Semiconductors

UM10038_2

ISP1583 Hi-Speed USB Device Split Bus Eval Kit

October 2003

User's Guide

Rev 2.0

Revision History:

Version	Date	Description	Author
2.0	September 2003	Updated the following: • All figures related to the eval kit. • Section 3. • Section 5.1. • Section 6.2. • Section 6.5. • Section 8.3. • Section 9. • Section 11.	Albert Goh
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 ISP1583 Hi-Speed USB Device Split Bus Eval Kit enables you to evaluate the features of the ISP1583—a Hi-Speed Universal Serial Bus (USB) device that supports Generic Mode and Split Bus mode CPU interface and direct interface to any ATA/ATAPI device—in Split Bus mode, that is, as a multiplexed address and data bus. Evaluate the ISP1583 as a Generic Direct Memory Access device (Split Bus kit).

This Split Bus eval board has onboard the ISP1583, Xilinx® XC95288XL, SRAM, and 8051 series microcontroller. The kit allows you to connect the ISP1583 to any generic processor when it is configured to the separate address and data bus mode (Generic Processor Mode).

Figure 1-1 shows the ISP1583 Split Bus board.

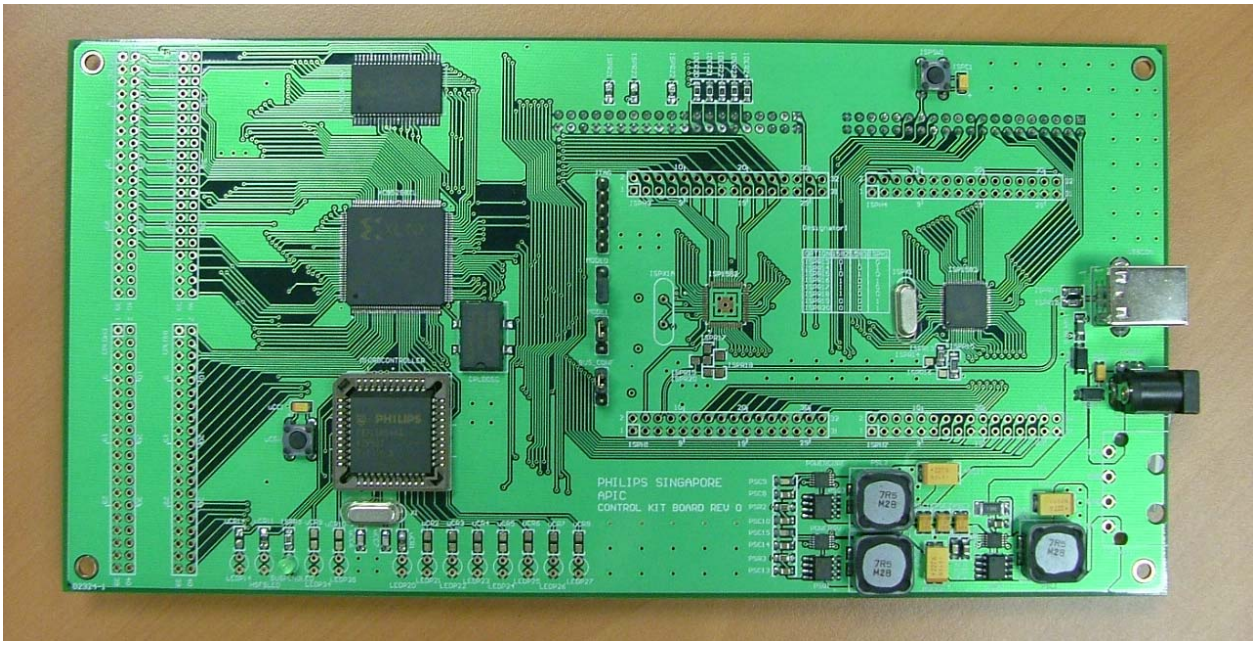


Figure 1-1: ISP1583 Split Bus Board

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

- 12 V DC power supply

Firmware

- Keil C Compiler*
- Firmware for Split Bus eval kit

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

3. Block Diagram

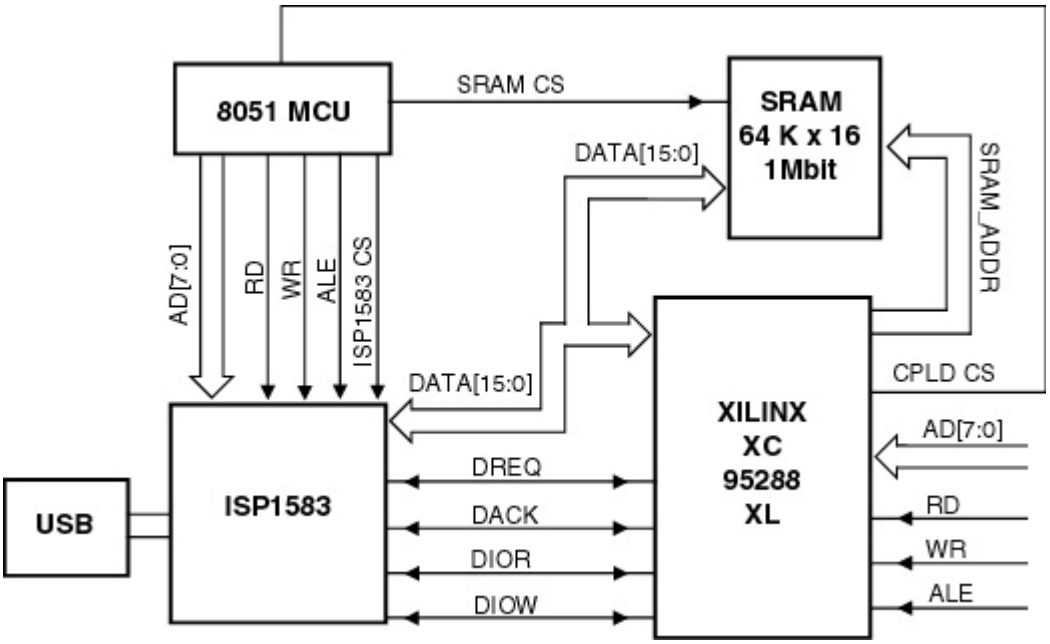


Figure 3-1: ISP1583 Split Bus Eval Board Block Diagram

Figure 3-1 shows the ISP1583 configured to operate in the Split Bus mode. The Xilinx XC95288XL acts as the local DMA Controller. On the split bus kit, the data from the DMA or PIO access is stored in the SRAM.

