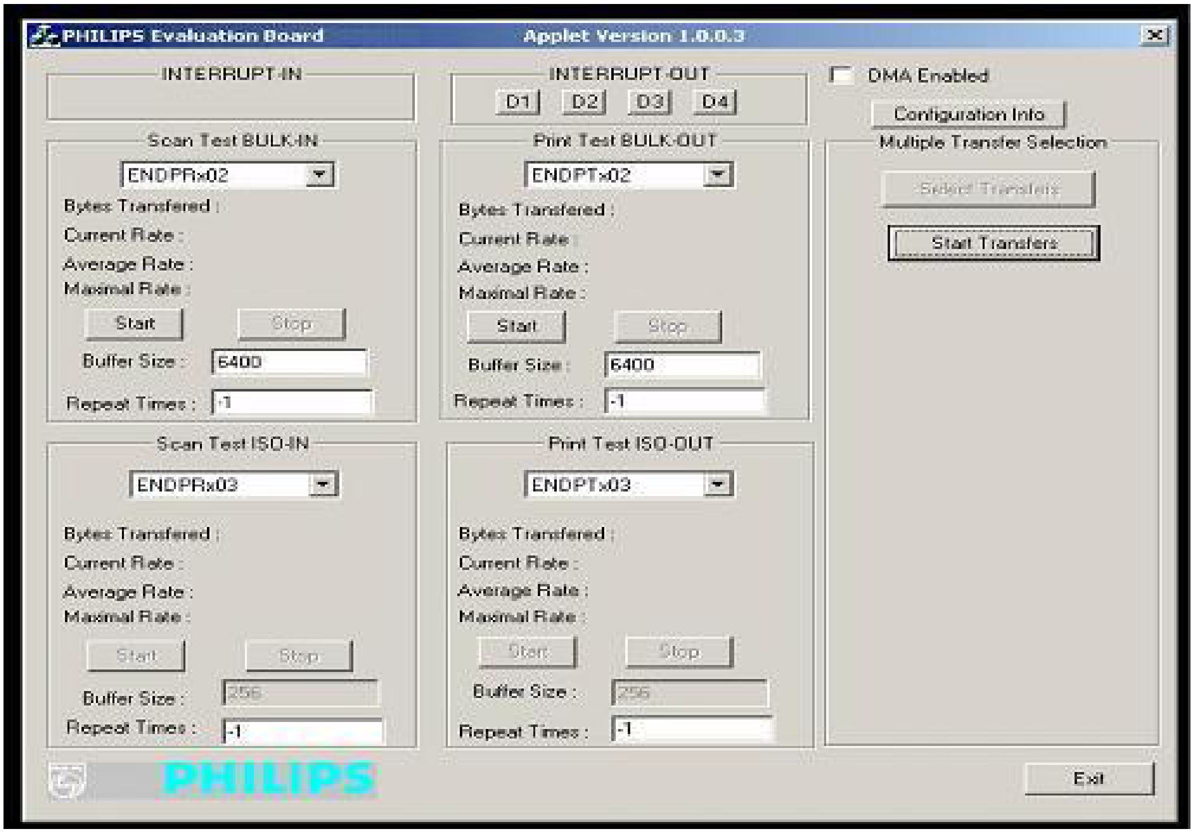
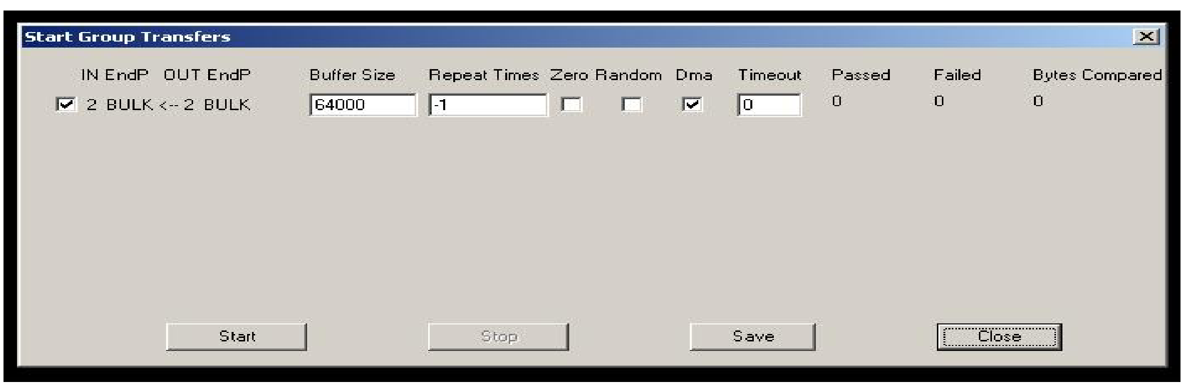


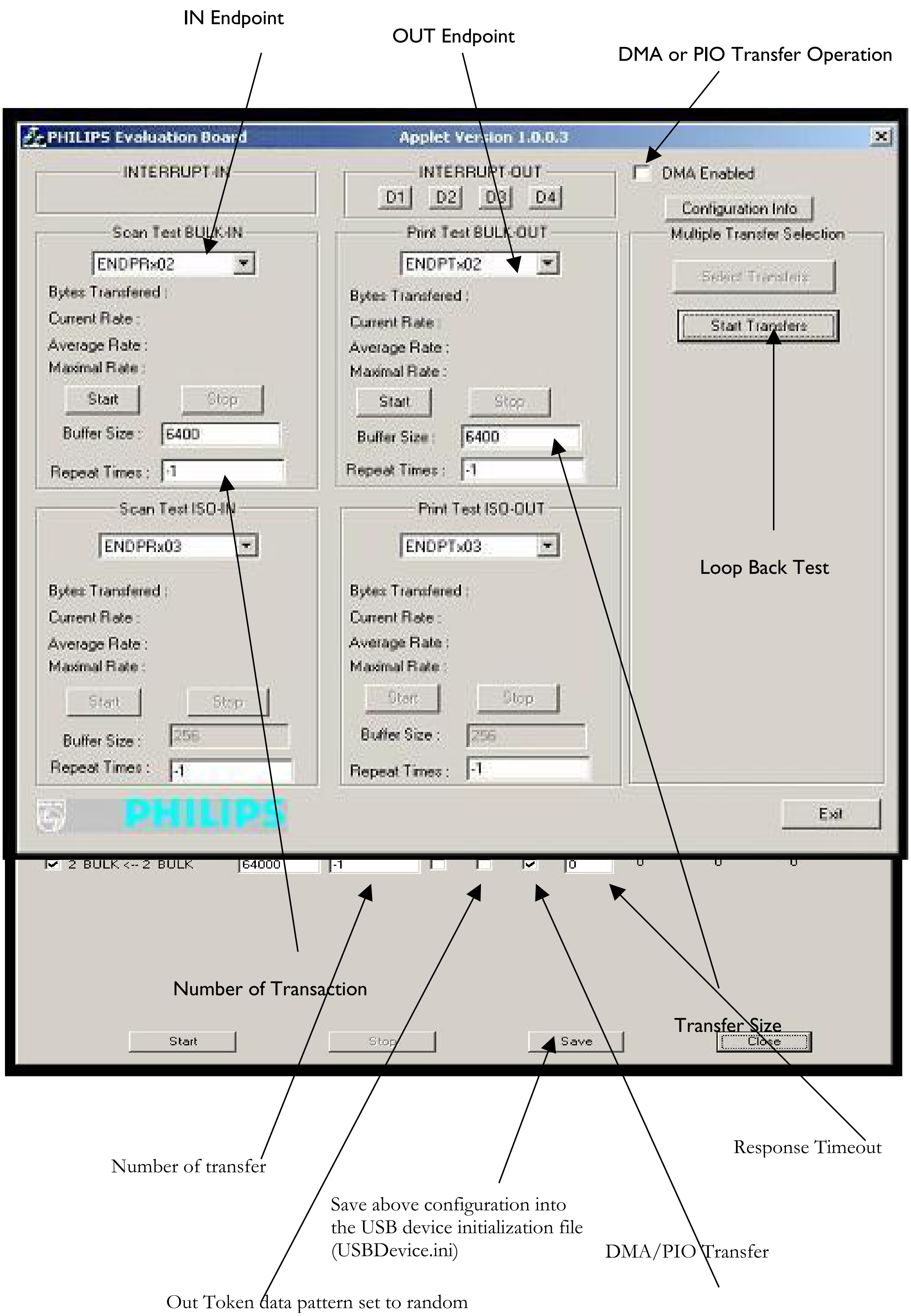
Figure 8-2: USB Full-Speed Device on Intel UHCI USB 1.1 Controller

8.3. PCI Kit Test Application

This application allows you to perform data transfer using the GDMA slave mode of the ISP1582. It is a simple application that will send data to the ISP1582 and read back the data. Data integrity is checked with this application.






IN Endpoint
OUT Endpoint
DMA or PIO Transfer Operation
PHILIPS Evaluation Board
Applet Version 1.0.0.3
INTERRUPT-IN
INTERRUPT-OUT
D1
D2
D3
D4
DMA Enabled
Configuration Info
Multiple Transfer Selection
Select Transfers
Start Transfers
Scan Test BULK-IN
ENDPRx02
Bytes Transfered :
Current Rate :
Average Rate :
Maximal Rate :
Start
Stop
Buffer Size :
6400
Repeat Times :
-1
Print Test BULK-OUT
ENDPTx02
Scan Test ISO-IN
ENDPRx03
256
Print Test ISO-OUT
ENDPTx03
Loop Back Test
PHILIPS
Exit
2 BULK <-- 2 BULK
64000
0
Number of Transaction
Transfer Size
Save
Close
Response Timeout
Number of transfer
Save above configuration into the USB device initialization file (USBDevice.ini)
DMA/PIO Transfer
Out Token data pattern set to random

UM10036_1

Table 8-1: Endpoint Description

Pipe Number	Endpoint Type	Operations
5*	Iso-Out	This pipe is defined as Isochronous Out pipe. Test applet and the ISP1582 evaluation board supports 3 test modes: loop-back mode, print mode and scan mode. The firmware uses I/O accesses on this endpoint.
4*	Iso-In	This pipe is defined as Isochronous In pipe. Test applet and the ISP1582 evaluation board supports 3 test modes: loop-back mode, print mode and scan mode. The firmware uses I/O accesses on this endpoint.
2*	Bulk-Out	This pipe is defined as Bulk Out pipe. Test applet and the ISP1582 evaluation board supports 3 test modes: loop-back mode, print mode and scan mode. The firmware uses I/O accesses on this endpoint.
3*	Bulk-In	This pipe is defined as Bulk In pipe. Test applet and the ISP1582 evaluation board supports 3 test modes: loop-back mode, print mode and scan mode. The firmware uses I/O accesses on this endpoint.

* Pipe number is not ENDPOINT NUMBER, the value may vary if you have different endpoint configuration.

Three test modes:

1. Scan mode: The ISP1582 evaluation board acts like a scanner. It sends data packets to the host PC as fast as possible. This mode is used to evaluate the Isochronous In and bulk In transfer rate.
2. Print mode: The ISP1582 evaluation board acts like a printer. It receives data packets from the host PC as fast as possible. This mode is used to evaluate the Isochronous Out and Bulk Out transfer rate.
3. Loop back mode: In this mode, the ISP1582 evaluation board receives data packets on Isochronous/Bulk Out endpoint and sends them back to the host PC on Isochronous/Bulk In endpoint. This mode is used to test the data integrity of transfers.

The "Buffer Size" setting on the test applet is determined by the firmware and hardware ability of the evaluation board. For PCI kit, the maximal size is limited to 64000 for Bulk transfer.

The "Repeat Times" for loop-back test controls the numbers of iterations of loop-back, which is useful for debugging. "-1" means it is infinite.

8.4. PCI Kit Resources

The INTA of the PCI bus is utilized to reflect the ISP1582 interrupt.

