

UMI0040_I

ISPI582 Hi-Speed USB Device PCI Mass Storage Kit

Semiconductors

August 2003

User's Guide Rev. 1.0

Revision History:

Version	Date	Description	Author
1.0	August 2003	Initial version	Guo Yang Bin

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

1.	INTRODUCTION	5
2.	SYSTEM REQUIREMENTS.....	5
3.	BLOCK DIAGRAM.....	6
4.	PCB LAYOUT	7
5.	COMPONENT PLACEMENT	8
5.1.	ISPI582.....	8
5.2.	XILINX XCS30XL FPGA AND XC17S30XL SERIAL PROM.....	9
5.3.	PLX TECHNOLOGY PCI9054 PCI BRIDGE AND SERIAL EEPROM	10
5.4.	SRAM.....	11
6.	HEADER AND CONNECTOR PLACEMENT	12
6.1.	USB CONNECTORS.....	12
6.2.	ISPI582 GENERIC PROCESSOR EXPANSION BUS	13
6.3.	JTAG HEADER	14
7.	ISPI582 PCI MASS STORAGE KIT SETUP PROCEDURE.....	15
7.1.	PCI KIT SETUP PROCEDURE	15
7.2.	PCI KIT RESOURCES.....	19
7.2.1.	I/O Address.....	19
7.2.2.	ISPI582 Address Port Access Diagram	20
7.2.3.	ISPI582 Data Port Access Diagram.....	22
7.2.4.	Xilinx Spartan Control Port Access Diagram.....	24
8.	SCHEMATICS OF EVAL BOARD	25
8.1.	ISPI582 PCI BOARD	25
9.	BILL OF MATERIAL	30
9.1.	ISPI582 PCI KIT MAIN BOARD	30
10.	XILINX XCS30XL DMA CONTROLLER VHDL CODE.....	35
11.	PLX TECHNOLOGY PCI9054 SERIAL EEPROM BINARY	49
12.	REFERENCE.....	49

FIGURES

Figure 1-1 ISPI582 PCI Board	5
Figure 3-1: PCI Board Block Diagram.....	6
Figure 4-1: PCI Board PCB.....	7

TABLES

Table 7-1: I/O Mapping.....	19
Table 9-1: Build of Material of ISPI582 Main Board	30

Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corp. in the United States and/or other countries. The names of actual companies and products mentioned herein may be the trademarks of their respective owners. All other names, products, and trademarks are the property of their respective owners.

I. Introduction

The ISPI582 Hi-Speed USB Device PCI Mass Storage kit allows you to develop a mass storage device with ISPI582—which is a Hi-Speed Universal Serial Bus (USB) interface device—in the Generic Processor Mode, that is, separate address and data bus operation.

The PCI eval board has the ISPI582, Xilinx® XCS30XL, Xilinx XC17S30XL serial PROM, PLX9054 PCI-to-local bus bridge, SRAM, and serial EEPROM on board. The PCI Mass Storage kit can connect the ISPI582 to any generic processor. Figure I-1 shows the ISPI582 PCI board.



Figure I-1 ISPI582 PCI Board

2. System Requirements

PC Host:

- Hi-Speed USB Host Controller add-on card*, on Windows 2000 or Windows XP.

Device:

- Device PC with floppy driver; 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 kit.

3. Block Diagram

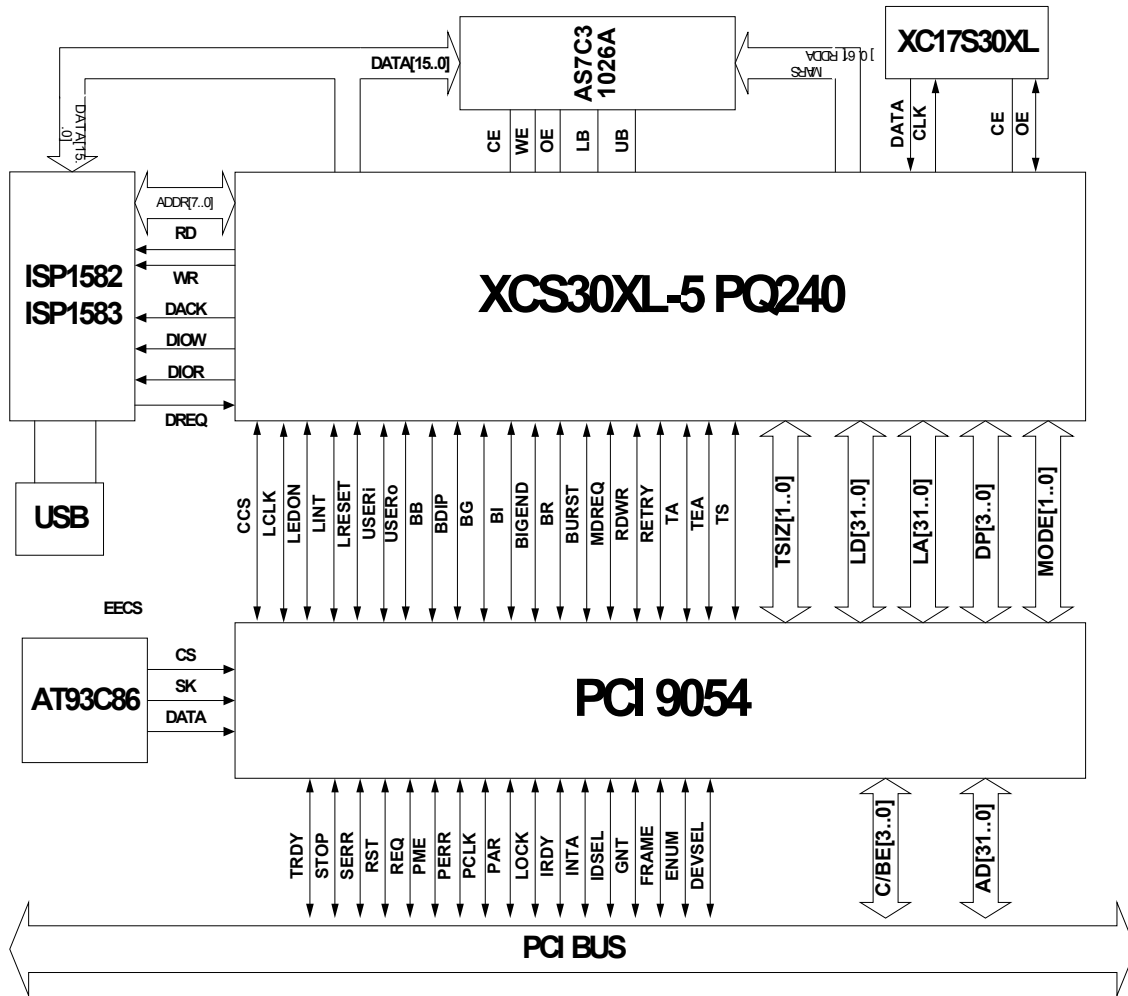


Figure 3-1: PCI Board Block Diagram

Figure 3-1 illustrates how the ISPI582 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 ISPI582 registers with the help of the Xilinx XCS30XL, which converts the PLX9054 local bus to the ISPI582 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

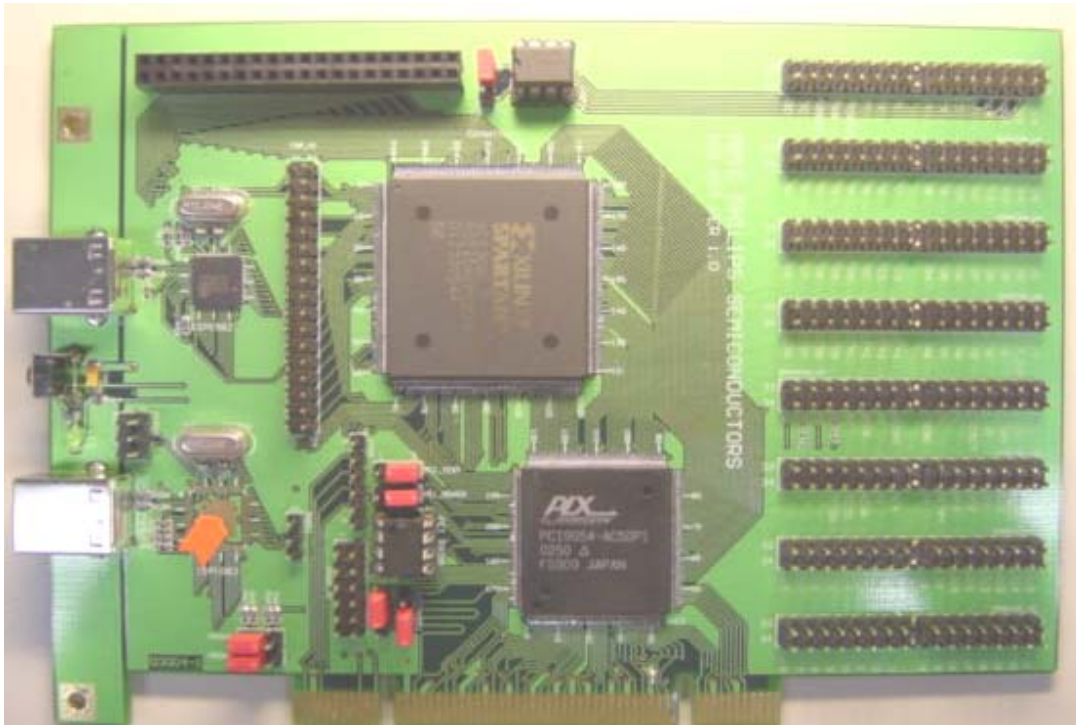
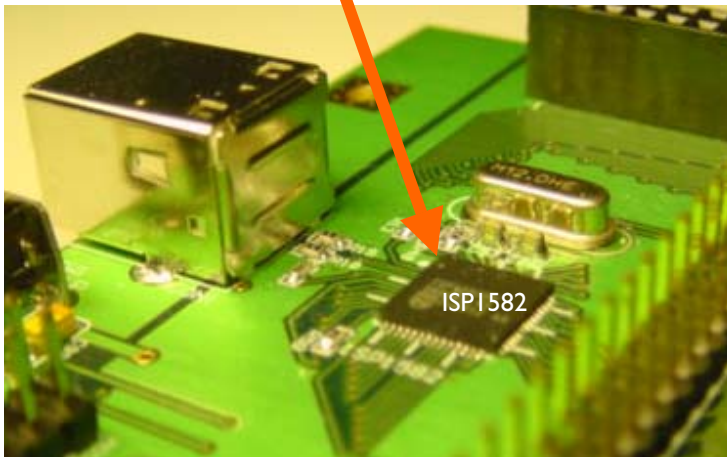
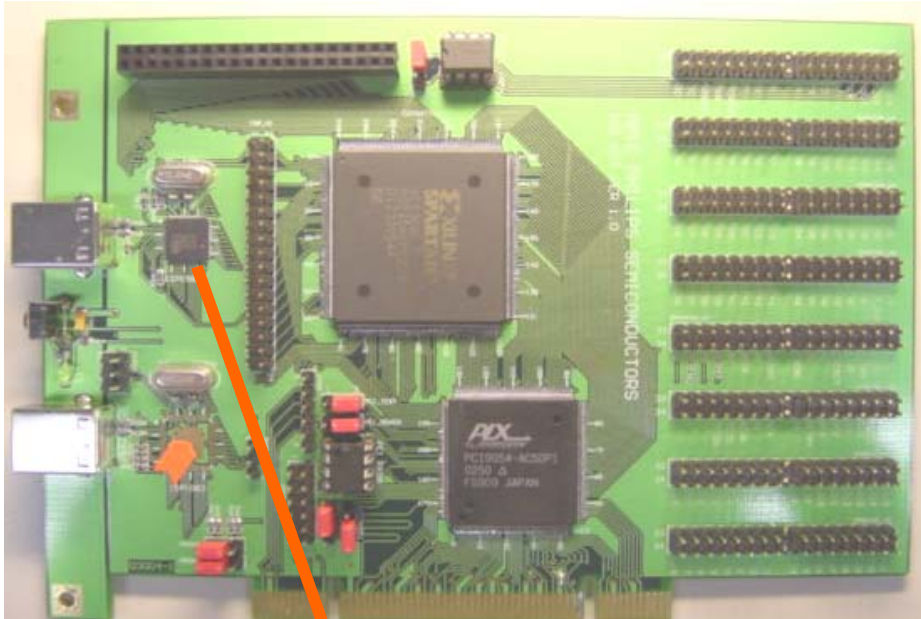