4. PCB Layout

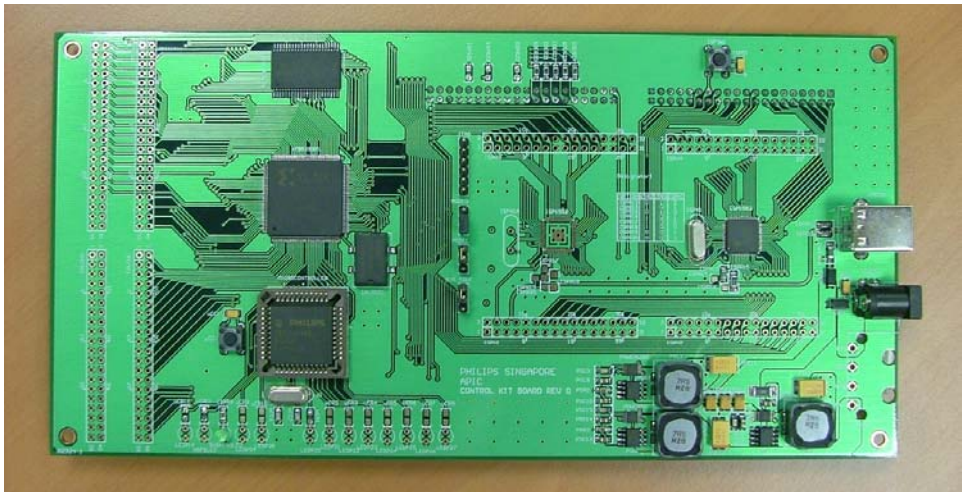


Figure 4-1: ISP1583 Split Bus Eval Board

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

5. Component Placement

5.1. ISP1583

Figure 5-1: ISP1583: At upper-right corner of the eval board

5.2. ISP1582

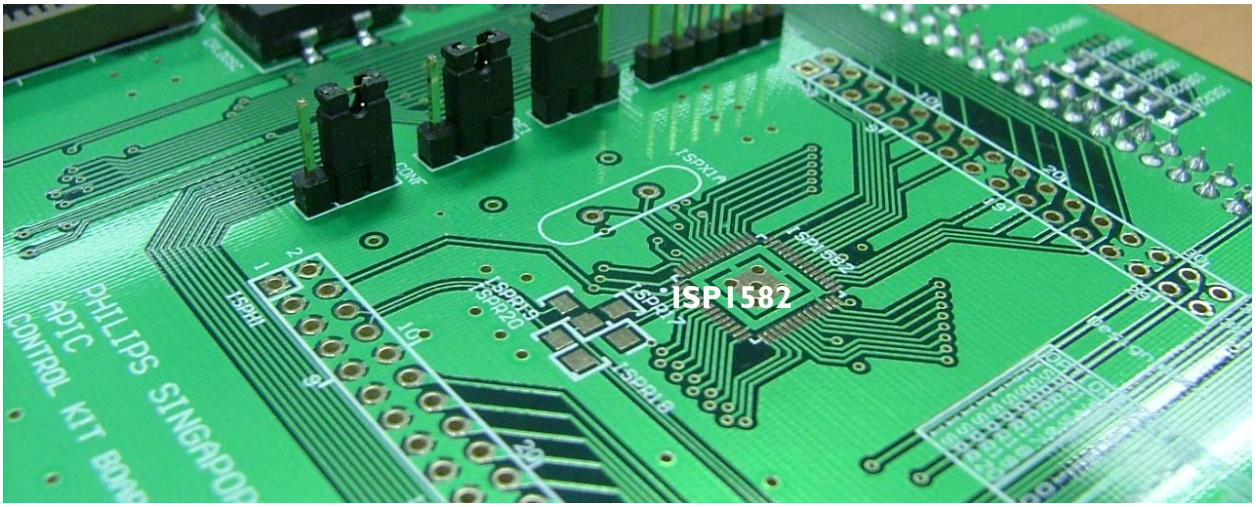


Figure 5-2: ISP1582 footprint

There is a footprint for ISP1582 to cater for future expansion of the Split Bus kit for ISP1582.

5.3. Xilinx XC95288XL CPLD

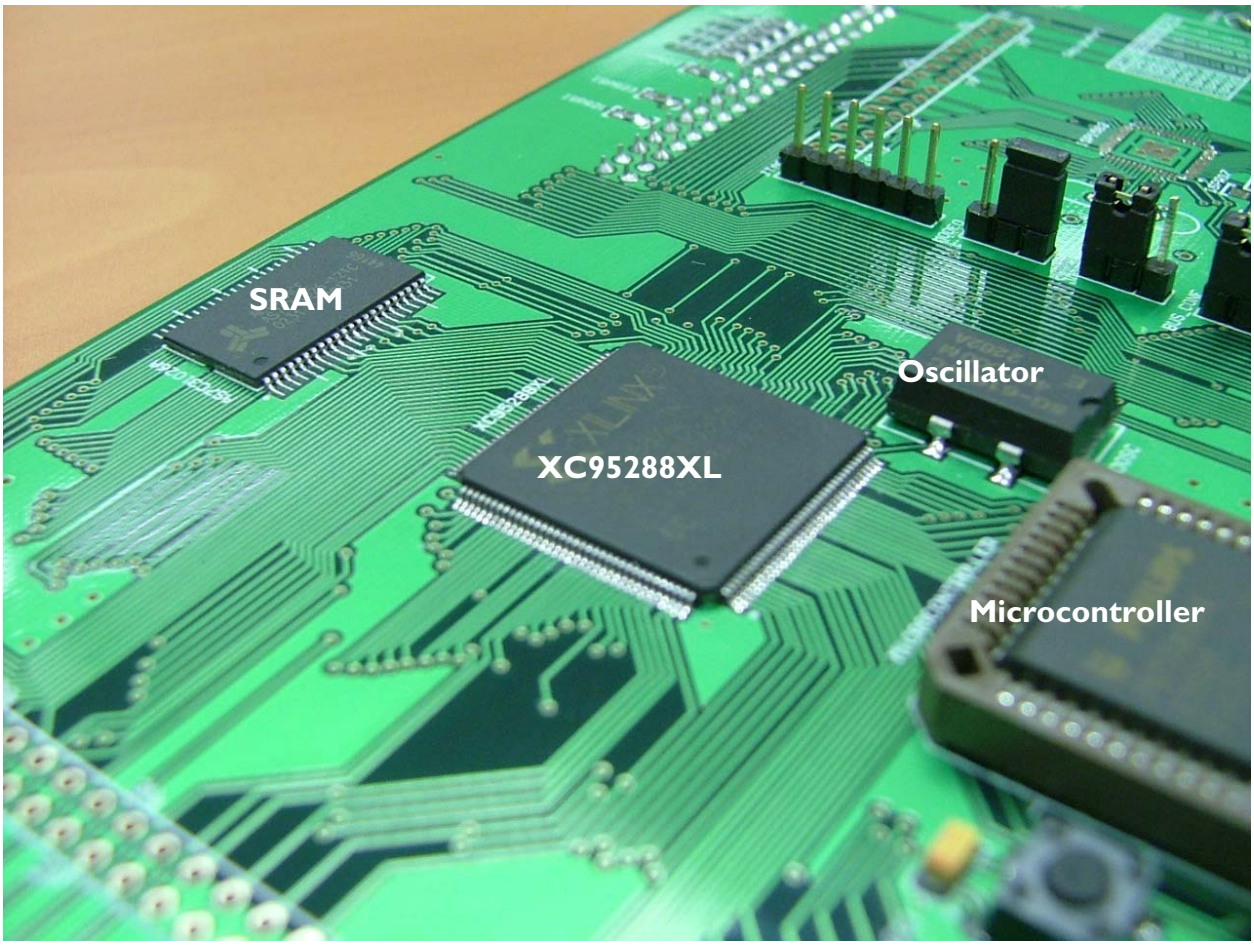


Figure 5-3: Xilinx XC95288XL

Xilinx XC95288XL acts as the Generic DMA controller for DMA transfer.