8.4.1. I/O Address

Table 8-2: I/O Mapping

I/O Address	Alignment	Operation	Function
PCI Base Address	Word (16 bit) Address Port	Write only	Address port. Since PCI cannot provide continuous address for the ISP1582, this port is used to latch the address for the ISP1582 register. Write on this port latches the lower byte of PCI data to the ISP1582 address lines. Read is not valid.
PCI Base Address + 2	Word (16 bit) Data Port	Write/Read	Data port. Read/write on this port causes operation on ISP1582 registers with the address at the address port.
PCI Base Address + 4	Word (16 bit) Ctrl Port	Write only	Control port. Operations on this port initiate the enable and disable operation on interrupt signal of ISP1582.
PCI Base Address + 6	Word (16 bit) DMA Port	Write only	DMA port. To program on board DMA Controller.

The operation behavior of an odd address is unpredictable. Byte access is not allowed.

The OS locates the PCI base address upon power on and the PCI kit firmware will initiate a PCI-find-bridge routine, which will determine the PCI IO base address.

There are four ports that are utilized by the firmware. The Address port is a gateway for the PCI bus to assign the register address to the ISP1582. The Data port holds the 16-bit data that is to be written to or read out from the ISP1582 register. The Control port has only a single function: to mask the interrupt from the ISP1582. This allows the processor to branch to the interrupt service routine to clear the respective interrupt. The DMA port controls the function of the external master DMA controller.

8.4.2. Control and DMA Port Bit

Table 8-3: Control Port Bit Definition

Bit	Definition	Operation	Description
15 to 8	Not valid	—	—
7	Interrupt enable	Write only	If this bit is logic 1 and the ISP1582 is also 1, INTA will be asserted. If this bit is logic 0, INTA is not asserted.
6 to 0	Not valid	—	—

The control port is used to mask off the interrupt from ISP1582. By masking the interrupt, allow the firmware to branch to the interrupt service routine and to clear the individual interrupt source.

Table 8-4 DMA Port Bit Definition

Bit	Definition	Operation	Description
15 to 3	Not valid	--	--
2	DMA Read Write Operation	Write only	Defines DMA Read Write Operation: **1**—DMA Read Operation **0**—DMA Write Operation
1	DMA Start Operation	Write only	Defines DMA operation: **1**—DMA Start Operation **0**—DMA Stop Operation
0	DMA Reset	Write only	This bit set or reset the DMA controller. It will also reset the SRAM address.

The DMA port controls the external master DMA controller. Bit 0, if it is set to logic zero, will initiate a reset procedure to the DMA controller. The DMA start bit controls the start and stop of the DMA transfer between the ISP1582 and the DMA controller. The read or write direction of the data transfer is reflected by the DMA RDWR bit.

8.4.3. Signal Configuration

Table 8-5: Signals and Program Level

ISP1582	Active level	Mode	Description
RD/WR	0	Rising edge	Asserted only at PIO access.
DIOR/DIOW	0	Rising edge	Asserted only at the DMA mode.
DREQ	1	Level	Default to active high level via the DMA hardware register of ISP1582
DACK	0	Level	Default to active low level via the DMA hardware register of ISP1582
INT	0	Level	Program to level trigger via the interrupt configuration register of ISP1582

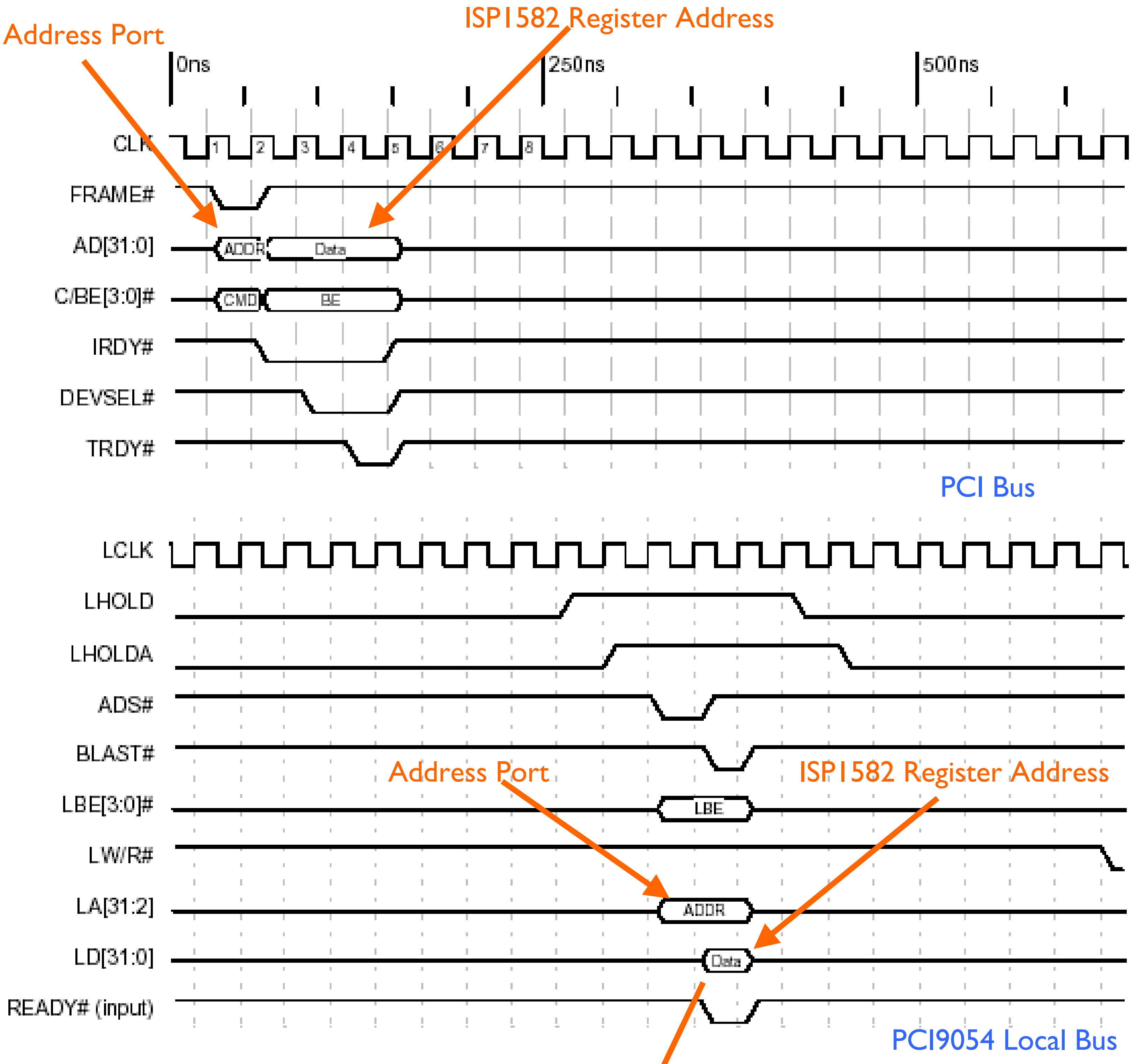
8.4.4. DMA Resources and Control Mode

When the ISP1582 asserts a DMA request, it sets the DREQ. Upon detection of the assertion of DREQ, the onboard DMA controller will assert DACK. The DMA controller will then start the DMA transfer by toggling the DIOR/DIOW strobe signal.

8.4.5. Configure DMA Port

The DMA controller is controlled by the DMA port, which is under the influence of the firmware. Therefore, the start, the reset, and the read/write operations are all under the control of the firmware. The DMA port has the bit 0 set to the DMA reset command, bit 1 to the DMA start command, and bit 2 to the DMA read/write command. Therefore, by accessing the DMA port, the DMA controller is configured to start or reset operation and the direction of the data transfer.

8.4.6. ISP1582 Address Port Access Diagram



Timing Diagram 5-24. PCI Target Single Write (32-Bit Local Bus)