Figure 4-1: PCI Board PCB

Figure 4-1 shows the PCB layout and the placement of components on the ISPI582 PCI board. The PCB is designed for future expansion for ISPI583.

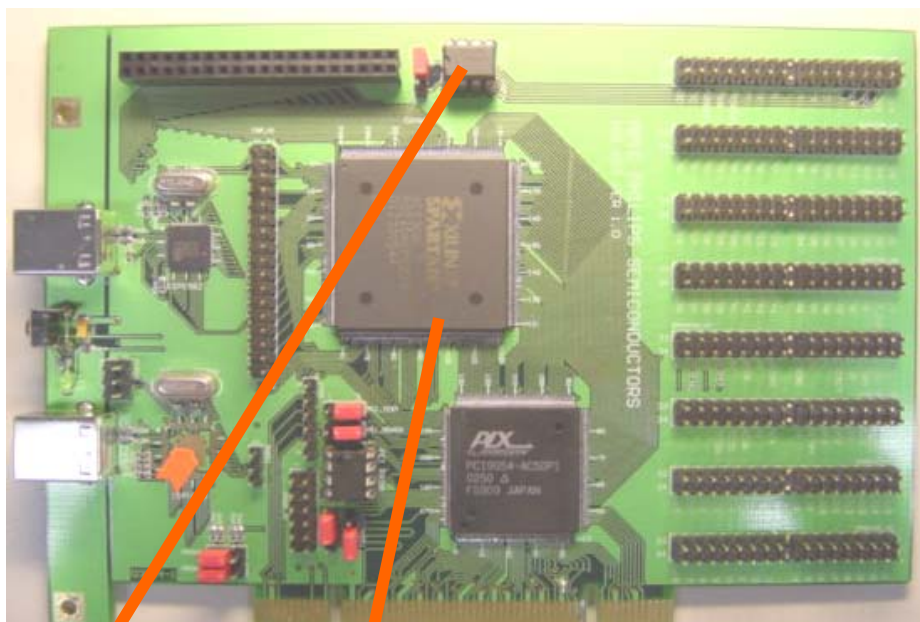
5. Component Placement

5.1. ISPI582



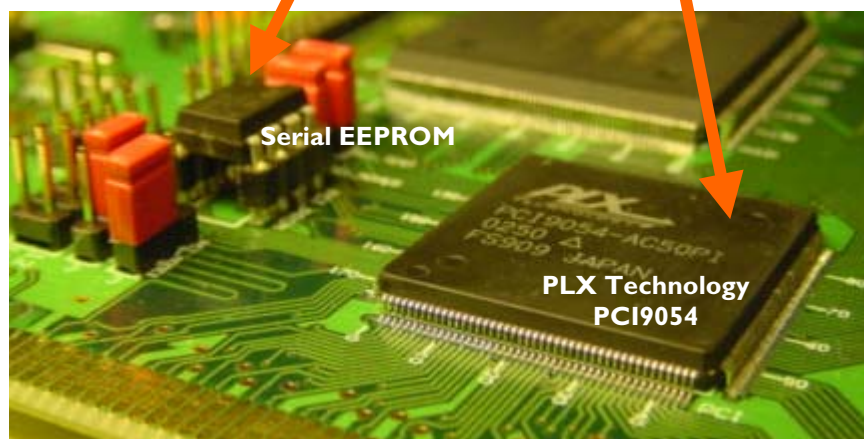
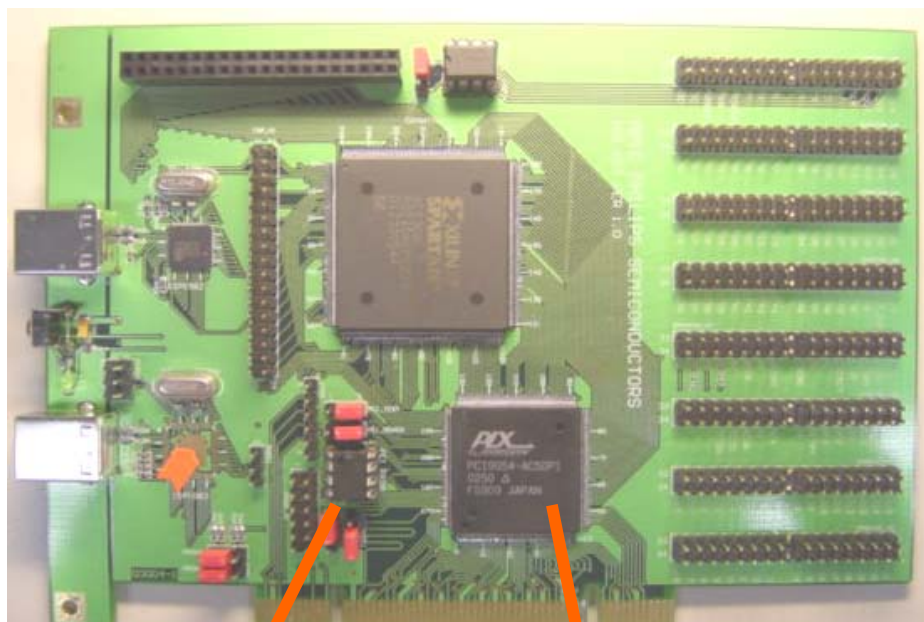
ISPI582 is at the upper-left corner of the PCI board.

5.2. Xilinx XCS30XL FPGA and XCI7S30XL Serial PROM



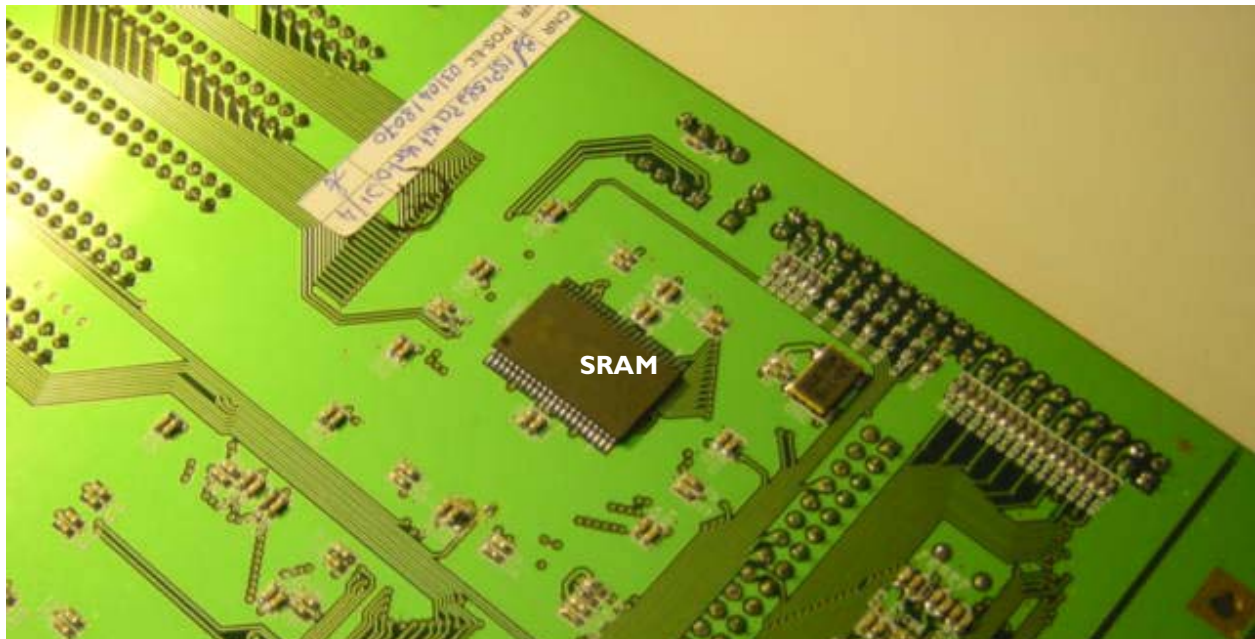
The Xilinx XCS30XL translates the PLX9054 PCI bridge local bus to the ISPI582 generic processor bus. It also acts as the Master DMA controller for the DMA transfer. Upon power on, the Xilinx XCI7S30XL serial PROM configures the Xilinx XCS30XL to the local bus translator and the master DMA controller.

5.3. 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.