6. Header and Connector Placement

6.1. USB and DC Power Input Supply Connectors

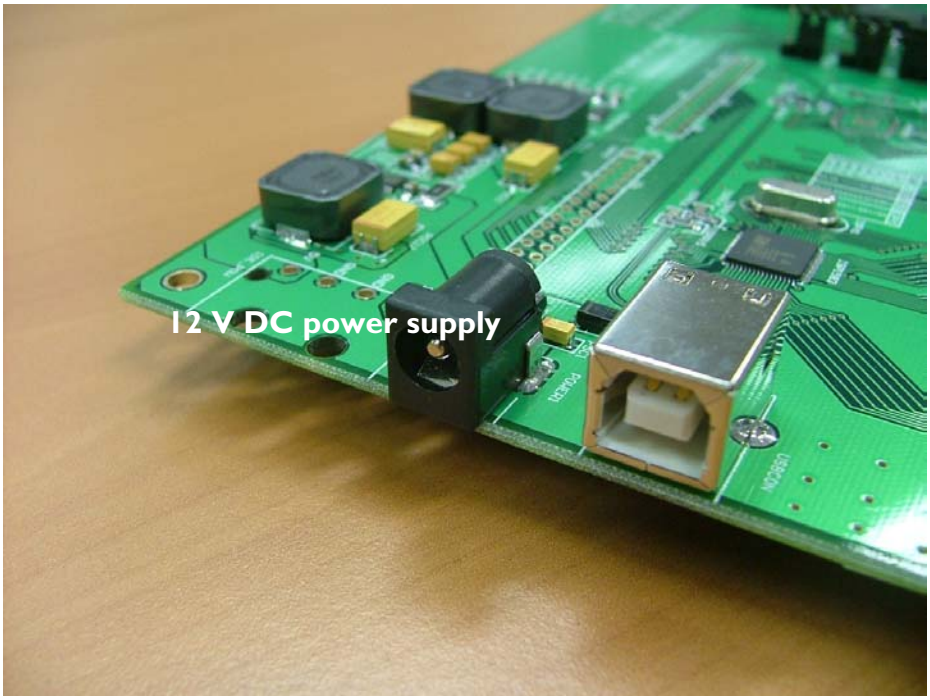
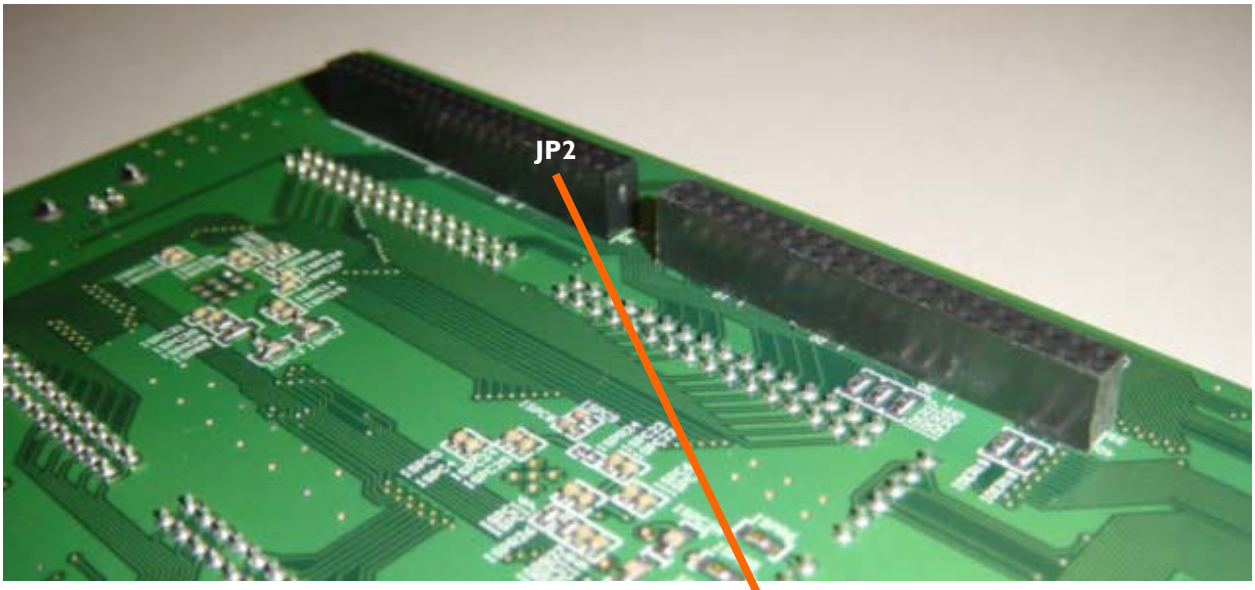


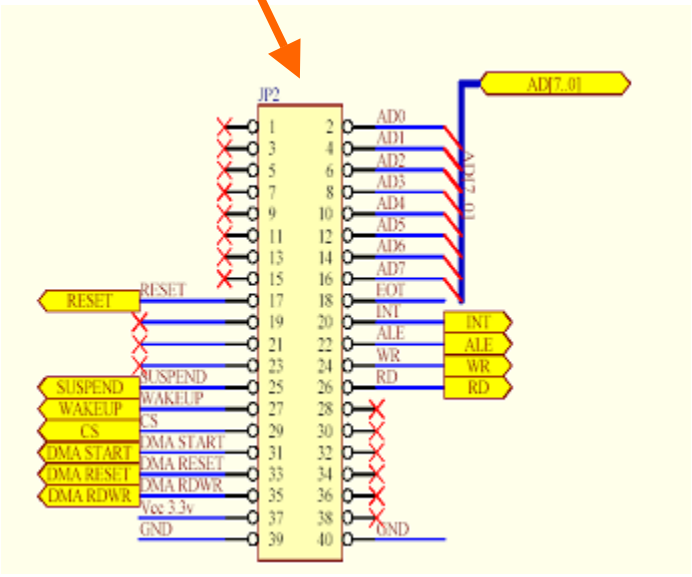
Figure 6-1: DC power supply input and USB connectors

The ISP1583 USB connector is next to the 12-volt DC power supply input.