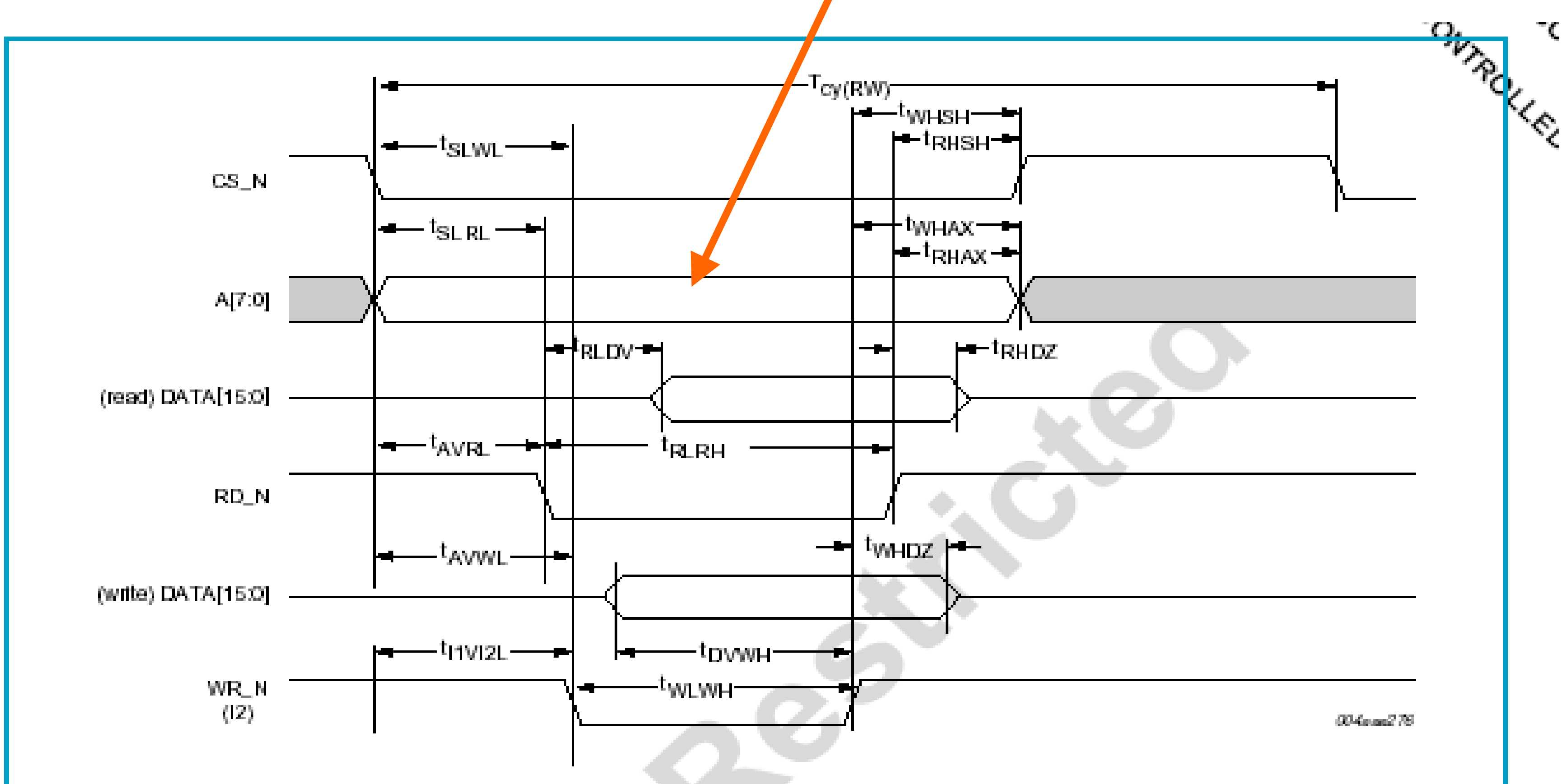
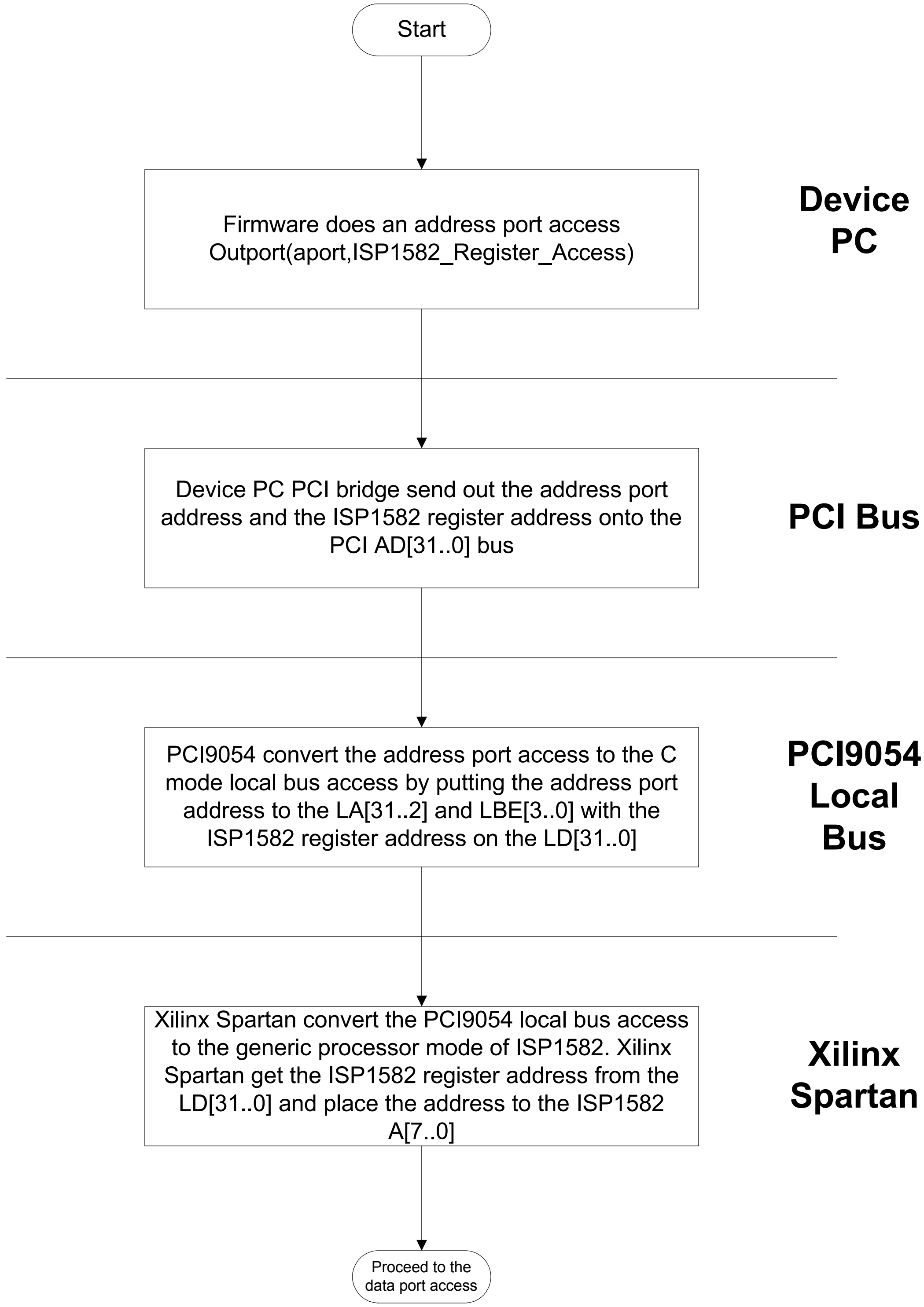
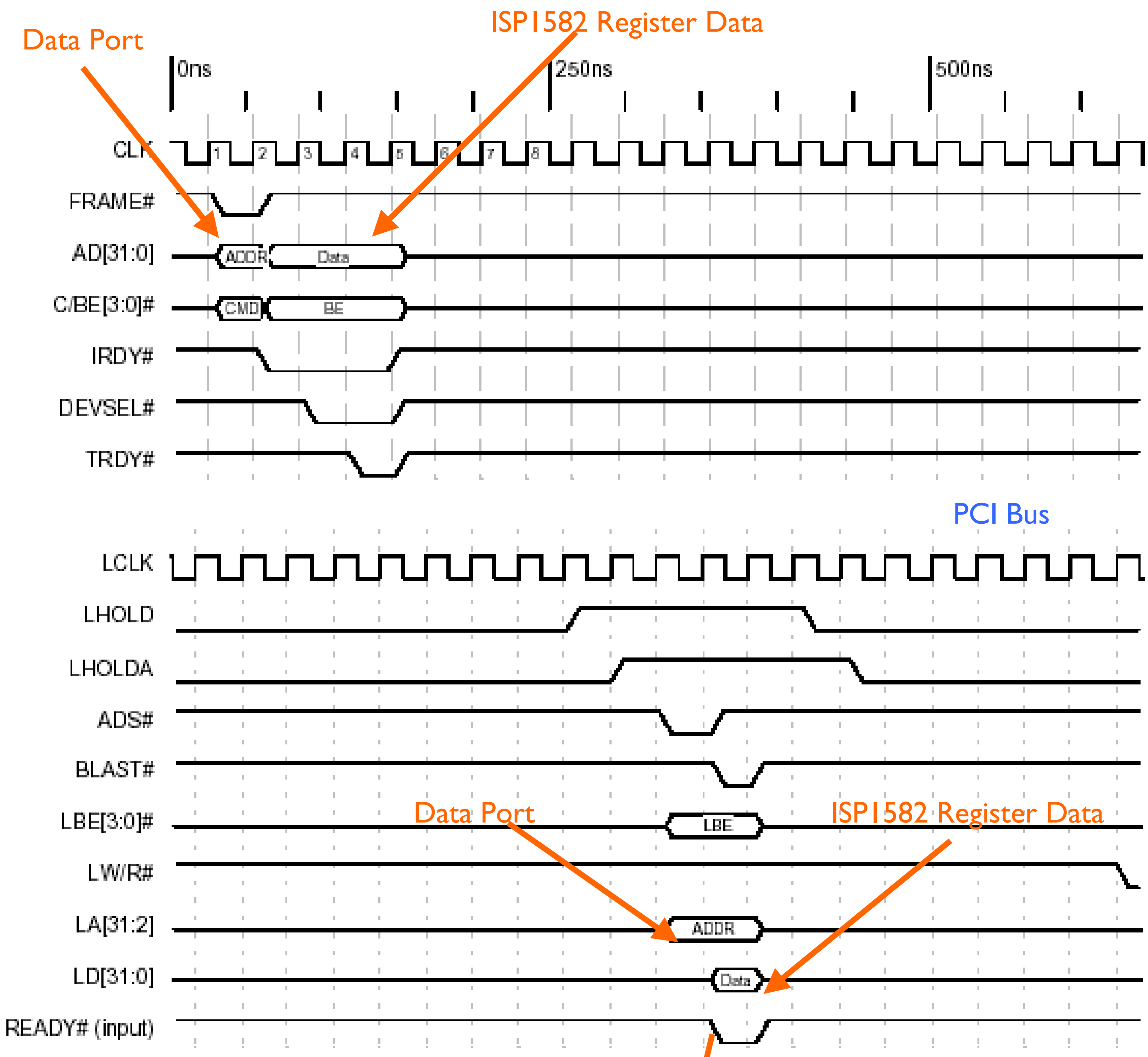


Fig 11. ISP1582 register access timing: separate address and data buses.


Start
Firmware does an address port access Outport(aport,ISP1582_Register_Access)
Device PC
Device PC PCI bridge send out the address port address and the ISP1582 register address onto the PCI AD[31..0] bus
PCI Bus
PCI9054 convert the address port access to the C mode local bus access by putting the address port address to the LA[31..2] and LBE[3..0] with the ISP1582 register address on the LD[31..0]
PCI9054 Local Bus
Xilinx Spartan convert the PCI9054 local bus access to the generic processor mode of ISP1582. Xilinx Spartan get the ISP1582 register address from the LD[31..0] and place the address to the ISP1582 A[7..0]
Xilinx Spartan
Proceed to the data port access

8.4.7. ISP1582 Data Port Access Diagram



Timing Diagram 5-24. PCI Target Single Write (32-Bit Local Bus)