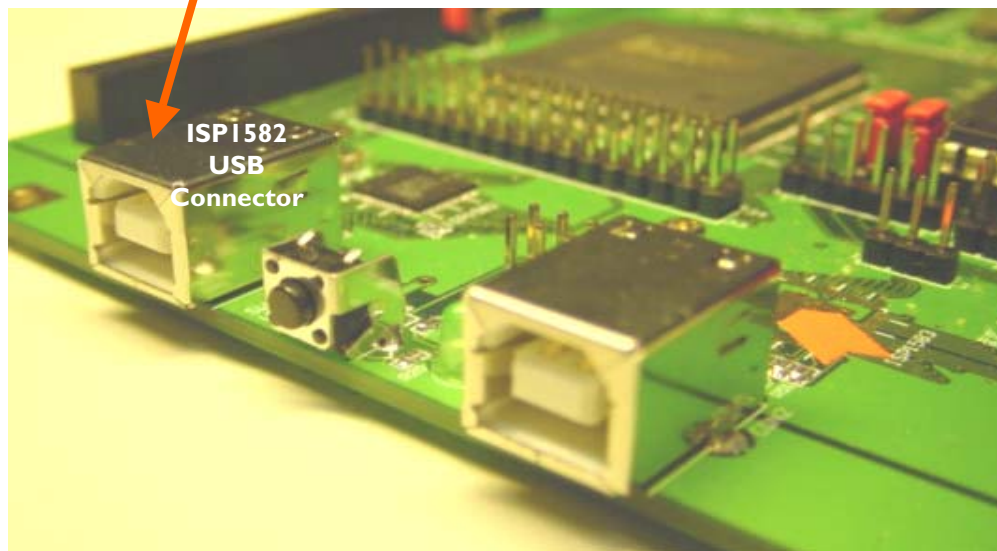
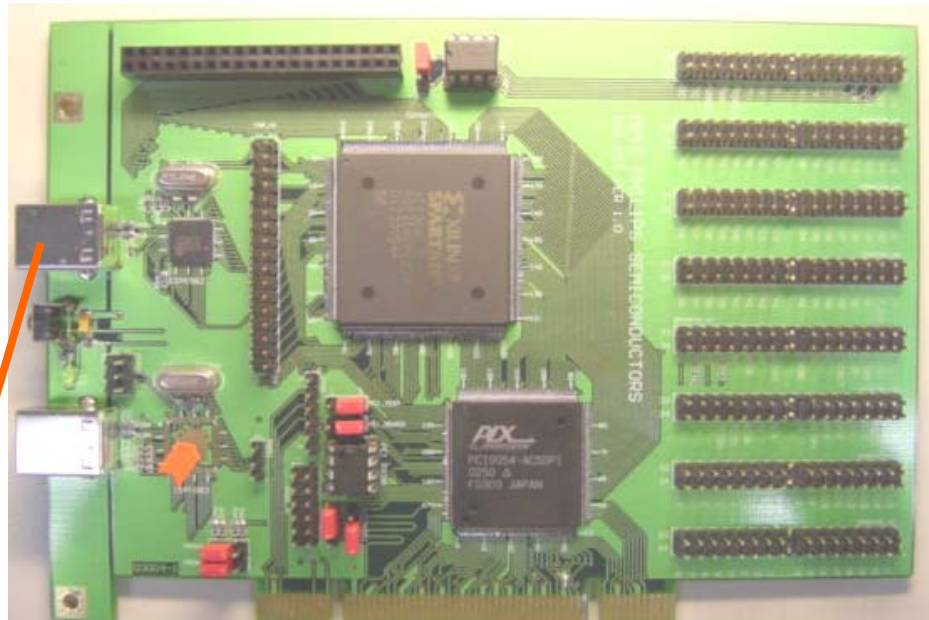
5.4. SRAM



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.

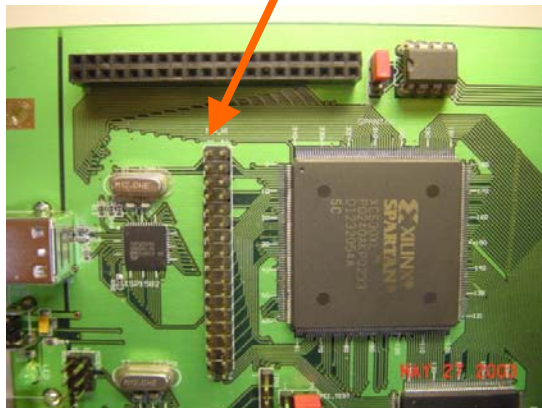
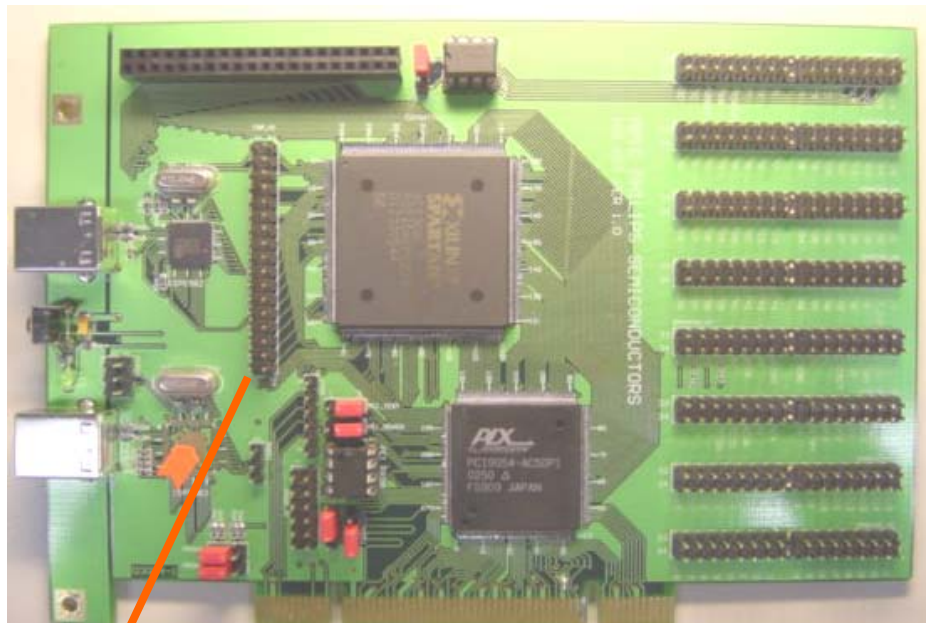
6. Header and Connector Placement

6.1. USB Connectors

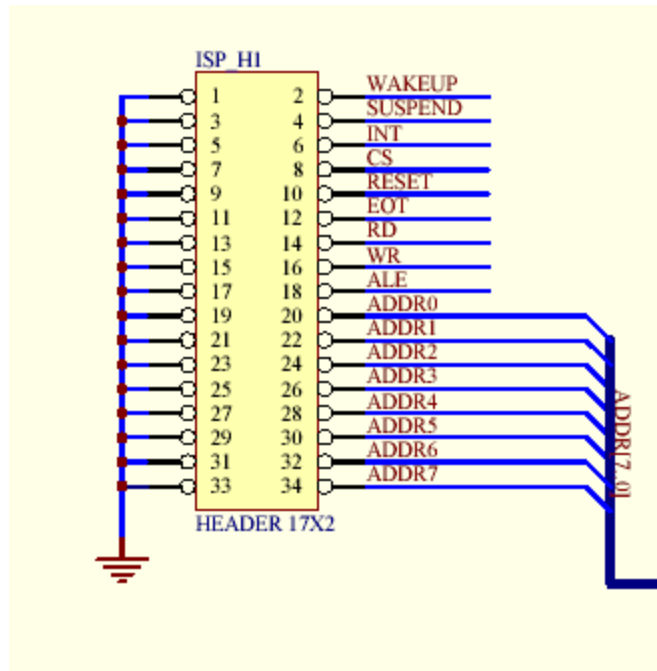


The ISPI582 USB connector is at the upper section of the PCI board..

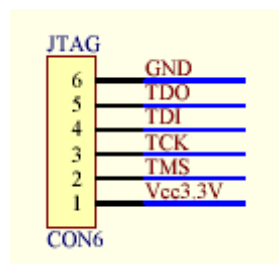
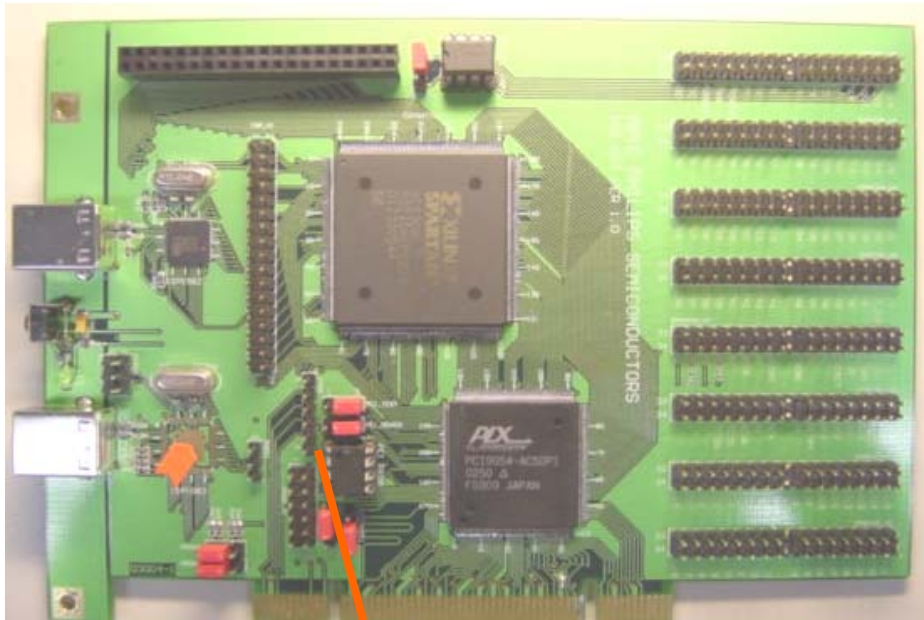
6.2. ISPI582 Generic Processor Expansion Bus



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



6.3. JTAG Header



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

7. ISPI582 PCI Mass Storage Kit Setup Procedure

7.1. PCI Kit Setup Procedure

1. Switch off the device PC and allocate an empty PCI slot.
2. Insert the ISPI582 PCI board into an available PCI slot, and switch on the device PC.



3. The *Add New Hardware Wizard* is displayed. Click **Next**.



4. The *Search for the best driver for your device* is selected by default. Click **Next**.



- In the next wizard, deselect all the checked options and click **Next**.