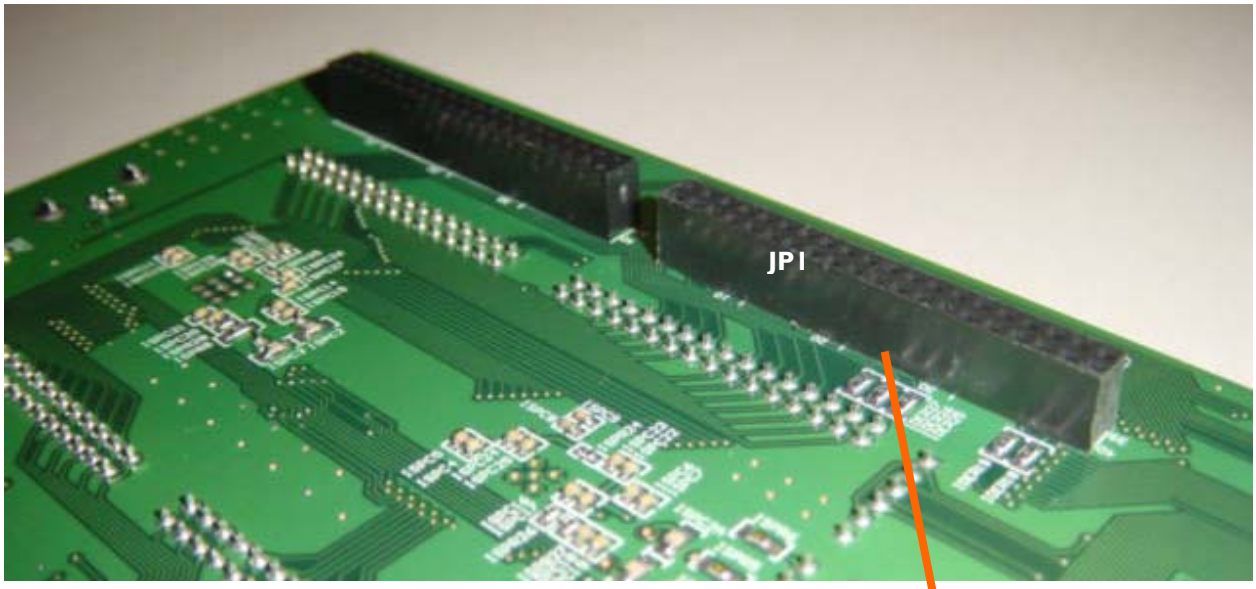
6.2. ISP1583 Processor Expansion bus



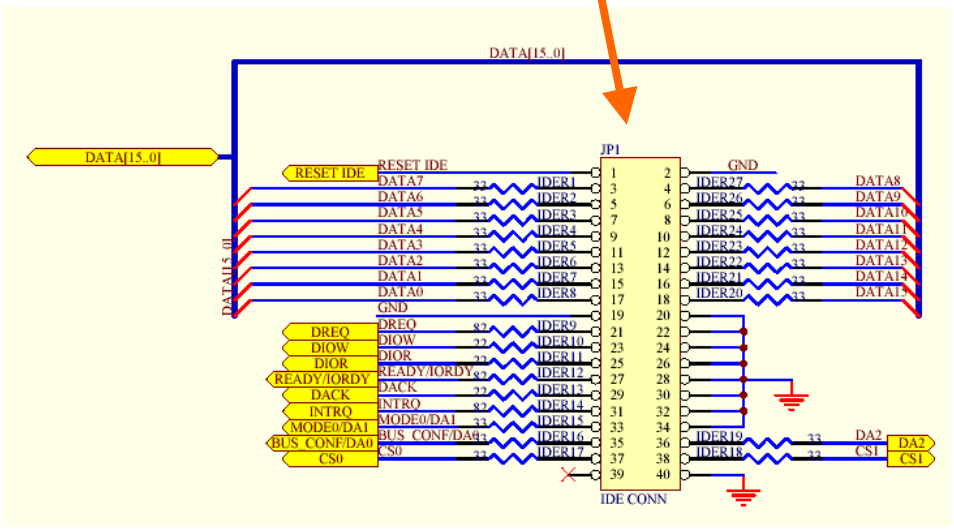
Act as an expansion bus for connecting to another processor or microcontroller.



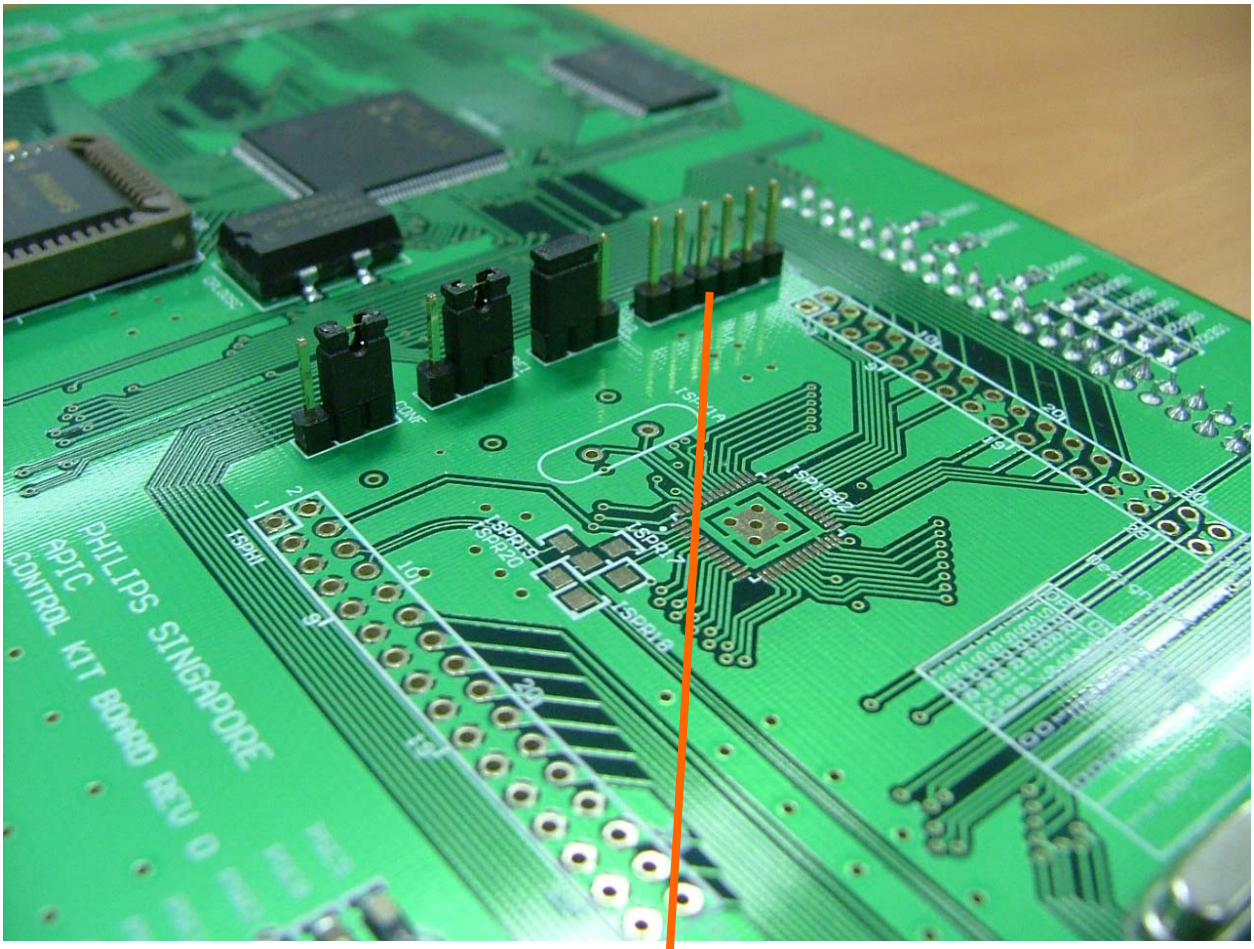
6.3. ISP1583 DMA Expansion Bus



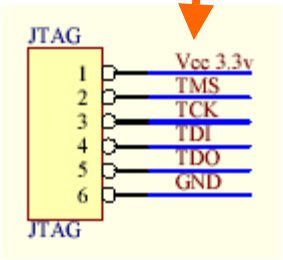
Acts as a DMA expansion bus for connecting to an external DMA controller and the ISP1583's 16-bit data bus.