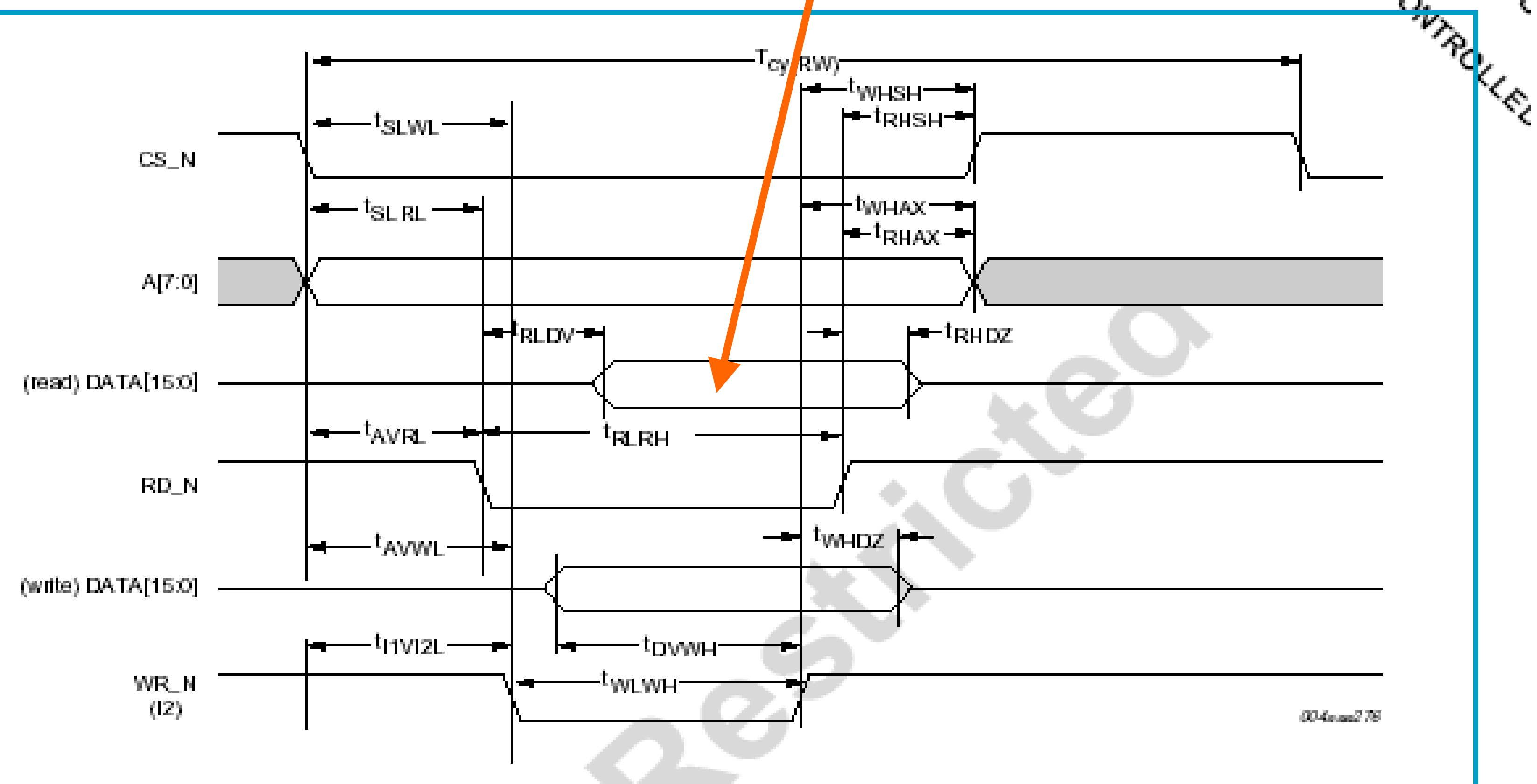
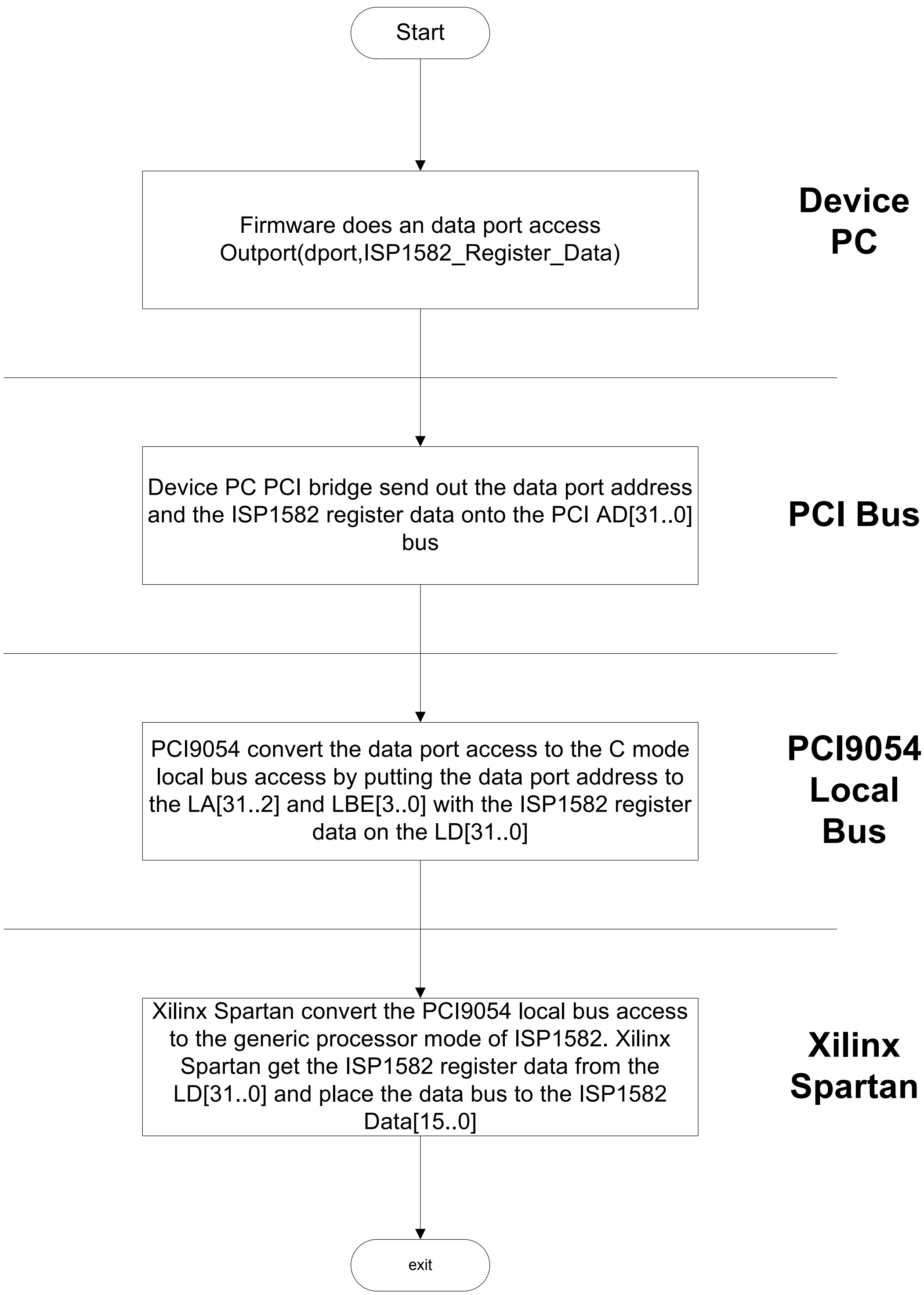
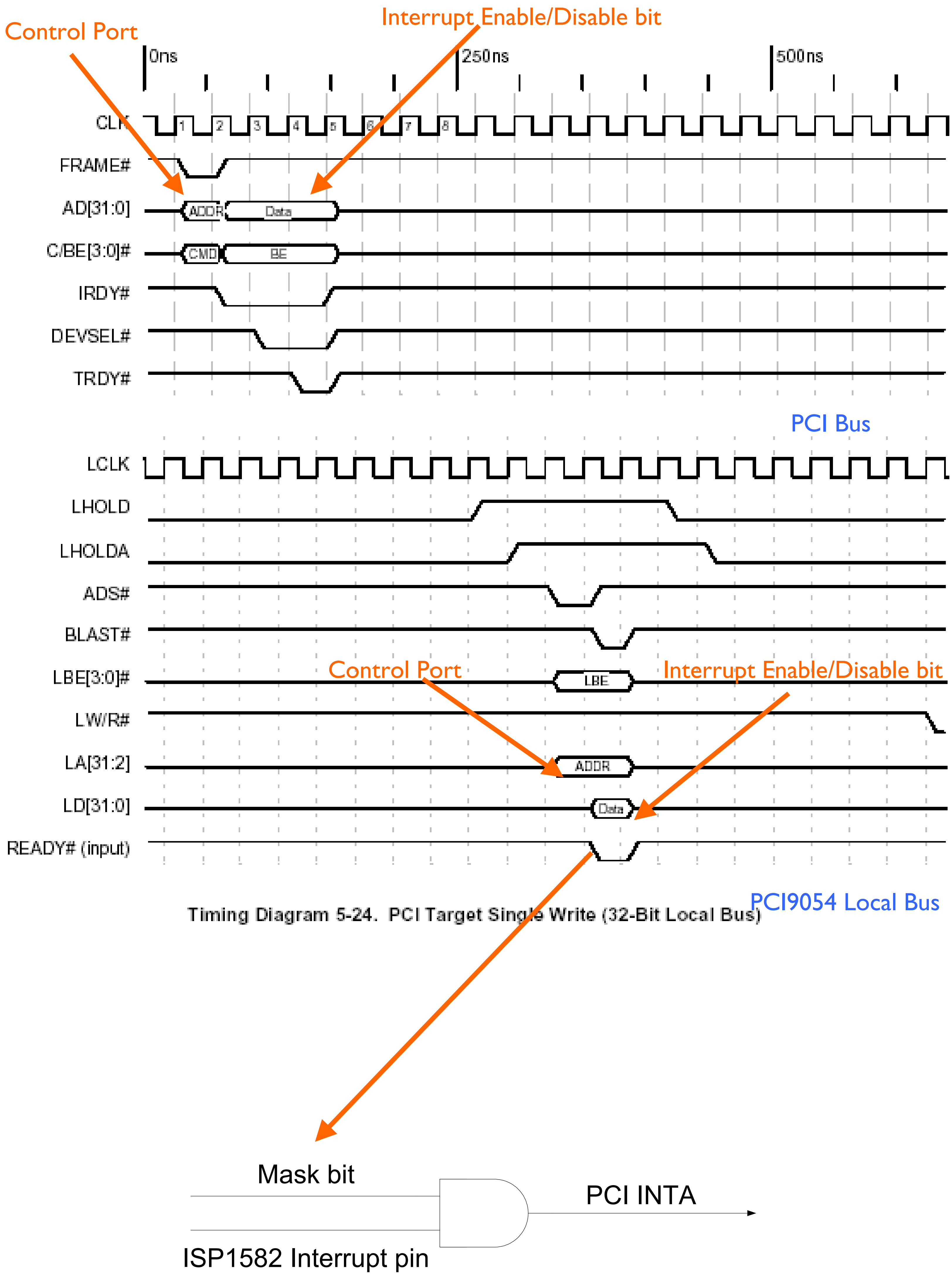


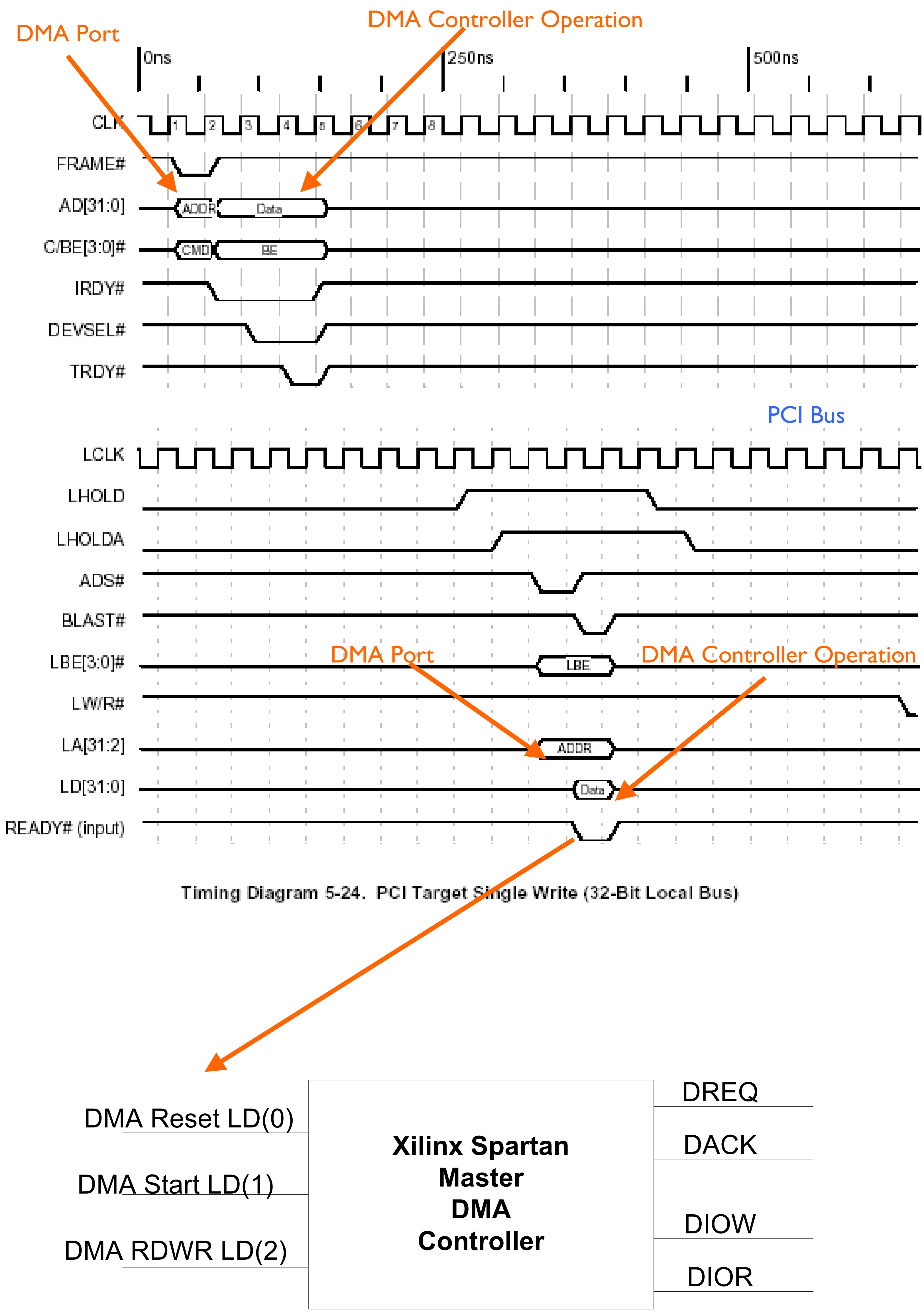
Fig 11. ISP1582 register access timing: separate address and data buses.