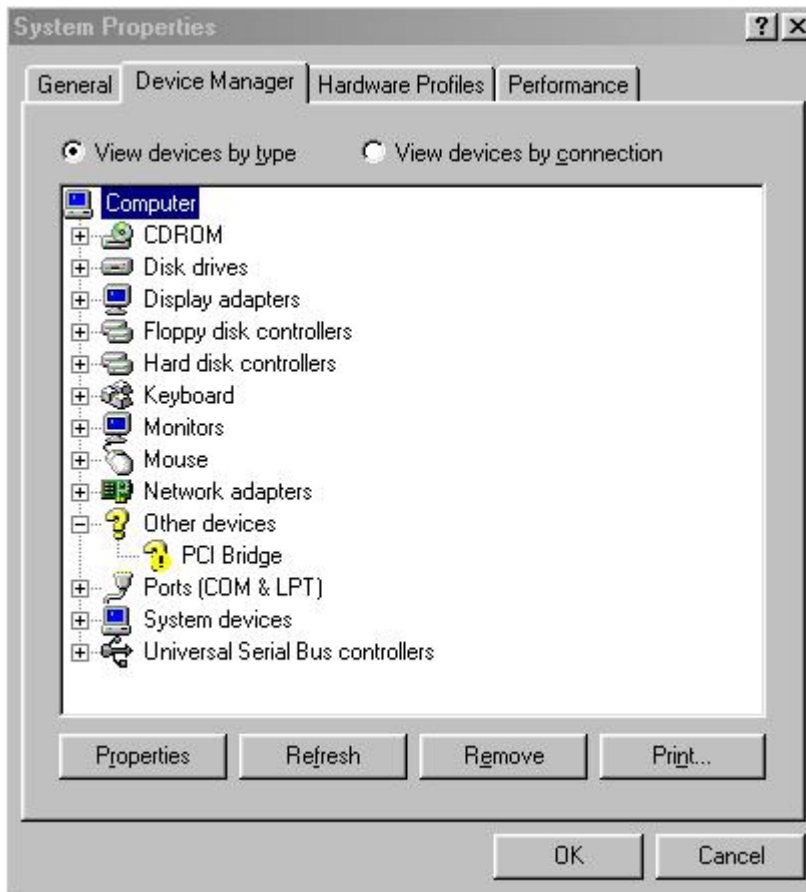
- Click **Next**.



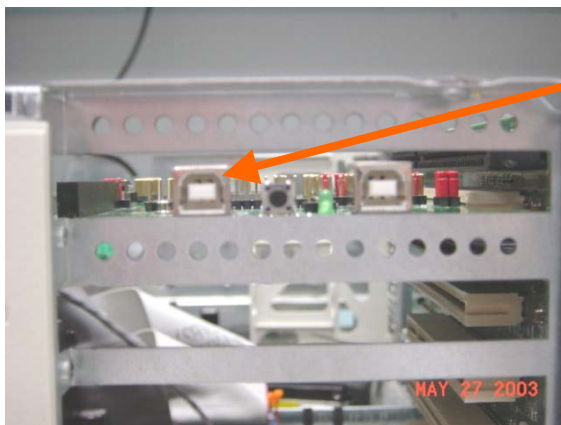
7. To complete the procedure for adding a new hardware, click **Finish**.



8. Open *System Properties* to view the new hardware.



9. Format a new 1.44M bytes floppy disk and insert the disk in the drive A of device PC. Run the PCKit.exe file on the device PC.
10. Plug in the host USB cable to the ISPI582 USB connector.



**ISPI582
USB
Connector**

After successful enumeration, you will see a new drive on the host PC. You can write and read files from this drive.

7.2. PCI Kit Resources

The INTA of the PCI bus is utilized to reflect the ISPI582 interrupt.

7.2.1. I/O Address

Table 7-1: 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 ISPI582, this port is used to latch the address for the ISPI582 register. Write on this port latches the lower byte of PCI data to the ISPI582 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 ISPI582 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 ISPI582.
PCI Base Address + 6	Word (16 bit) DMA Port	Write only	DMA port. To program on board DMA Controller.

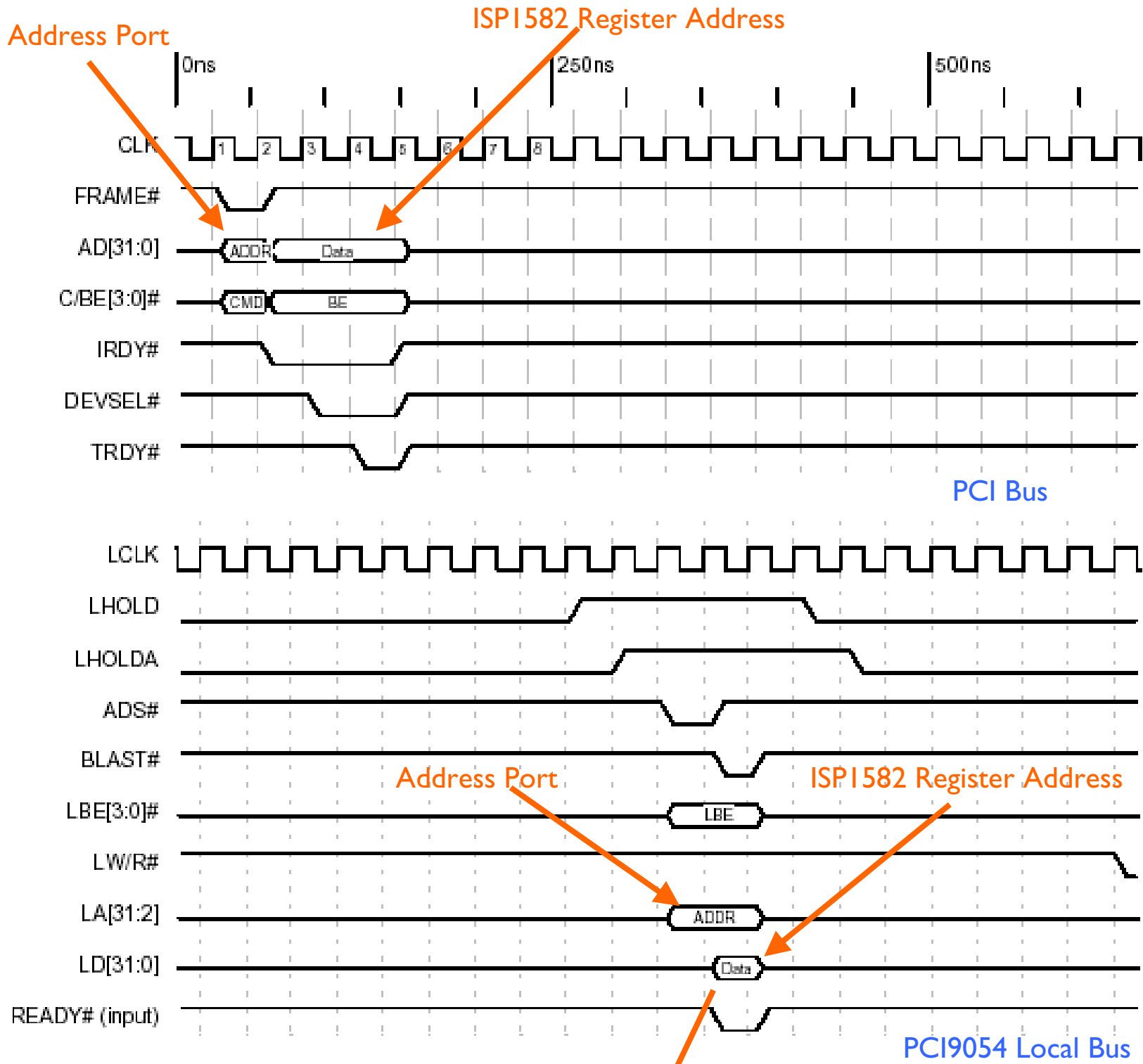
The operation behavior of an odd address is unpredictable. Byte access is not allowed.

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 ISPI582.
- The Data port holds the 16-bit data that is to be written to or read out from the ISPI582 register.
- The Control port has only a single function: to mask the interrupt from the ISPI582. 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.