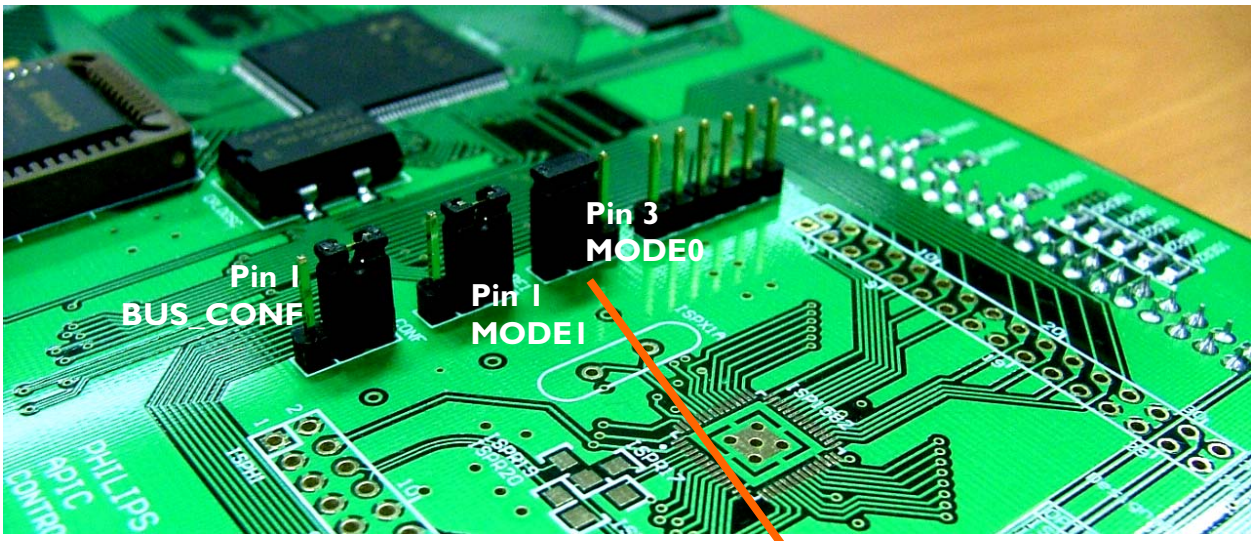
6.4. JTAG Header



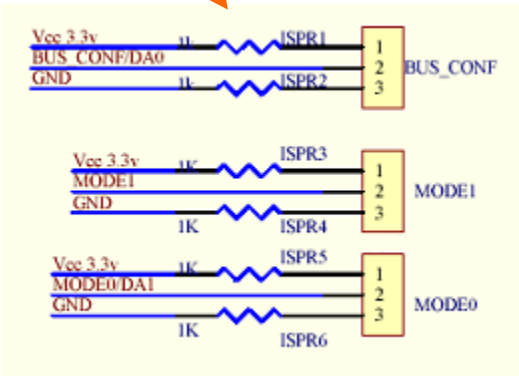
The JTAG header allows you to reprogram the Xilinx XC95288XL for user-defined operations.



6.5. ISP1583 Processor Selector

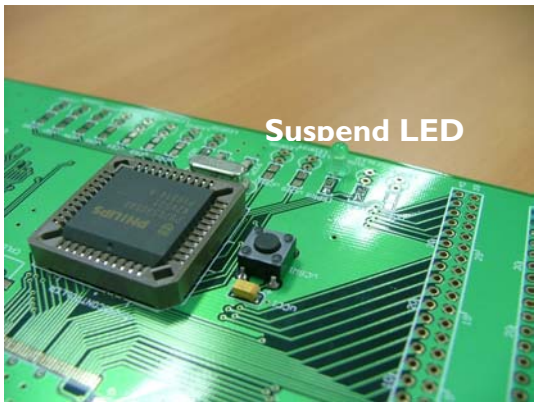


The ISP1583 Split Bus eval kit is configured to run under the multiplexed 8-bit address and data bus (Split Bus Mode).

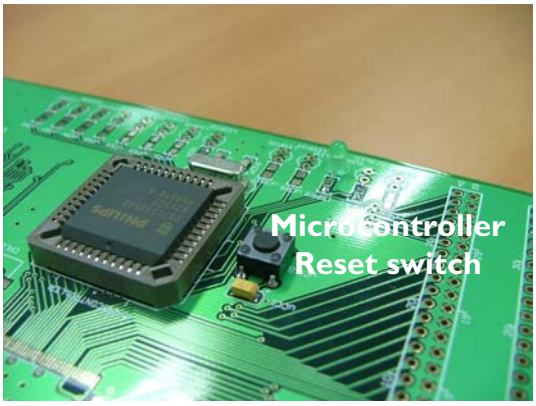


Processor Mode	Bus Config Pin	Mode 1 Pin	Mode 0 Pin
Split Bus Mode	2 - 3	2 - 3	1 – 2

7. Switch and LED Placement



The Wake-Up switch is tied to the ISP1583's wake-up pin, which will wake up the ISP1583 when it is in suspend mode. The Suspend LED when lit indicates that the ISP1583 is in the suspend mode.

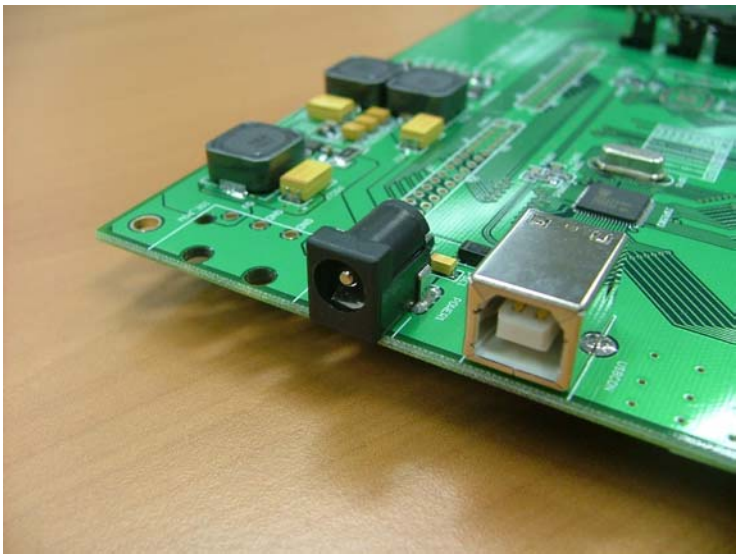


The Microcontroller Reset switch resets the microcontroller, which in turn resets the ISP1583.

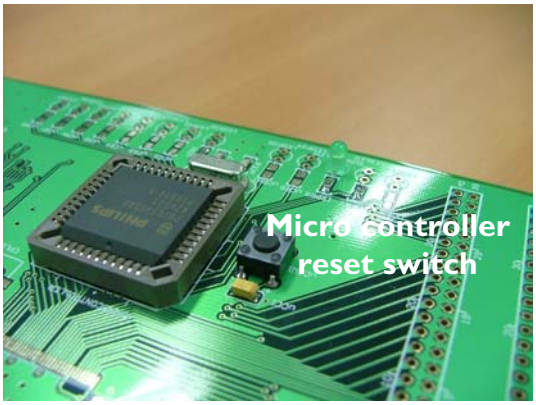
8. ISP1583 Split Bus Eval Kit Setup Procedure

8.1. Split bus Kit Setup Procedure

1. Insert the 12-volt DC power supply that is supplied together with the kit to the DC jack and switch on the power.



2. Press the Microcontroller Reset switch.



3. Plug in the USB cable to the ISP1583 USB connector.
4. After successful enumeration, run the USB Device applet on the host PC.
5. **Caution**: Make sure that the Bus Config, Mode 0 and Mode 1 pin are at the default setting.

Processor Mode	Bus Config Pin	Mode 1 Pin	Mode 0 Pin
Split Bus Mode	2 - 3	2 - 3	1 – 2

8.2. Split Bus Kit Host PC Setup and Bus Enumeration Procedure

If it is the first time the evaluation board is connected 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 ISP1583 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.