Start
Firmware does an data port access
Outport(dport,ISP1582_Register_Data)
Device PC
Device PC PCI bridge send out the data port address and the ISP1582 register data onto the PCI AD[31..0] bus
PCI Bus
PCI9054 convert the data port access to the C mode local bus access by putting the data port address to the LA[31..2] and LBE[3..0] with the ISP1582 register data on the LD[31..0]
PCI9054 Local Bus
Xilinx Spartan convert the PCI9054 local bus access to the generic processor mode of ISP1582. Xilinx Spartan get the ISP1582 register data from the LD[31..0] and place the data bus to the ISP1582 Data[15..0]
Xilinx Spartan
exit

8.4.8. Xilinx Spartan Control Port Access Diagram



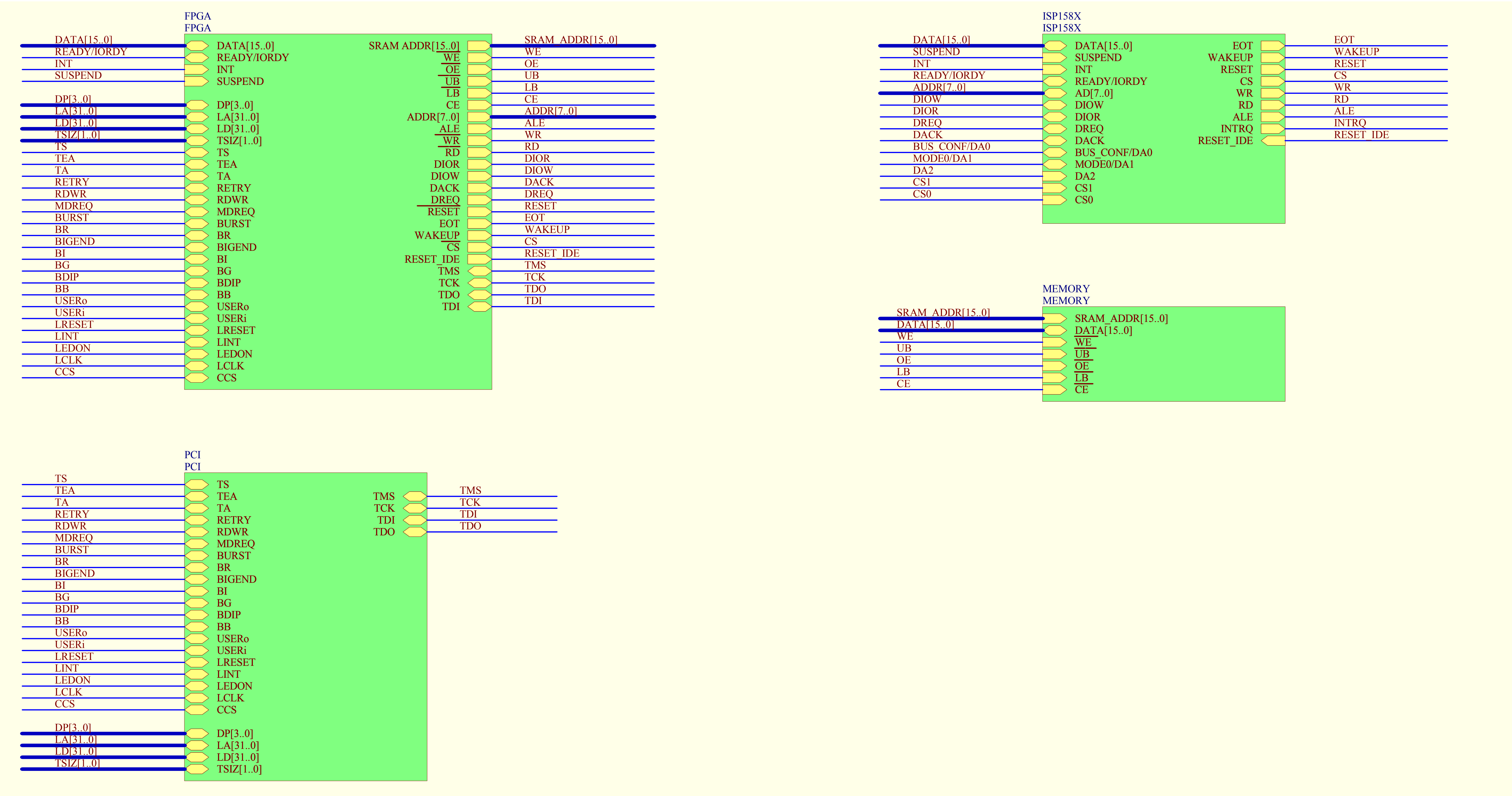
8.4.9. Xilinx Spartan Master DMA Controller DMA port Access Diagram

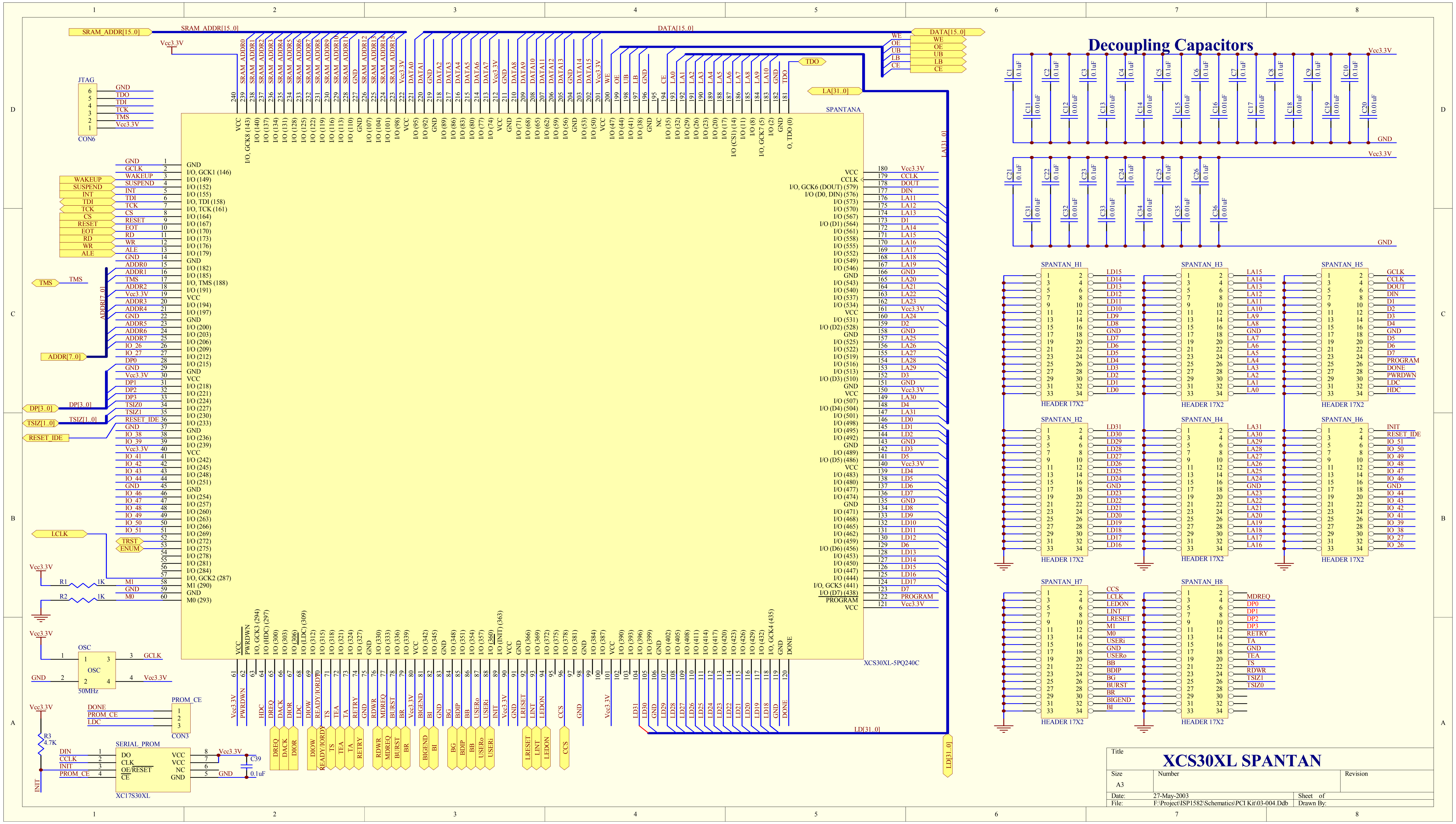


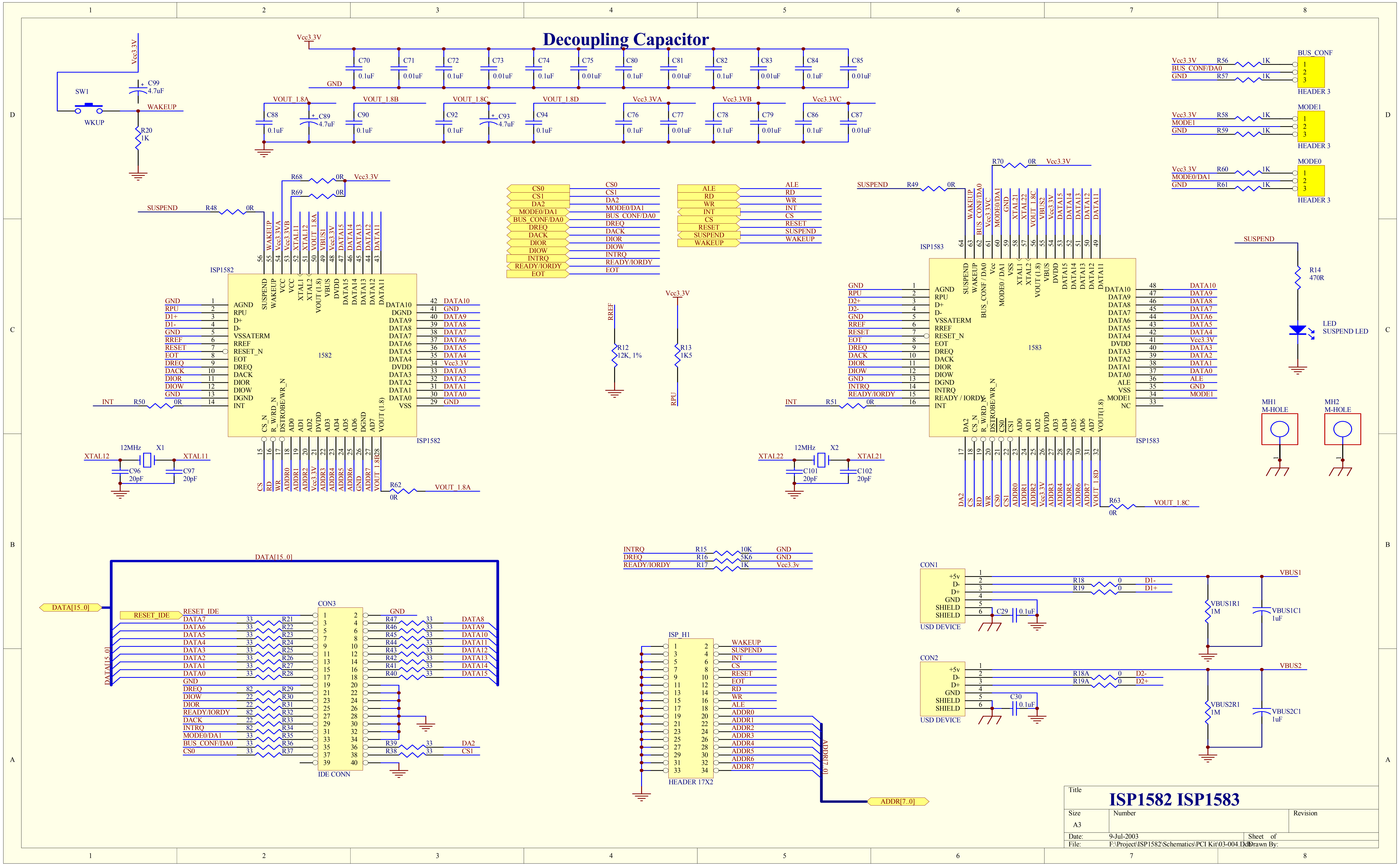
Timing Diagram 5-24. PCI Target Single Write (32-Bit Local Bus)