7.2.2. ISP1582 Address Port Access Diagram



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

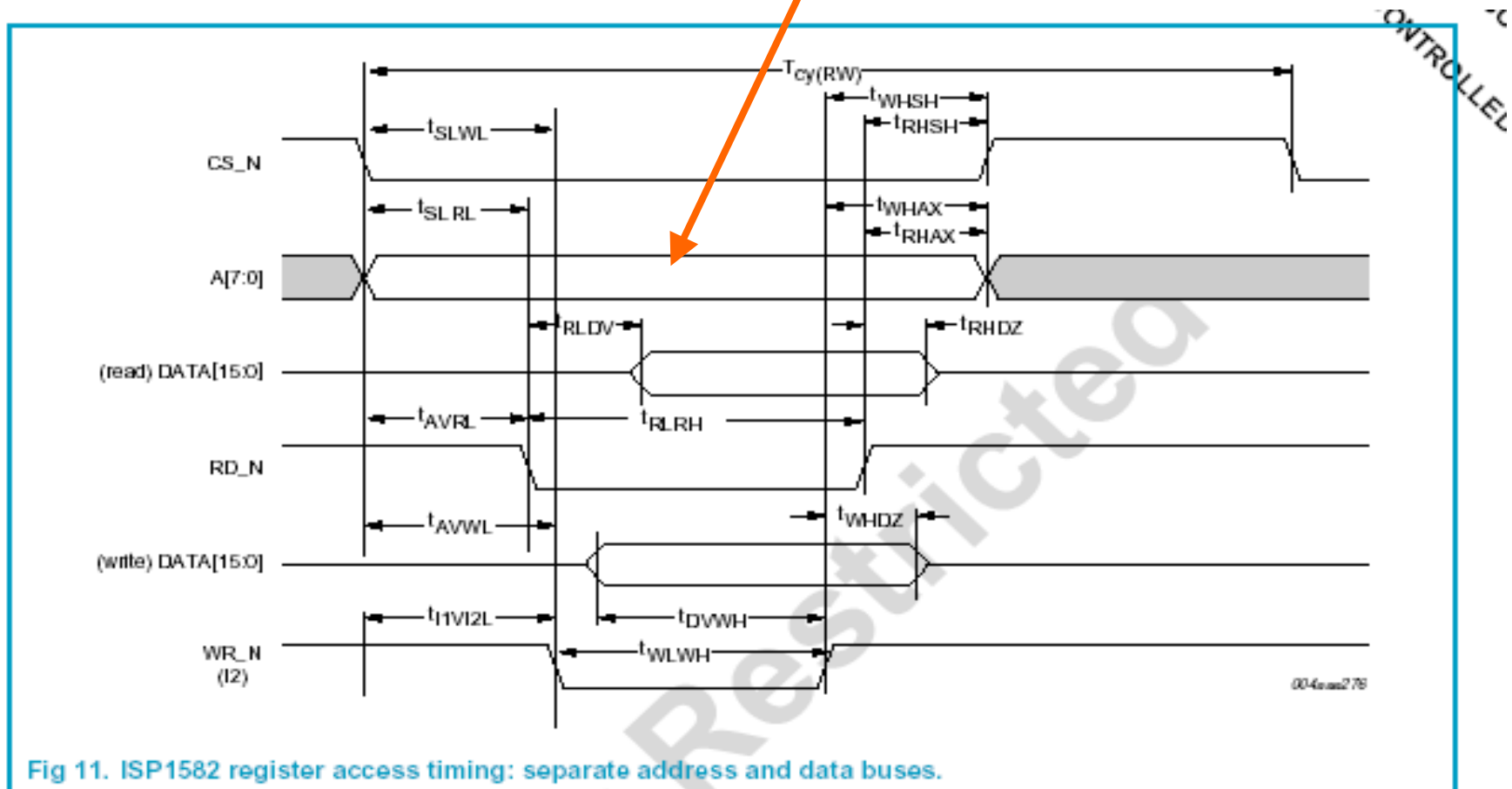
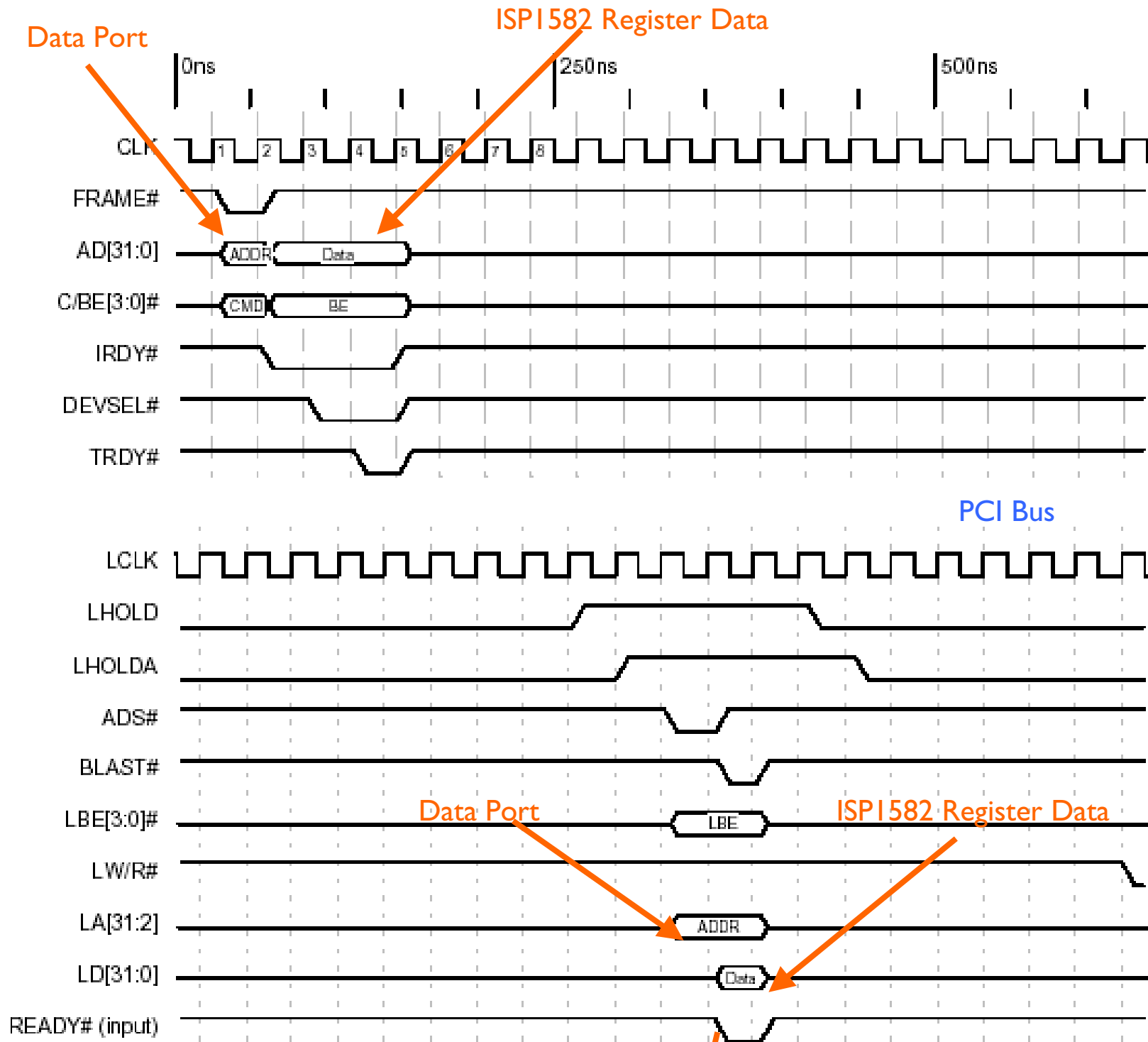


Fig 11. ISP1582 register access timing: separate address and data buses.

7.2.3. ISPI582 Data Port Access Diagram



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

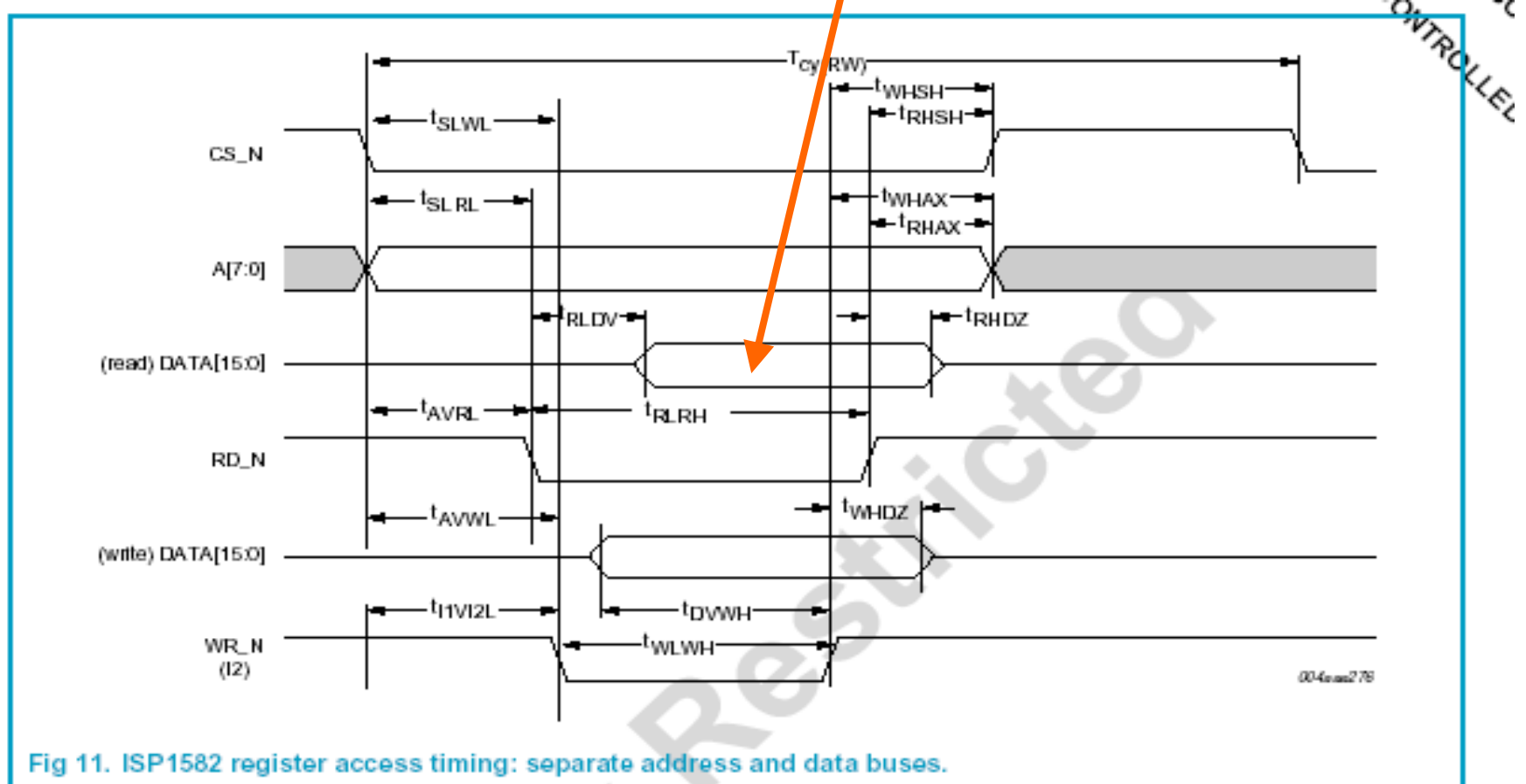
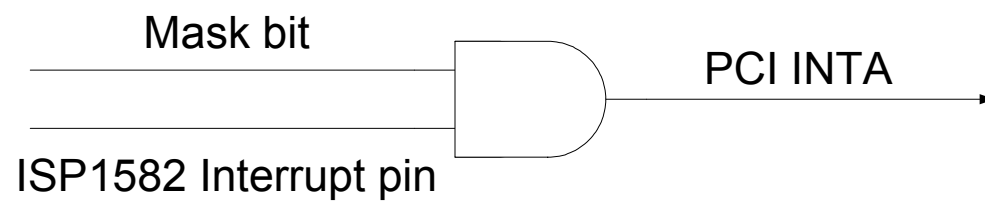
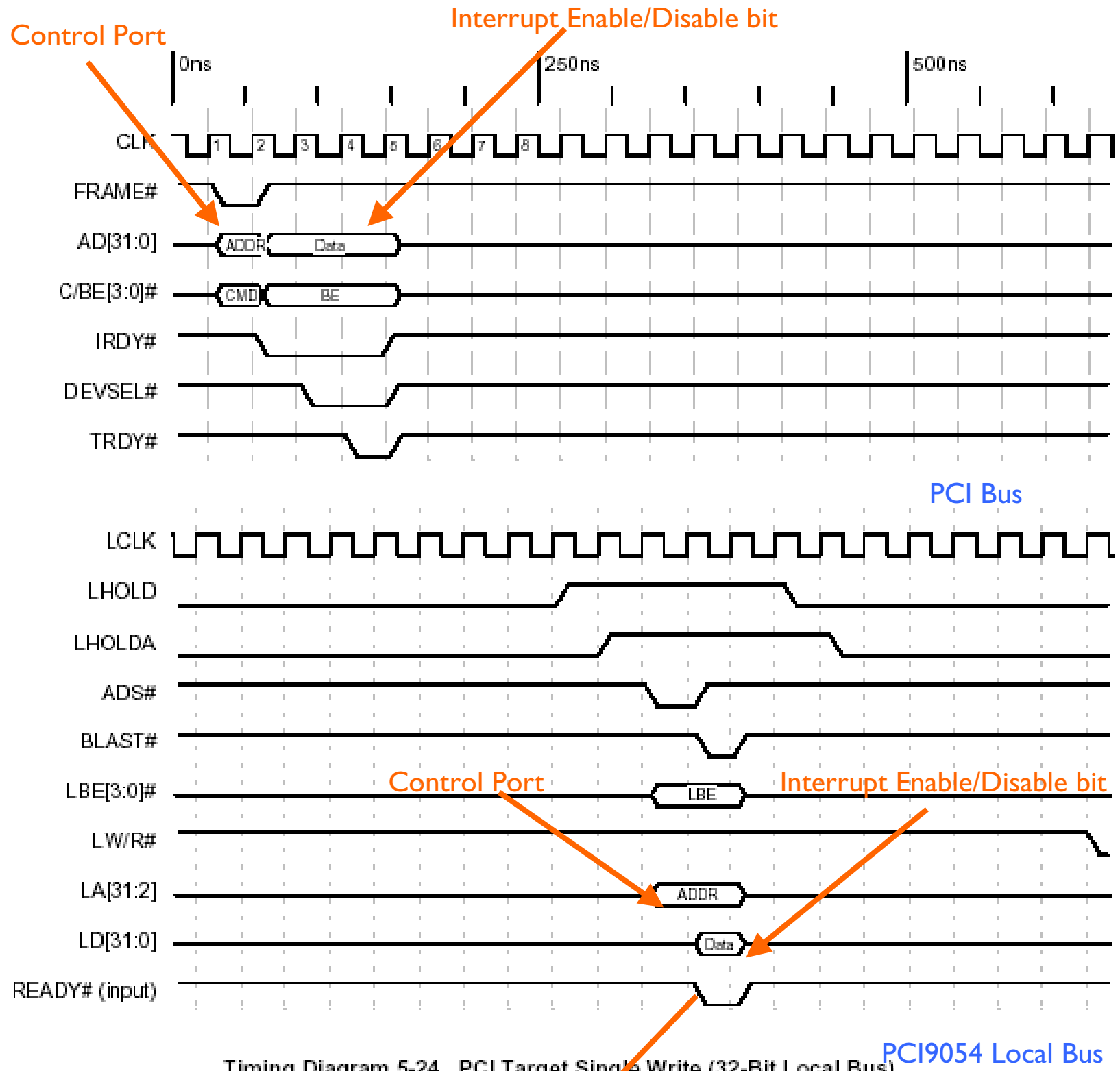