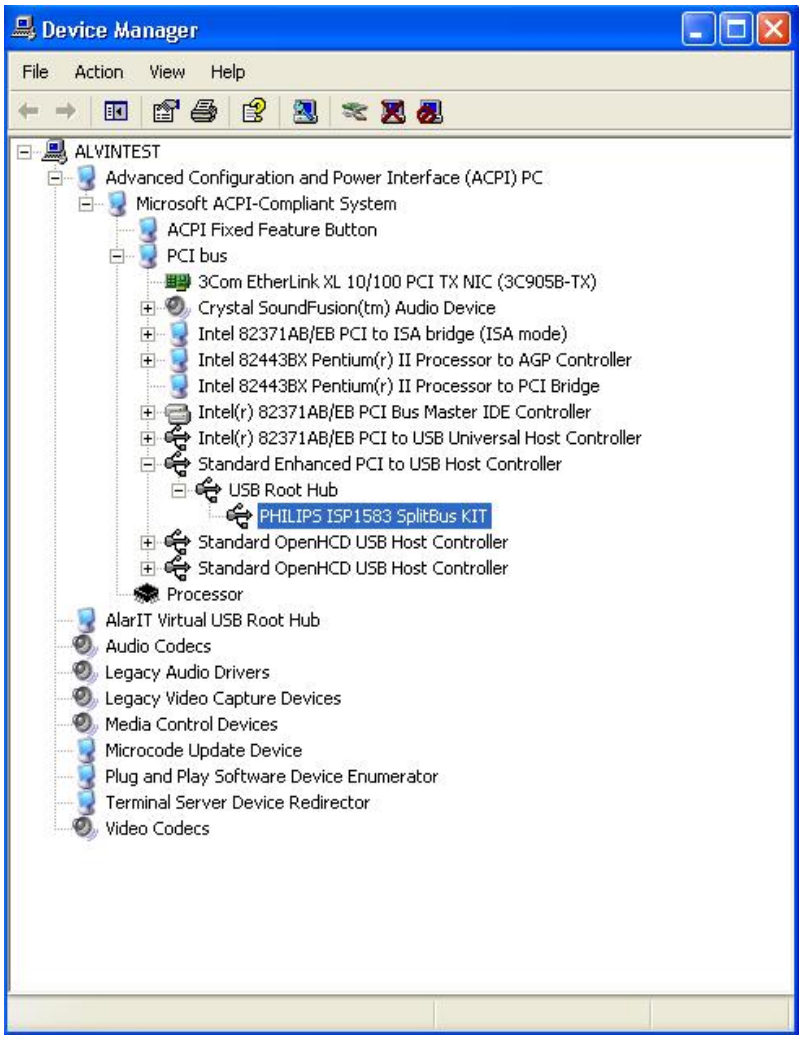


Figure 8-1: Hi-Speed USB Device on Philips ISP1561 EHCI Hi-Speed USB Controller

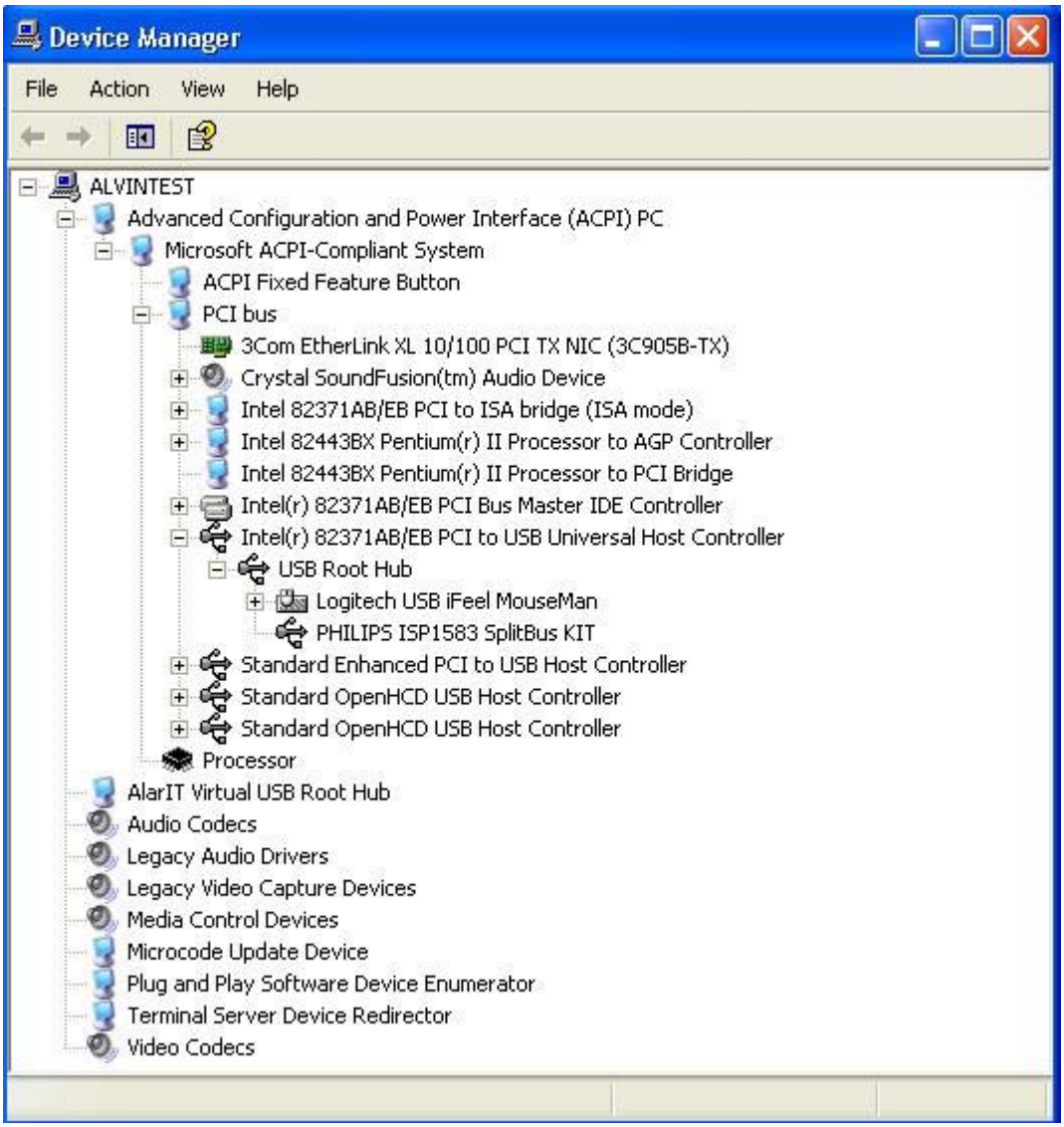
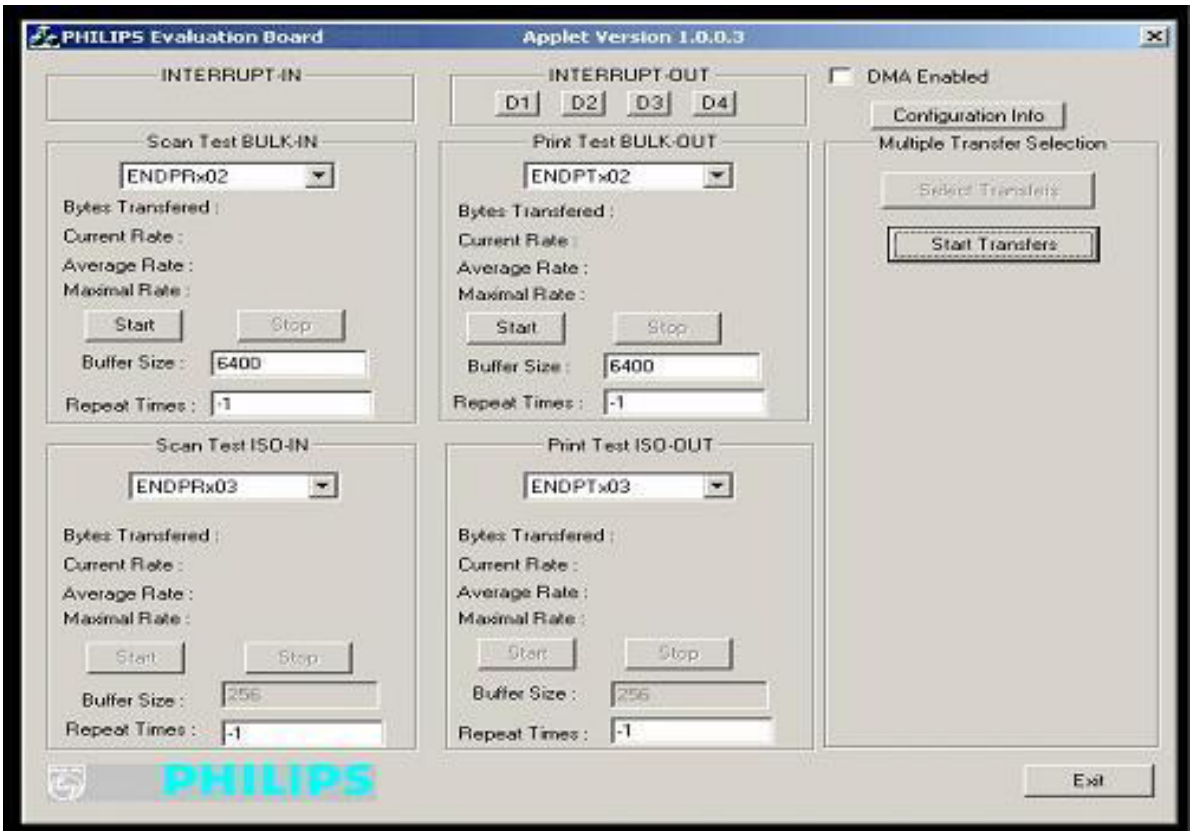
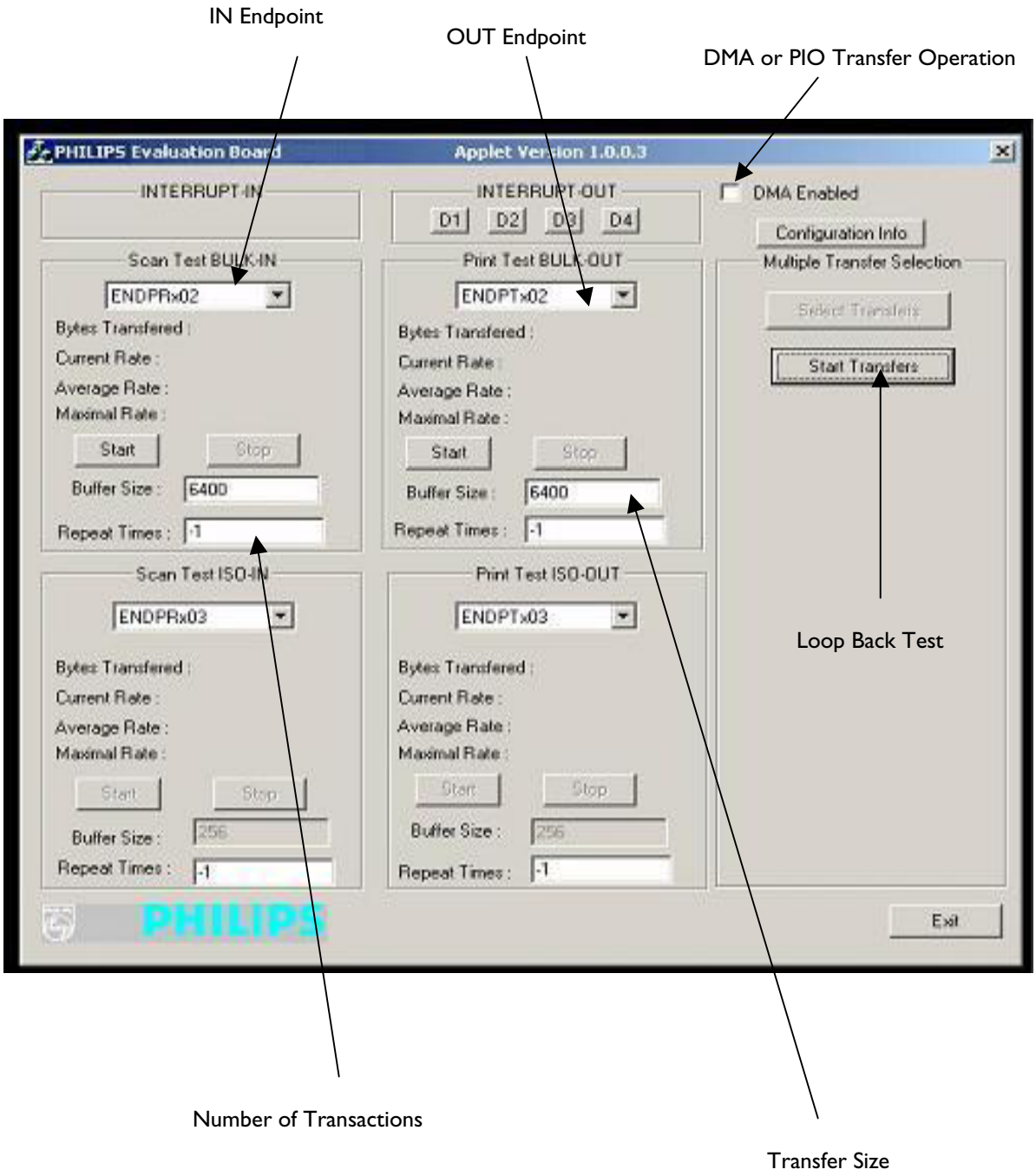


Figure 8-2: Original USB Device on Intel UHCI USB Controller

8.3. Split Bus Kit Test Application

This application allows you to perform data transfer using the GDMA slave mode of the ISP1583. It sends data to the ISP1583 and reads it back, and so checks for data integrity.