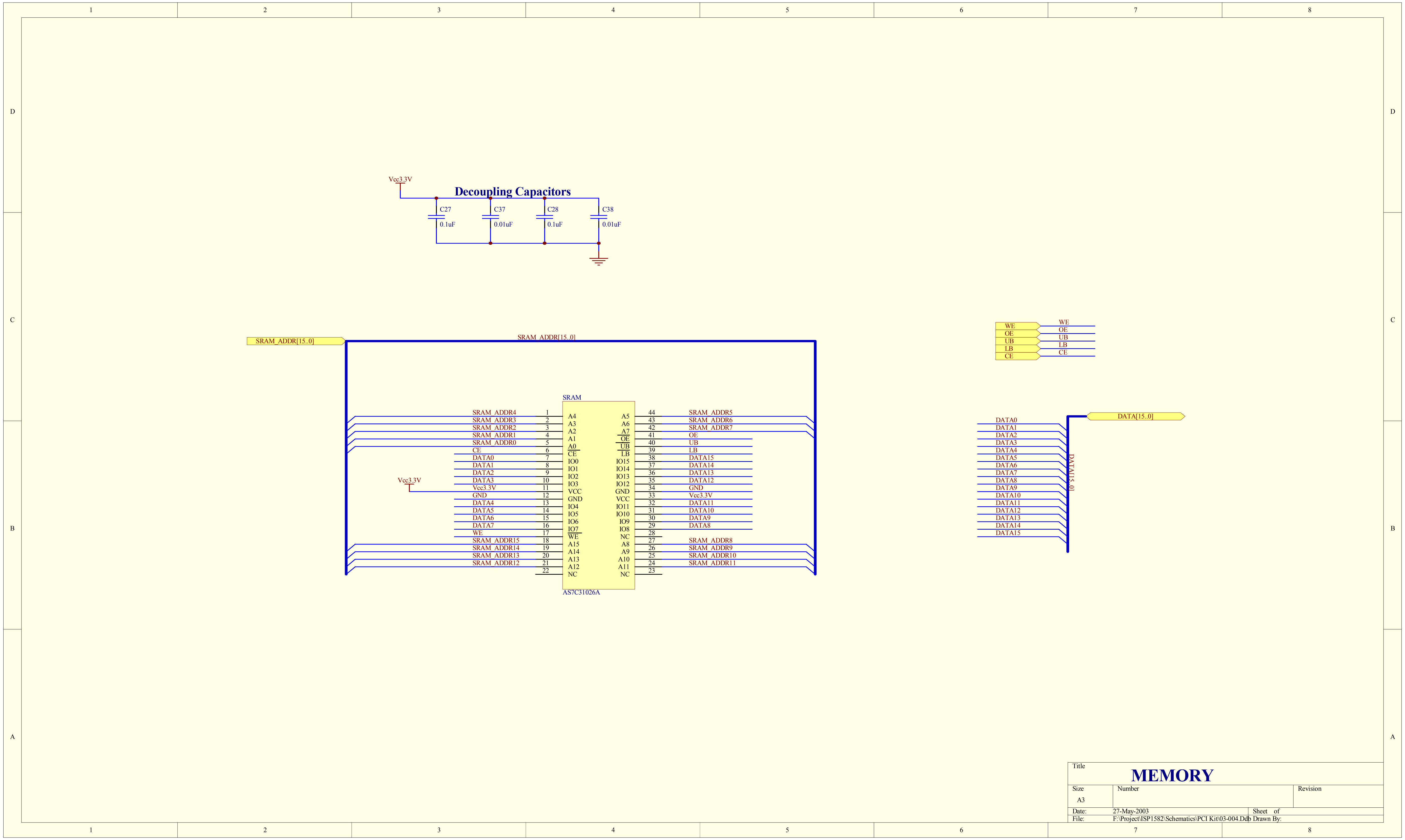
9. Schematics of Eval Board

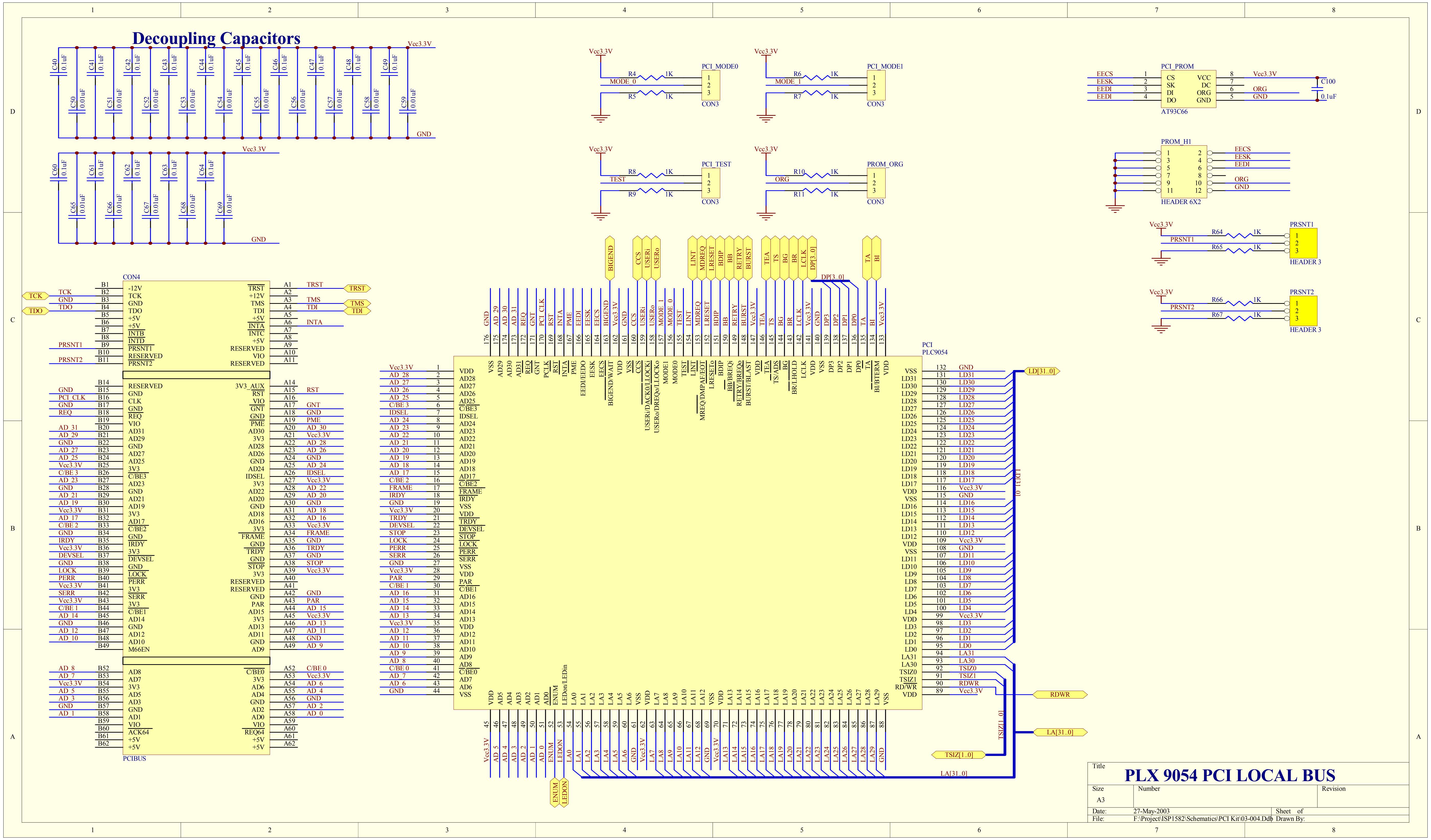
9.1. ISP1582 PCI Board




Decoupling Capacitors
XCS30XL SPANTAN
SPANTANA
XCS30XL-5PQ240C
SERIAL_PROM
XC17S30XL
JTAG
CON6
CON3
OSC
50MHz
Title
Size
A3
Number
Revision
Date: 27-May-2003
File: F:\Project\ISP1582\Schematics\PCI Kit\03-004.Ddb
Sheet of
Drawn By:


Decoupling Capacitor
ISP1582
ISP1583
IDE CONN
HEADER 17X2
USD DEVICE
Title
ISP1582 ISP1583
Size
A3
Number
Revision
Date: 9-Jul-2003
Sheet of
File: F:\Project\ISP1582\Schematics\PCI Kit\03-004.Ddb
Drawn By:


Decoupling Capacitors
Vcc3.3V
C27 0.1uF
C37 0.01uF
C28 0.1uF
C38 0.01uF
SRAM_ADDR[15..0]
SRAM
AS7C31026A
WE
OE
UB
LB
CE
DATA[15..0]
MEMORY
Title
Size A3
Number
Revision
Date: 27-May-2003
Sheet of
File: F:\Project\ISP1582\Schematics\PCI Kit\03-004.Ddb Drawn By:


Decoupling Capacitors
CON4
PCIBUS
PCI_MODE0
PCI_MODE1
PCI_TEST
PROM_ORG
PCI_PROM
AT93C66
PROM_H1
HEADER 6X2
PRSNT1
PRSNT2
HEADER 3
PCI
PLC9054
Title
PLX 9054 PCI LOCAL BUS
Size
A3
Number
Revision
Date: 27-May-2003
Sheet of
File: F:\Project\ISP1582\Schematics\PCI Kit\03-004.Ddb
Drawn By:

10. Bill Of Material

10.1. ISP1582 PCI Kit Main Board

Table 10-1: Build of Material of ISP1582 Main Board

Part Type	Designator	Footprint
0	R19A	0603R
0	R18A	0603R
0	R18	0603R
0	R19	0603R
0.01uF	C83	0603C
0.1uF	C84	0603C
0.1uF	C82	0603C
0.1uF	C80	0603C
0.01uF	C81	0603C
0.01uF	C85	0603C
0.1uF	C90	0603C
0.1uF	C92	0603C
0.1uF	C88	0603C
0.1uF	C86	0603C
0.01uF	C87	0603C
0.1uF	C70	0603C
0.01uF	C71	0603C
0.1uF	C72	0603C
0.1uF	C60	0603C
0.01uF	C59	0603C
0.01uF	C58	0603C
0.01uF	C73	0603C
0.01uF	C77	0603C
0.1uF	C78	0603C
0.01uF	C79	0603C
0.1uF	C74	0603C
0.01uF	C75	0603C
0.1uF	C76	0603C
0.1uF	C94	0603C
0.1uF	C43	0603C
0.1uF	C42	0603C
0.1uF	C41	0603C
0.1uF	C46	0603C
0.1uF	C45	0603C
0.1uF	C44	0603C
0.1uF	C28	0603C
0.1uF	C27	0603C
0.1uF	C29	0603C