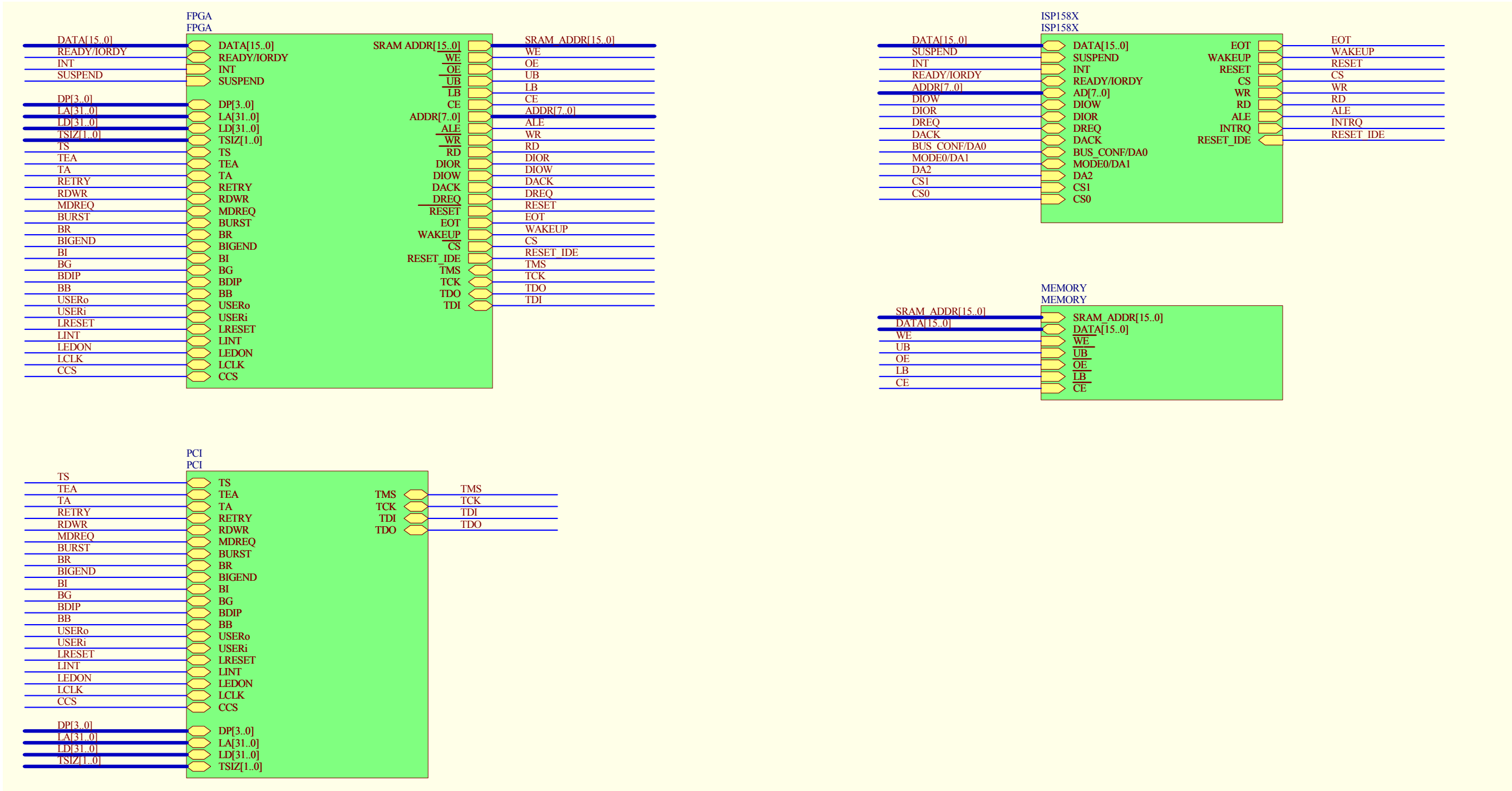
Fig 11. ISPI582 register access timing: separate address and data buses.

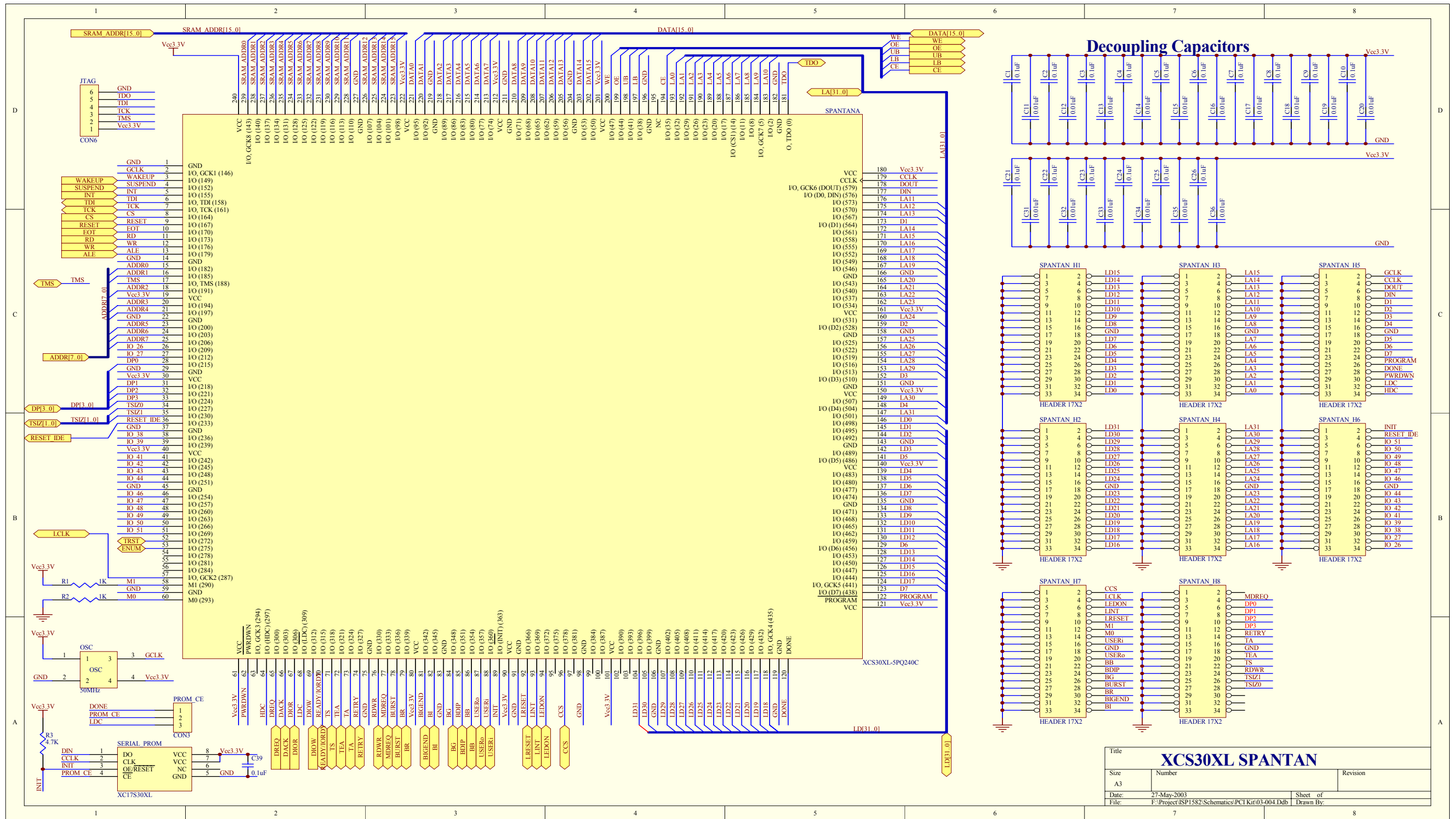
7.2.4. Xilinx Spartan Control Port Access Diagram