Part Type	Designator	Footprint
0.1uF	C40	0603C
0.01uF	C38	0603C
0.01uF	C37	0603C
0.01uF	C55	0603C
0.01uF	C54	0603C
0.01uF	C53	0603C
0.1uF	C30	0603C
0.01uF	C57	0603C
0.01uF	C56	0603C
0.1uF	C49	0603C
0.1uF	C48	0603C
0.1uF	C47	0603C
0.01uF	C52	0603C
0.01uF	C51	0603C
0.01uF	C50	0603C
0.01uF	C17	0603C
0.01uF	C18	0603C
0.01uF	C19	0603C
0.01uF	C14	0603C
0.01uF	C15	0603C
0.01uF	C16	0603C
0.01uF	C20	0603C
0.1uF	C24	0603C
0.1uF	C25	0603C
0.1uF	C26	0603C
0.1uF	C21	0603C
0.1uF	C22	0603C
0.1uF	C23	0603C
0.1uF	C4	0603C
0.1uF	C5	0603C
0.1uF	C6	0603C
0.1uF	C1	0603C
0.1uF	C2	0603C
0.1uF	C3	0603C
0.1uF	C7	0603C
0.01uF	C11	0603C
0.01uF	C12	0603C
0.01uF	C13	0603C
0.1uF	C8	0603C
0.1uF	C9	0603C
0.1uF	C10	0603C
0.01uF	C66	0603C
0.01uF	C65	0603C
0.01uF	C68	0603C
0.01uF	C67	0603C

Part Type	Designator	Footprint
0.1uF	C64	0603C
0.1uF	C62	0603C
0.1uF	C61	0603C
0.1uF	C39	0603C
0.1uF	C63	0603C
0.01uF	C69	0603C
0.01uF	C34	0603C
0.01uF	C35	0603C
0.01uF	C32	0603C
0.01uF	C33	0603C
0.01uF	C31	0603C
0.1uF	C100	0603C
0.01uF	C36	0603C
0R	R51	0603R
0R	R62	0603R
0R	R63	0603R
0R	R49	0603R
0R	R70	0603R
0R	R69	0603R
0R	R68	0603R
0R	R50	0603R
0R	R48	0603R
1K5	R13	0603R
1K	R17	0603R
1K	R1	0603R
1K	R56	0603R
1K	R65	0603R
1K	R61	0603R
1K	R67	0603R
1K	R66	0603R
1K	R58	0603R
1K	R57	0603R
1K	R60	0603R
1K	R59	0603R
1K	R7	0603R
1K	R10	0603R
1K	R5	0603R
1K	R6	0603R
1K	R8	0603R
1K	R9	0603R
1K	R11	0603R
1K	R20	0603R
1K	R64	0603R
1K	R4	0603R
1K	R2	0603R

Part Type	Designator	Footprint
1M	VBUS2R1	
1M	VBUS1R1	
1uF	VBUS1C1	
1uF	VBUS2C1	
4.7K	R3	0603R
4.7uF	C99	CASE-A
4.7uF	C93	CASE-A
4.7uF	C89	CASE-A
5K6	R16	0603R
10K	R15	0603R
12K, 1%	R12	0603R
12MHz	X1	XTAL-HC49/4H
12MHz	X2	XTAL-HC49/4H
20pF	C97	0603C
20pF	C102	0603C
20pF	C96	0603C
20pF	C101	0603C
22	R33	0603R
22	R31	0603R
22	R30	0603R
33	R28	0603R
33	R26	0603R
33	R38	0603R
33	R27	0603R
33	R23	0603R
33	R24	0603R
33	R21	0603R
33	R22	0603R
33	R39	0603R
33	R46	0603R
33	R44	0603R
33	R40	0603R
33	R47	0603R
33	R42	0603R
33	R41	0603R
33	R45	0603R
33	R43	0603R
33	R36	0603R
33	R37	0603R
33	R35	0603R
33	R25	0603R
50MHz	OSC	XTAL-SMD4
82	R29	0603R
82	R32	0603R
82	R34	0603R

Part Type	Designator	Footprint
470R	R14	0603R
AS7C31026A	SRAM	TSOP2-44P
AT93C66	PCI_PROM	DIP8
CON3	PROM_ORG	SIP3
CON3	PROM_CE	SIP3
CON3	PCI_TEST	SIP3
CON3	PCI_MODE0	SIP3
CON3	PCI_MODE1	SIP3
CON6	JTAG	SIP6
HEADER 3	MODE0	SIP3
HEADER 3	PRSNT2	SIP3
HEADER 3	PRSNT1	SIP3
HEADER 3	MODE1	SIP3
HEADER 3	BUS_CONF	SIP3
HEADER 6X2	PROM_H1	IDC12
HEADER 17X2	SPANTAN_H7	IDC34
HEADER 17X2	SPANTAN_H8	IDC34
HEADER 17X2	SPANTAN_H6	IDC34
HEADER 17X2	SPANTAN_H5	IDC34
HEADER 17X2	SPANTAN_H4	IDC34
HEADER 17X2	SPANTAN_H2	IDC34
HEADER 17X2	SPANTAN_H1	IDC34
HEADER 17X2	SPANTAN_H3	IDC34
HEADER 17X2	ISP_H1	IDC34
IDE CONN	CON3	IDC40
ISP1582	ISP1582	HVQFN56/P.5N
ISP1583	ISP1583	HVQFN64/P.5N
M-HOLE	MH2	M-HOLE2
M-HOLE	MH1	M-HOLE2
PCIBUS	CON4	PCIBus2
PLC9054	PCI	F-QFP176/P.5N
SUSPEND LED	LED	LED3
USD DEVICE	CON2	USB-TYPEB
USD DEVICE	CON1	USB-TYPEB
WKUP	SW1	
XC17S30XL	SERIAL_PROM	DIP8
XCS30XL-5PQ240C	SPANTAN	F-QFP32X32-G240/P.5N

11. Xilinx XCS30XL DMA Controller VHDL Code

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;


entity PCIKit is
    port (

            -- Global Clock

            GCLK:               in STD_LOGIC;

            -- PCI

        TRST:                   out STD_LOGIC;

            -- ISP1582

        WAKEUP:         in STD_LOGIC;
        SUSPEND:        in STD_LOGIC;
        INT:                    in STD_LOGIC;
        CS:                     out STD_LOGIC;
        RESET:                  out STD_LOGIC;
        EOT:                    out STD_LOGIC;
        RD:                     out STD_LOGIC;
        WR:                     out STD_LOGIC;
        ALE:                    out STD_LOGIC;
        ADDR:                   out STD_LOGIC_VECTOR (7 downto 0);
        RESET_IDE:              out STD_LOGIC;
        DREQ:                   in STD_LOGIC;
        DACK:                   inout STD_LOGIC;
        DIOR:                   out STD_LOGIC;
        DIOW:                   out STD_LOGIC;
        READY:                  in STD_LOGIC;


            -- PLX9054

        ENUM:                   in STD_LOGIC;
        LCLK:                   out STD_LOGIC;
        DP:                     inout STD_LOGIC_VECTOR (3 downto 0);
        LBE:                    in STD_LOGIC_VECTOR (3 downto 0);
        ADS:                    in STD_LOGIC;
        LSERR:                  in STD_LOGIC;
        LREADY:         out STD_LOGIC;
        BREQo:                  in STD_LOGIC;
        LWR:                    in STD_LOGIC;
        MDREQ:                  in STD_LOGIC;
--      BLAST:                  in STD_LOGIC;
        LHOLD:                  in STD_LOGIC;
        BIGEND:         out STD_LOGIC;
        BTERM:                  inout STD_LOGIC;
```

```
            LHOLDA:             out STD_LOGIC;
            LWAIT:              inout STD_LOGIC;
            BREQi:              OUT STD_LOGIC;
            USERo:              in STD_LOGIC;
            USERi:              out STD_LOGIC;
            LRESET:         in STD_LOGIC;
            LINT:               out STD_LOGIC;
            LEDON:              inout STD_LOGIC;
            CCS:                out STD_LOGIC;
            LD:                 inout STD_LOGIC_VECTOR (31 downto 0);
            LA:                 in STD_LOGIC_VECTOR (31 downto 2);


              -- SRAM
            CE:                 out STD_LOGIC;
            LB:                 out STD_LOGIC;
            UB:                 out STD_LOGIC;
            OE:                 out STD_LOGIC;
            WE:                 out STD_LOGIC;
            DATA:               inout STD_LOGIC_VECTOR (15 downto 0);
            SRAM_ADDR:          out STD_LOGIC_VECTOR (15 downto 0)
  );
end PCIKit;

architecture PCIKit_arch of PCIKit is

type op_code is (S0,S1,S2,S3,S4,S5);
signal STATE,ADDR_STATE,DATA_STATE,CNTRL_STATE,DMA_STATE: op_code;

constant asserted,read,master:                  STD_LOGIC := '1';
constant deasserted,write,slave:            STD_LOGIC := '0';

signal ADDR_COUNTER: integer range 0 to 65536;
signal address_decode,ADDR_INC:         std_logic;
signal DMA_START,DMA_RDWR,DMA_RESET: STD_LOGIC;

signal ISP1582_Address:   STD_LOGIC_VECTOR(7 downto 0);
signal ISP1582_Address_Port,ISP1582_Data_Port,Control_Port,DMA_Port: STD_LOGIC;
signal Address_Ready,Data_Ready,Control_Ready,DMA_Ready: STD_LOGIC;
signal LOCAL_RD,LOCAL_WR,INT_EN: STD_LOGIC;
signal MODE: STD_LOGIC_VECTOR(1 downto 0);

begin

      ISP1582_Address <= LA(7 downto 2) & LBE(1 downto 0);

      TRST    <= '1';


    CS <= not LHOLD when ISP1582_Data_Port = asserted else asserted;
    RESET <= '1';
    EOT <= '0';
    ALE <= '0';
    RESET_IDE <= '1';
```

```
        LCLK <= GCLK;
        DP <= (others => 'Z');


            DATA <= LD(15 downto 0) when LWR = asserted and ISP1582_Data_Port = asserted else (others =>
    'Z');
            LD(15 downto 0) <= DATA when LWR = deasserted and ISP1582_Data_Port = asserted else (others =>
    'Z');


            LREADY <= Address_Ready and Data_Ready and Control_Ready and DMA_Ready;


    ISP1582_Control_Process:

            process(GCLK,LRESET,Control_Port)
            begin

                if LRESET = '0' then

                    CNTRL_STATE <= S0;
                    Control_Ready <= '1';

                elsif rising_edge(GCLK) then


                    case CNTRL_STATE is

                    when S0 =>
                        if Control_Port = '1' then
                            CNTRL_STATE <= S1;
                        end if;

                    when S1 =>

                        if Control_Port = '1' then
                            Control_Ready <= '0';
                            INT_EN <= LD(7);

                        end if;

                        CNTRL_STATE <= S2;

                    when S2 =>

                        CNTRL_STATE <= S3;

                    when others =>

                        Control_Ready <= '1';

                        if Control_Port = '0' then
                            CNTRL_STATE <= S0;
                        end if;
                    end case;
                end if;
```

```
        end process;

    ISP1582_DMA_Process:

        process(GCLK,LRESET,DMA_Port)
        begin

            if LRESET = '0' then

                DMA_STATE <= S0;
                DMA_Ready <= '1';
                DMA_RESET <= deasserted;
                DMA_START <= deasserted;
                DMA_RDWR <= deasserted;
                MODE <= "00";

            elsif rising_edge(GCLK) then


                case DMA_STATE is

                when S0 =>
                    if DMA_Port = '1' then
                        DMA_STATE <= S1;
                        DMA_RESET <= LD(0);
                        DMA_START <= LD(1);
                        DMA_RDWR <= LD(2);
                        MODE <= LD(4 downto 3);
                    end if;

                when S1 =>

                    if DMA_Port = '1' then
                        DMA_Ready <= '0';
                    end if;

                    DMA_STATE <= S2;

                when S2 =>

                    DMA_STATE <= S3;

                when others =>

                    DMA_Ready <= '1';

                    if DMA_Port = '0' then
                        DMA_STATE <= S0;
                    end if;
                end case;
            end if;
        end process;


    ISP1582_Register_Address_Process:
```

```
        process(GCLK,LRESET,ISP1582_Address_Port)
        begin

                if LRESET = '0' then

                        ADDR_STATE <= S0;
                        Address_Ready <= '1';
                        ADDR <= "11111111";

                elsif rising_edge(GCLK) then


                        case ADDR_STATE is

                        when S0 =>
                                if ISP1582_Address_Port = '1' then
                                        ADDR <= LD(7 downto 0);
                                        ADDR_STATE <= S1;
                                end if;