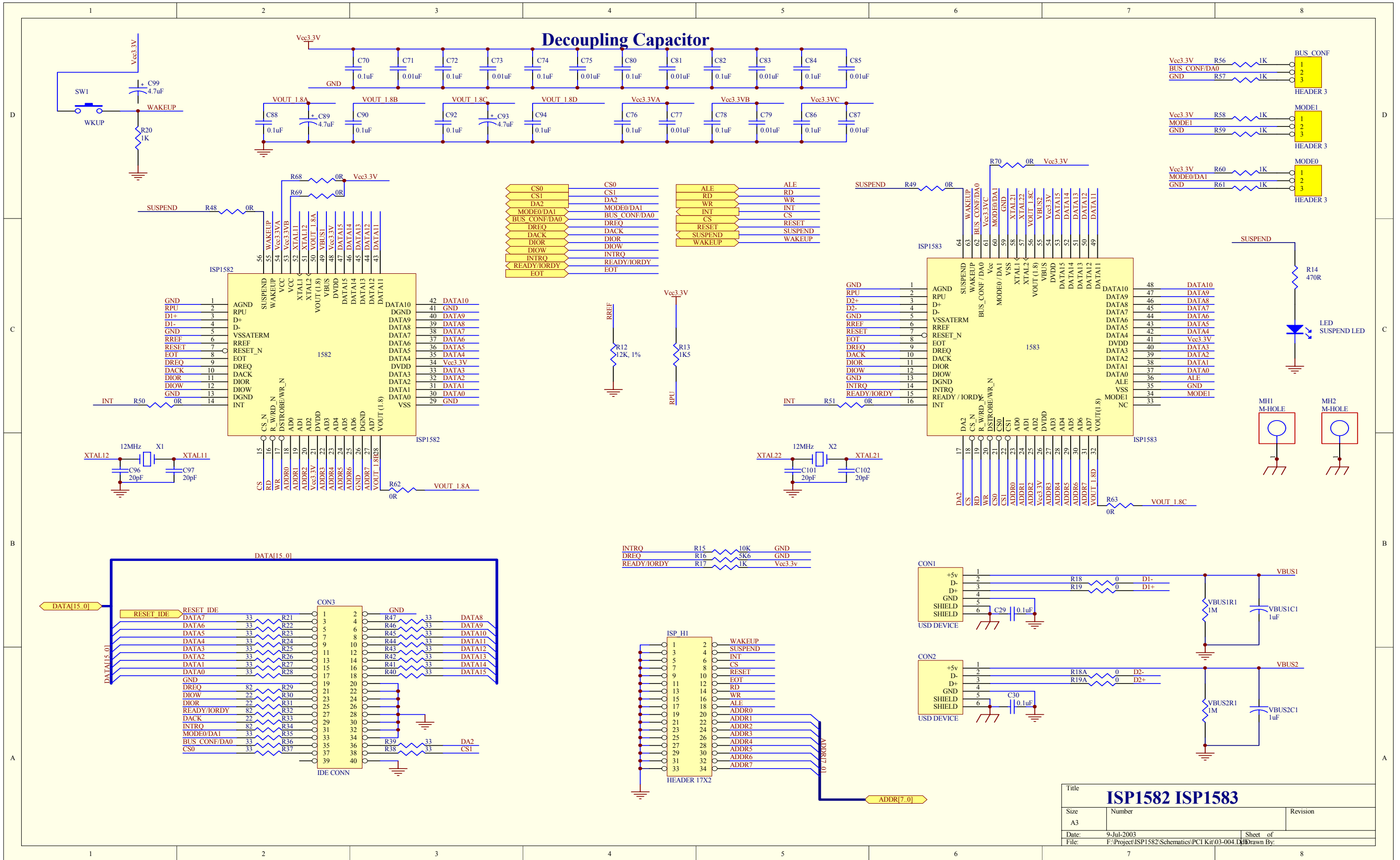
8. Schematics of Eval Board

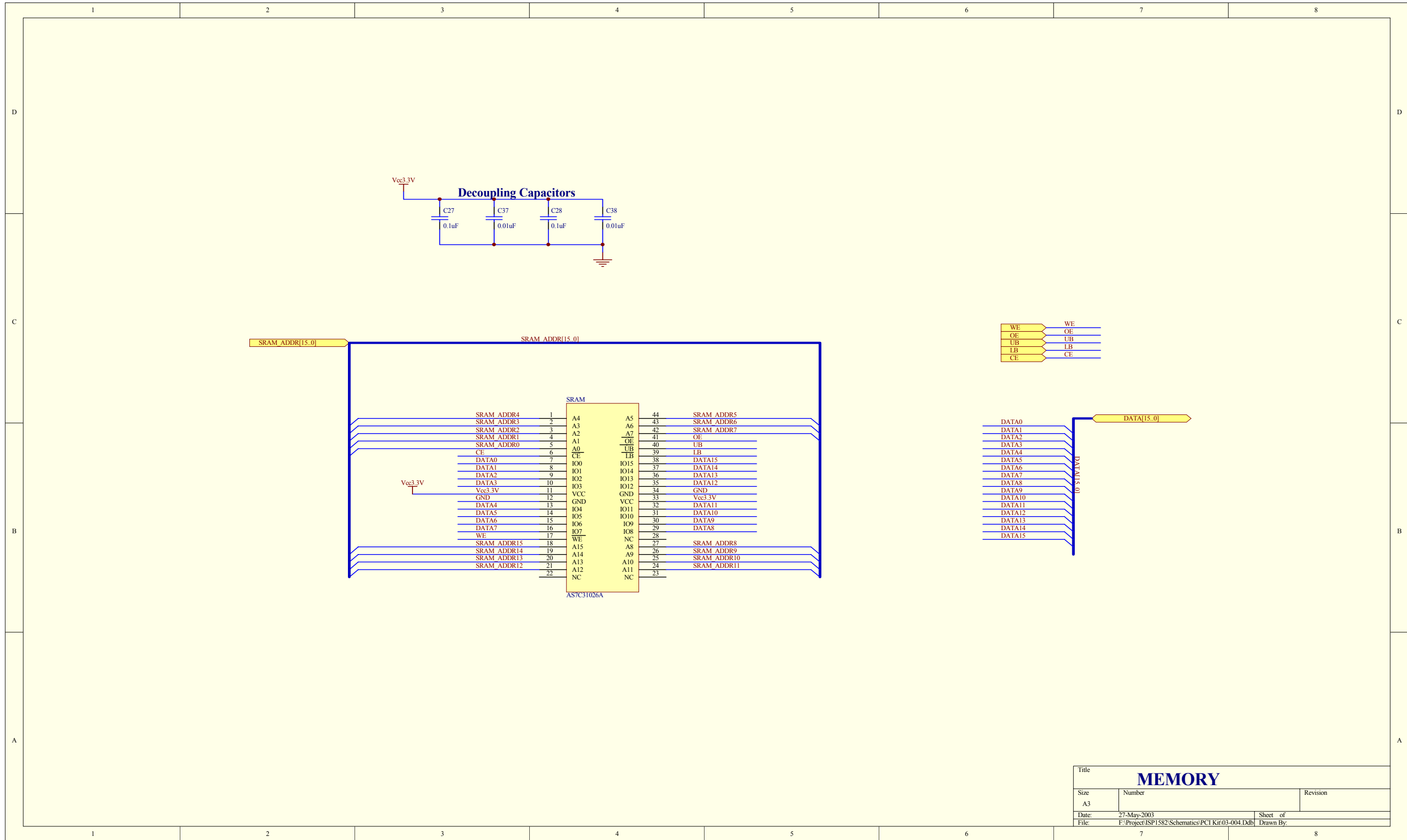
8.1. ISP1582 PCI Board



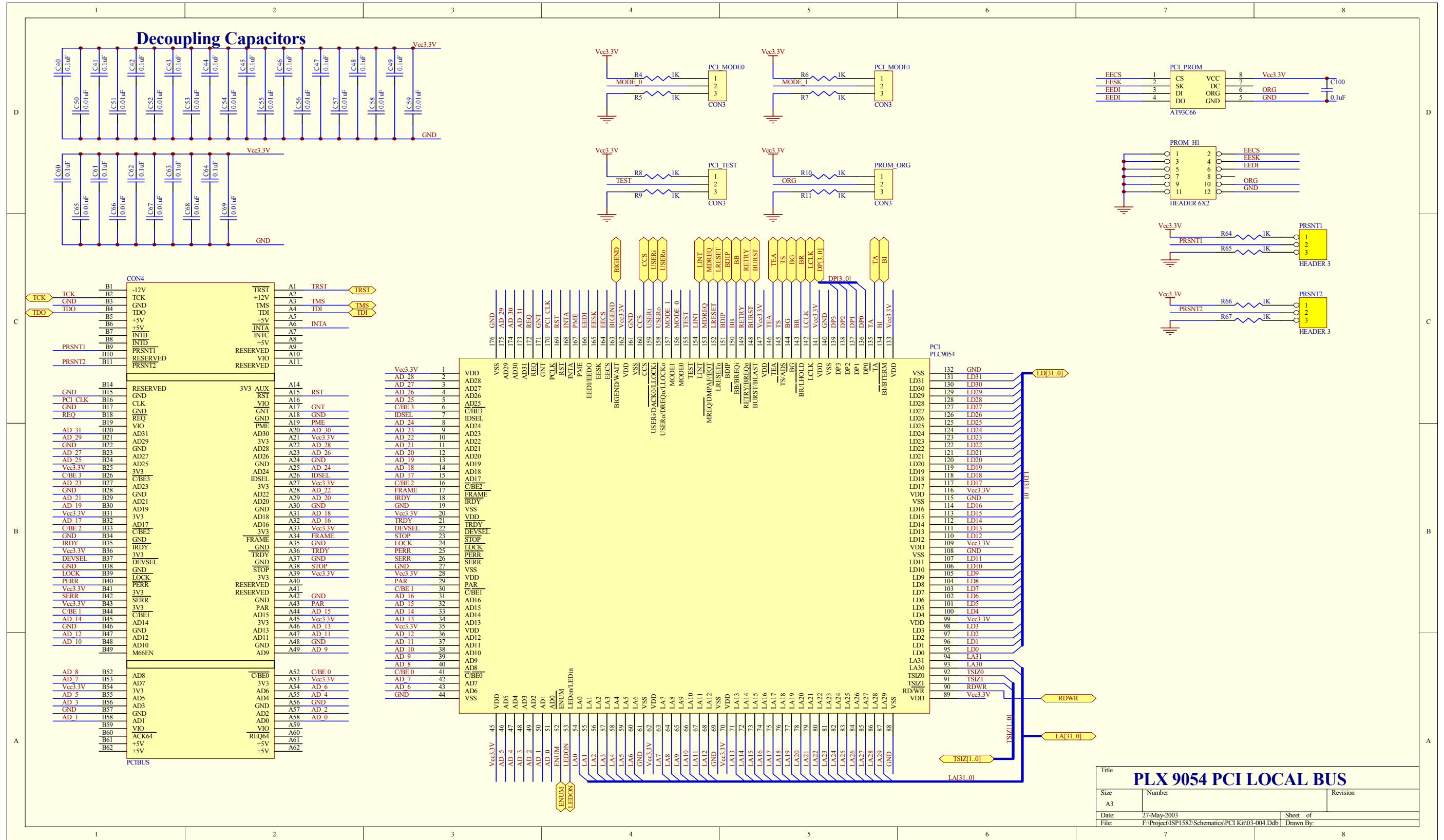


Title		
XCS30XL SPANTAN		
Size	Number	Revision
A3		
Date:	27-May-2003	Sheet of
File:	F:\Project\ISP1582\Schematics\PCI Kit\03-004.Ddb	Drawn By:





Title		
MEMORY		
Size	Number	Revision
A3		
Date:	27-May-2003	Sheet of
File:	F:\Project\ISP1582\Schematics\PCI Kit03-004.Ddb	Drawn By:



9. Bill Of Material

9.1. ISPI582 PCI Kit Main Board

Table 9-1: Build of Material of ISPI582 Main Board

Part Type	Designator	Footprint
0	R19A	0603R
0	R18A	0603R
0	R18	0603R
0	R19	0603R
0.01 μ F	C83	0603C
0.1 μ F	C84	0603C
0.1 μ F	C82	0603C
0.1 μ F	C80	0603C
0.01 μ F	C81	0603C
0.01 μ F	C85	0603C
0.1 μ F	C90	0603C
0.1 μ F	C92	0603C
0.1 μ F	C88	0603C
0.1 μ F	C86	0603C
0.01 μ F	C87	0603C
0.1 μ F	C70	0603C
0.01 μ F	C71	0603C
0.1 μ F	C72	0603C
0.1 μ F	C60	0603C
0.01 μ F	C59	0603C
0.01 μ F	C58	0603C
0.01 μ F	C73	0603C
0.01 μ F	C77	0603C
0.1 μ F	C78	0603C
0.01 μ F	C79	0603C
0.1 μ F	C74	0603C
0.01 μ F	C75	0603C
0.1 μ F	C76	0603C
0.1 μ F	C94	0603C
0.1 μ F	C43	0603C
0.1 μ F	C42	0603C
0.1 μ F	C41	0603C
0.1 μ F	C46	0603C
0.1 μ F	C45	0603C
0.1 μ F	C44	0603C
0.1 μ F	C28	0603C
0.1 μ F	C27	0603C
0.1 μ F	C29	0603C
0.1 μ F	C40	0603C

Part Type	Designator	Footprint
0.01 μ F	C38	0603C
0.01 μ F	C37	0603C
0.01 μ F	C55	0603C
0.01 μ F	C54	0603C
0.01 μ F	C53	0603C
0.1 μ F	C30	0603C
0.01 μ F	C57	0603C
0.01 μ F	C56	0603C
0.1 μ F	C49	0603C
0.1 μ F	C48	0603C
0.1 μ F	C47	0603C
0.01 μ F	C52	0603C
0.01 μ F	C51	0603C
0.01 μ F	C50	0603C
0.01 μ F	C17	0603C
0.01 μ F	C18	0603C
0.01 μ F	C19	0603C
0.01 μ F	C14	0603C
0.01 μ F	C15	0603C
0.01 μ F	C16	0603C
0.01 μ F	C20	0603C
0.1 μ F	C24	0603C
0.1 μ F	C25	0603C
0.1 μ F	C26	0603C
0.1 μ F	C21	0603C
0.1 μ F	C22	0603C
0.1 μ F	C23	0603C
0.1 μ F	C4	0603C
0.1 μ F	C5	0603C
0.1 μ F	C6	0603C
0.1 μ F	C1	0603C
0.1 μ F	C2	0603C
0.1 μ F	C3	0603C
0.1 μ F	C7	0603C
0.01 μ F	C11	0603C
0.01 μ F	C12	0603C
0.01 μ F	C13	0603C
0.1 μ F	C8	0603C
0.1 μ F	C9	0603C
0.1 μ F	C10	0603C
0.01 μ F	C66	0603C
0.01 μ F	C65	0603C
0.01 μ F	C68	0603C
0.01 μ F	C67	0603C
0.1 μ F	C64	0603C

Part Type	Designator	Footprint
0.1 μ F	C62	0603C
0.1 μ F	C61	0603C
0.1 μ F	C39	0603C
0.1 μ F	C63	0603C
0.01 μ F	C69	0603C
0.01 μ F	C34	0603C
0.01 μ F	C35	0603C
0.01 μ F	C32	0603C
0.01 μ F	C33	0603C
0.01 μ F	C31	0603C
0.1 μ F	C100	0603C
0.01 μ F	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
1M	VBUS2R1	

Part Type	Designator	Footprint
1M	VBUS1R1	
1 μ F	VBUS1C1	
1 μ F	VBUS2C1	
4.7K	R3	0603R
4.7 μ F	C99	CASE-A
4.7 μ F	C93	CASE-A
4.7 μ F	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
470R	R14	0603R

Part Type	Designator	Footprint
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
ISPI582	ISPI582	HVQFN56/P.5N
ISPI583	ISPI583	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	
XCI7S30XL	SERIAL_PROM	DIP8
XCS30XL-5PQ240C	SPANTAN	F-QFP32X32-G240/P.5N

10. 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;

        -- ISPI582

        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 ISPI582_Address: STD_LOGIC_VECTOR(7 downto 0);
signal ISPI582_Address_Port,ISPI582_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

    ISPI582_Address <= LA(7 downto 2) & LBE(1 downto 0);

    TRST    <= '1';

    CS <= not LHOLD when ISPI582_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 ISPI582_Data_Port = asserted else (others =>
'Z');
LD(15 downto 0) <= DATA when LWR = deasserted and ISPI582_Data_Port = asserted else (others =>
'Z');

LREADY <= Address_Ready and Data_Ready and Control_Ready and DMA_Ready;

```

ISPI582_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 if;
end process;

```

```

end process;

ISPI582_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;

ISPI582_Register_Address_Process:

```

```

process(GCLK,LRESET,ISPI582_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 ISPI582_Address_Port = '1' then
                    ADDR <= LD(7 downto 0);
                    ADDR_STATE <= S1;
                end if;

            when S1 =>

                if ISPI582_Address_Port = '1' then
                    Address_Ready <= '0';
                end if;

                ADDR_STATE <= S2;

            when S2 =>

                ADDR_STATE <= S3;

            when others =>

                Address_Ready <= '1';

                if ISPI582_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;

```

ISPI582_Register_Data_Process:

```

process(LWR,ISPI582_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 ISPI582_Data_Port = '1' then -- write operation start
                    WR <= '0';
                    RD <= '1';
                    DATA_STATE <= S1;

                elsif LWR = '0' and ISPI582_Data_Port = '1' then -- read operation
                    RD <= '0';
                    WR <= '1';
                    DATA_STATE <= S1;

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

            start
        end case;
    end if;
end process;

```



```

        when S1 =>
            if LWR = '1' and ISPI582_Data_Port = '1' then -- write operation start
                WR <= '1';
                RD <= '1';

            elsif LWR = '0' and ISPI582_Data_Port = '1' then -- read operation

                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 ISPI582_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;

```

ISPI582_Address_Decompile_Process:

```

process(ADS,GCLK,LRESET)
begin

    if LRESET = '0' then

        ISPI582_Address_Port <= '0';
        ISPI582_Data_Port <= '0';
        Control_Port <= '0';
        DMA_Port <= '0';

    elsif rising_edge(GCLK) then

```

```
if ADS = '0' then

    case ISPI582_Address(7 downto 0) is

        when "00000000" => -- address port

            ISPI582_Address_Port <= '1';
            ISPI582_Data_Port <= '0';
            Control_Port <= '0';
            DMA_Port <= '0';

        when "00000010" => -- data port

            ISPI582_Address_Port <= '0';
            ISPI582_Data_Port <= '1';
            Control_Port <= '0';
            DMA_Port <= '0';

        when "00000100" => -- control port

            ISPI582_Address_Port <= '0';
            ISPI582_Data_Port <= '0';
            Control_Port <= '1';
            DMA_Port <= '0';

        when "00000110" => -- dma port

            ISPI582_Address_Port <= '0';
            ISPI582_Data_Port <= '0';
            Control_Port <= '0';
            DMA_Port <= '1';

        when others =>

            ISPI582_Address_Port <= '0';
            ISPI582_Data_Port <= '0';
            Control_Port <= '0';
            DMA_Port <= '0';

    end case;

else

    ISPI582_Address_Port <= ISPI582_Address_Port;
    ISPI582_Data_Port <= ISPI582_Data_Port;
    Control_Port <= Control_Port;
    DMA_Port <= DMA_Port;

    if LHOLD = '0' then
        ISPI582_Address_Port <= '0';
        ISPI582_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;
            end case;
        end if;
    end if;
end process;

```

```

        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;

```

Note: DMA is not used.

11. 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 ISPI582 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*.

12. Reference

- *ISPI582 Hi-Speed Universal Serial Bus interface device datasheet.*
- *ISPI582 PCI Mass Storage Kit Firmware Guide.*
- *PLX Technology PCI9054 PCI to Local Bus Bridge datasheet and user manual*

Philips Semiconductors

Philips Semiconductors is a worldwide company with over 100 sales offices in more than 50 countries. For a complete up-to-date list of our sales offices please e-mail sales.addresses@www.semiconductors.philips.com. A complete list will be sent to you automatically. You can also visit our website <http://www.semiconductors.philips.com/sales/>

www.semiconductors.philips.com

© **Koninklijke Philips Electronics N.V. 2003**

All rights reserved. Reproduction in whole or in part is prohibited without the prior written consent of the copyright owner. The information presented in this document does not form part of any quotation or contract, is believed to be accurate and reliable and may be changed without notice. No liability will be accepted by the publisher for any consequence of its use. Publication thereof does not convey or imply any license under patent – or other industrial or intellectual property rights.