                        when S1 =>

                                if ISP1582_Address_Port = '1' then
                                        Address_Ready <= '0';
                                end if;

                                ADDR_STATE <= S2;

                        when S2 =>

                                ADDR_STATE <= S3;

                        when others =>

                                Address_Ready <= '1';

                                if ISP1582_Address_Port = '0' then
                                        ADDR_STATE <= S0;
                                end if;
                        end case;
                end if;
        end process;

DATA_READY_Process:

        process(GCLK,LRESET,DATA_STATE)
        begin

                if LRESET = '0' then

                        Data_Ready <= '1';

                elsif rising_edge(GCLK) then

                        case DATA_STATE is
```

```
                    when S0 =>
                            Data_Ready <= '1';
                    when S1 =>
                            Data_Ready <= '0';
                    when S2 =>
                            Data_Ready <= '0';
                    when S3 =>
                            Data_Ready <= '1';
                    when S4 =>
                            Data_Ready <= '1';
                    when others =>
                            Data_Ready <= '1';


                    end case;
            end if;
    end process;


ISP1582_Register_Data_Process:

    process(LWR,ISP1582_Data_Port,GCLK,LRESET)
    begin


            if LRESET = '0' then

                    WR <= '1';
                    RD <= '1';
                    DATA_STATE <= S0;

            elsif rising_edge(GCLK) then


                    case DATA_STATE is

                    when S0 =>
                                    if LWR = '1' and ISP1582_Data_Port = '1' then -- write operation start
                                            WR <= '0';
                                            RD <= '1';
                                            DATA_STATE <= S1;

                                    elsif LWR = '0' and ISP1582_Data_Port = '1' then -- read operation
start
                                            RD <= '0';
                                            WR <= '1';
                                            DATA_STATE <= S1;

                                    else
                                            DATA_STATE <= S0;
                                            RD <= '1';
                                            WR <= '1';
                                    end if;
```

```
                when S1 =>
                        if LWR = '1' and ISP1582_Data_Port = '1' then -- write operation start
                            WR <= '1';
                            RD <= '1';

                        elsif LWR = '0' and ISP1582_Data_Port = '1' then -- read operation
start

                            RD <= '1';
                            WR <= '1';

                        end if;

                        DATA_STATE <= S2;

                when S2 =>
                        WR <= '1';
                        RD <= '1';
                        DATA_STATE <= S3;
                when S3 =>
                        WR <= '1';
                        RD <= '1';
                        DATA_STATE <= S4;

                when S4 =>
                        if ISP1582_Data_Port = '0' then
                            DATA_STATE <= S0;
                        end if;

                        WR <= '1';
                        RD <= '1';

                when others =>
                        WR <= '1';
                        RD <= '1';
                        DATA_STATE <= S0;
                end case;

            end if;
        end process;


ISP1582_Address_Decode_Process:

        process(ADS,GCLK,LRESET)
        begin


            if LRESET = '0' then

                ISP1582_Address_Port <= '0';
                ISP1582_Data_Port <= '0';
                Control_Port <= '0';
                DMA_Port <= '0';

            elsif rising_edge(GCLK) then
```

```
if ADS = '0' then

        case ISP1582_Address(7 downto 0) is

        when "00000000" => -- address port

                ISP1582_Address_Port <= '1';
                ISP1582_Data_Port <= '0';
                Control_Port <= '0';
                DMA_Port <= '0';

        when "00000010" => -- data port

                ISP1582_Address_Port <= '0';
                ISP1582_Data_Port <= '1';
                Control_Port <= '0';
                DMA_Port <= '0';

        when "00000100" => -- control port

                ISP1582_Address_Port <= '0';
                ISP1582_Data_Port <= '0';
                Control_Port <= '1';
                DMA_Port <= '0';

        when "00000110" => -- dma port

                ISP1582_Address_Port <= '0';
                ISP1582_Data_Port <= '0';
                Control_Port <= '0';
                DMA_Port <= '1';

        when others =>

                ISP1582_Address_Port <= '0';
                ISP1582_Data_Port <= '0';
                Control_Port <= '0';
                DMA_Port <= '0';

        end case;

else
        ISP1582_Address_Port <= ISP1582_Address_Port;
        ISP1582_Data_Port <= ISP1582_Data_Port;
        Control_Port <= Control_Port;
        DMA_Port <= DMA_Port;


        if LHOLD = '0' then
                ISP1582_Address_Port <= '0';
                ISP1582_Data_Port <= '0';
                Control_Port <= '0';
                DMA_Port <= '0';
        end if;
end if;
```

```
                end if;
        end process;




    BIGEND <= 'Z';


        process(GCLK,LHOLD,LRESET)
        begin

                if LRESET = '0' then
                        LHOLDA <= '0';

                elsif rising_edge(GCLK) then
                        LHOLDA <= LHOLD;
                end if;
        end process;

    BREQi <= 'Z';
        USERi <= 'Z';
    LINT <= INT when INT_EN = '1' else '1';
    CCS <= '1';

    LB <= '0';
    UB <= '0';
    CE <= not DMA_START;

        SRAM_ADDR <= CONV_STD_LOGIC_VECTOR(ADDR_COUNTER,16) when DMA_RESET = asserted
else (others => '0');

STATE_MACHINE_Process:

        process(GCLK,DMA_RESET,DREQ,DMA_RDWR,LRESET)
        begin

                if DMA_RESET = deasserted or LRESET = deasserted then

                        STATE <= S0;

                elsif rising_edge(GCLK) then

                        if ((DREQ = asserted and DMA_START = asserted) or
                           (DREQ = deasserted and DACK = deasserted and DMA_START = asserted) or
                           (STATE /= S0 and DMA_START = asserted)) then

                                case    STATE is

                                when S0 =>
                                        STATE <= S1;
```

```
                    when S1 =>
                            STATE <= S2;
                    when S2 =>
                            STATE <= S3;
                    when S3 =>
                            STATE <= S0;
                    when others =>
                            STATE <= S0;

                    end case;
            else
                    STATE <= STATE;
            end if;
        end if;
    end process;

DACK_Process:

    process(GCLK,DMA_RESET,DREQ,DACK,MODE,DMA_RDWR,LRESET)
    begin

        if DMA_RESET = deasserted or LRESET = deasserted then

            DACK <= asserted;

        elsif rising_edge(GCLK) then

            case MODE is

            when "00" =>

                if DMA_START = asserted then
                    if STATE = S2 then
                        DACK <= not DREQ;
                    else
                        DACK <= DACK;
                    end if;
                else
                    DACK <= asserted;
                end if;

            when others =>

                if DMA_RDWR = READ and MODE = "01" then

                    if STATE = S2 then
                        DACK <= not DREQ;
                    else
                        DACK <= DACK;
                    end if;

                else

                    case STATE is

                    when S3 =>
```

```
                        if DREQ = asserted then
                            DACK <= deasserted;
                        else
                            DACK <= asserted;
                        end if;

                    when S1 =>

                        if DREQ = asserted then

                            DACK <= asserted;

                        elsif DREQ = deasserted then

                            DACK <= deasserted;

                        end if;

                    when S2 =>

                        if DREQ = deasserted then

                            DACK <= asserted;

                        else
                            DACK <= DACK;

                        end if;

                    when others =>
                            DACK <= DACK;

                    end case;
                end if;
            end case;
        end if;
    end process;

DIOR_DIOW_Process:

    process(GCLK,DMA_RESET,DACK,DMA_RDWR,MODE,LRESET,DMA_START)
    begin

        if DMA_RESET = deasserted or LRESET = deasserted then

            DIOR <= asserted;
            DIOW <= asserted;

        elsif rising_edge(GCLK) then

            if DACK = deasserted and DMA_START = asserted then

            case MODE is

            when "00" =>
```

```
        case STATE is

        when S3 =>

                if DMA_RDWR = WRITE then
                        DIOW <= deasserted;
                        DIOR <= asserted;
                else
                        DIOW <= asserted;
                        DIOR <= deasserted;
                end if;

        when S0 =>

                if DMA_RDWR = WRITE then
                        DIOW <= deasserted;
                        DIOR <= asserted;
                else
                        DIOW <= asserted;
                        DIOR <= deasserted;
                end if;

        when others  =>

                DIOW <= asserted;
                DIOR <= asserted;

        end case;

when "01" =>

        case STATE is

        when S3 =>

                if DMA_RDWR = WRITE then
                        DIOW <= asserted;
                        DIOR <= asserted;
                else
                        DIOW <= asserted;
                        DIOR <= deasserted;
                end if;

        when S0 =>


                if DMA_RDWR = WRITE then
                        DIOW <= asserted;
                        DIOR <= asserted;
                else
                        DIOW <= asserted;
                        DIOR <= deasserted;
                end if;

        when others =>
```

```
                                        DIOW <= asserted;
                                        DIOR <= asserted;

                                end case;

                        when others =>
                                DIOW <= asserted;
                                DIOR <= asserted;

                        end case;
                        else
                                DIOW <= asserted;
                                DIOR <= asserted;
                        end if;
                end if;
        end process;

ADDR_Counter_Process:

        process(ADDR_INC,DMA_RESET,LRESET)
        begin

                if DMA_RESET = deasserted or DMA_START = deasserted or LRESET = deasserted then

                        ADDR_COUNTER <= 0;

                elsif rising_edge(ADDR_INC) then

                        ADDR_COUNTER <= ADDR_COUNTER + 1;

                end if;
        end process;

SRAM_RDWR_Process:

        process(GCLK,DMA_RESET,MODE,LRESET)

        variable DELAY: STD_LOGIC;

        begin

                if DMA_RESET = deasserted or LRESET = deasserted then

                        WE <= asserted;
                        OE <= asserted;
                        DELAY := asserted;
                        ADDR_INC <= deasserted;



                elsif rising_edge(GCLK) then

                        if DMA_RDWR = WRITE then

                                case STATE is
```

```
        when S2 =>

            ADDR_INC <= deasserted;

            if DREQ = asserted then
                OE <= deasserted;
            else
                OE <= asserted;
            end if;

        when S0 =>

            if DACK = deasserted then
                ADDR_INC <= asserted;
            else
                ADDR_INC <= deasserted;
            end if;

        when others =>
                OE <= asserted;
                ADDR_INC <= deasserted;

        end case;

    else

        case STATE is

        when S0 =>

            ADDR_INC <= deasserted;

            if DACK = deasserted then
                WE <= deasserted;
            else
                WE <= asserted;
                DELAY := asserted;

            end if;

        when S2 =>

            if DELAY = deasserted then
                ADDR_INC <= asserted;
            else
                ADDR_INC <= deasserted;
                DELAY := deasserted;
            end if;

        when others =>
                WE <= asserted;

        end case;
    end if;
```

```
        end if;
    end process;

end PCIKit_arch;
```

12. PLX Technology PCI9054 Serial EEPROM Binary

```
5406    10B5    0680    000B    0000    010A    0000    0000
0000    0000    FFFF    FF01    2000    0001    0161    000C
0030    0500    0000    0000    0000    0010    8941    0041
FF00    0000    4000    0000    5000    0000    0000    2000
0000    0000    9054    10B5    FFFF    FF01    2000    0001
0000    0141    0000    4C06
```

After reset, the PCI chip (PCI 9054) on the ISP1582 PCI evaluation board reads the contents of EEPROM for its PCI Configuration registers. For more details, refer to the *PCI 9054 Data Book from PLX Technology*.

13. Reference

- *ISP1582 Hi-Speed Universal Serial Bus interface device* datasheet.
- *ISP1582/83 Software Guide.*
- *PLX Technology PCI9054 PCI to Local Bus Bridge* datasheet and user manual

Philips Semiconductors