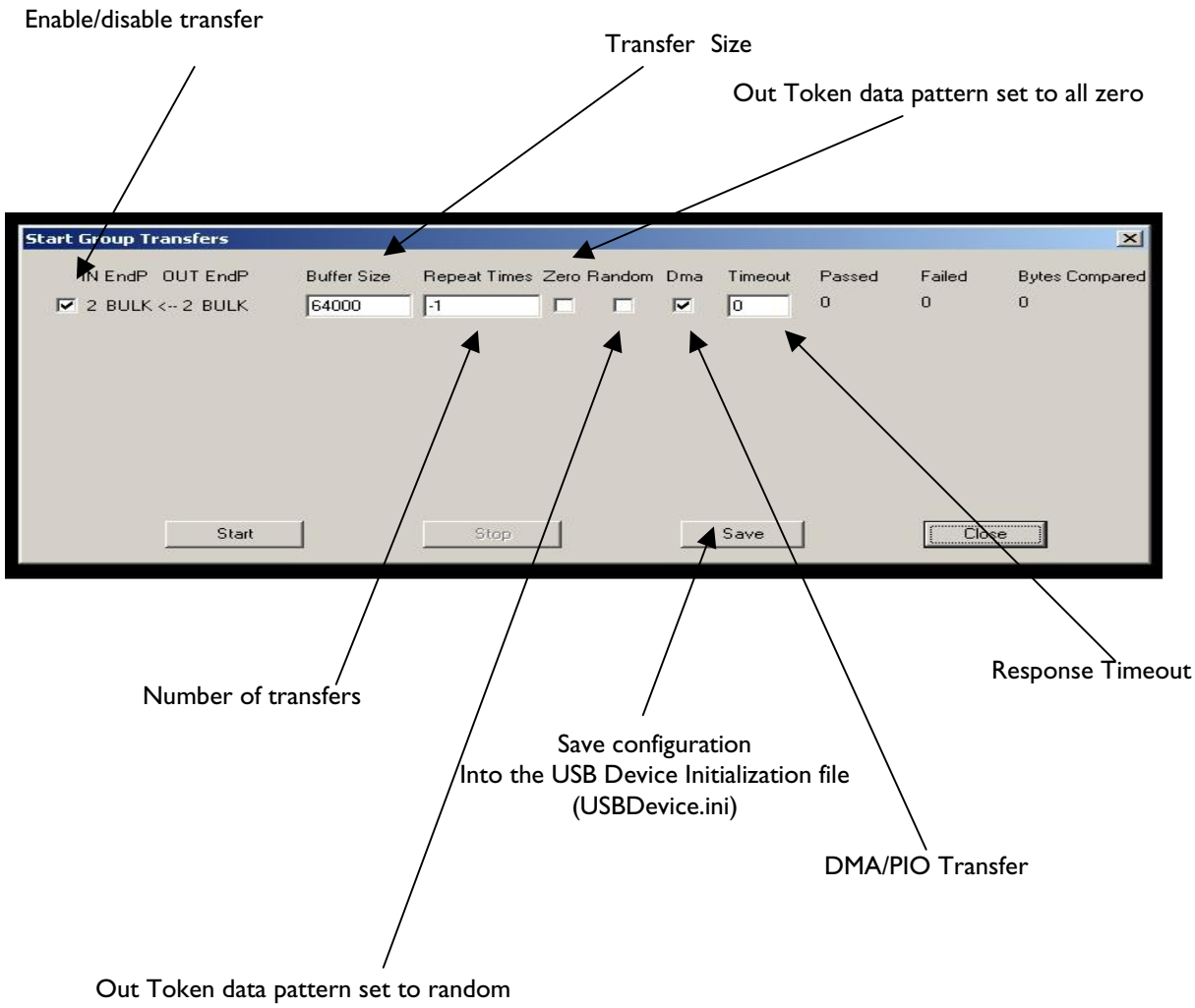


Table 8-1: Endpoint Description

Pipe Number	Endpoint Type	Operations
2*	Bulk-Out	This pipe is defined as Bulk Out pipe. Test applet and the ISP1583 evaluation board supports three test modes: loopback 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 ISP1583 evaluation board supports three test modes: loopback 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 a different endpoint configuration.

Three test modes:

- Scan mode: The ISP1583 evaluation board acts like a split bus. It sends data packets to the host PC as fast as possible. This mode is used to evaluate the Bulk In transfer rate.
- Print mode: The ISP1583 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 Bulk Out transfer rate.
- Loop back mode: In this mode, the ISP1583 evaluation board receives data packets on Isochronous/Bulk Out endpoint and sends them back to the host PC on Bulk In endpoint. This mode is used to test the data integrity of transfers.

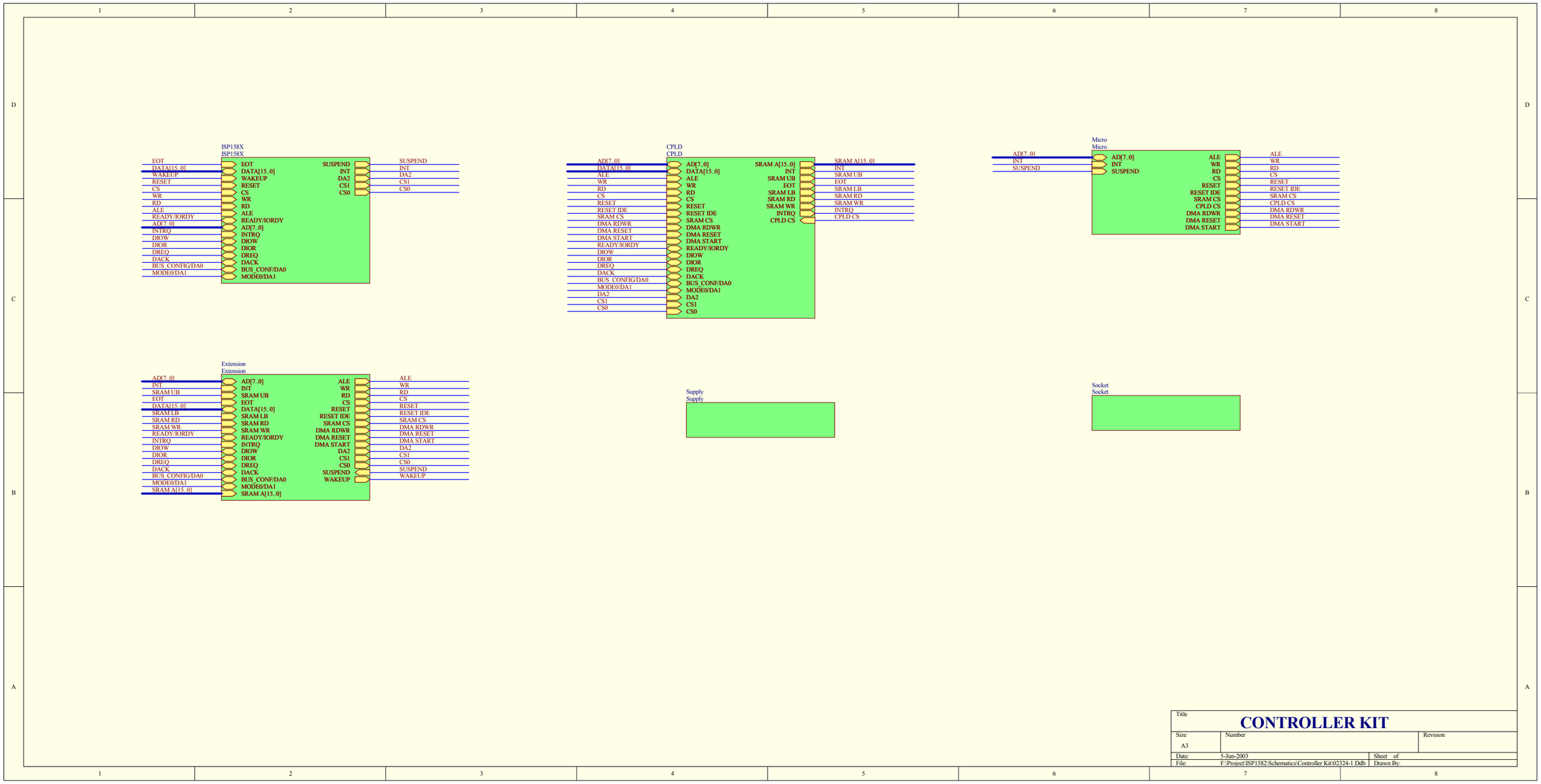
The "Buffer Size" setting on the test applet is determined by the firmware and hardware ability of the evaluation board. For the Split bus kit, the maximal size is limited to 64000 for Bulk transfer.

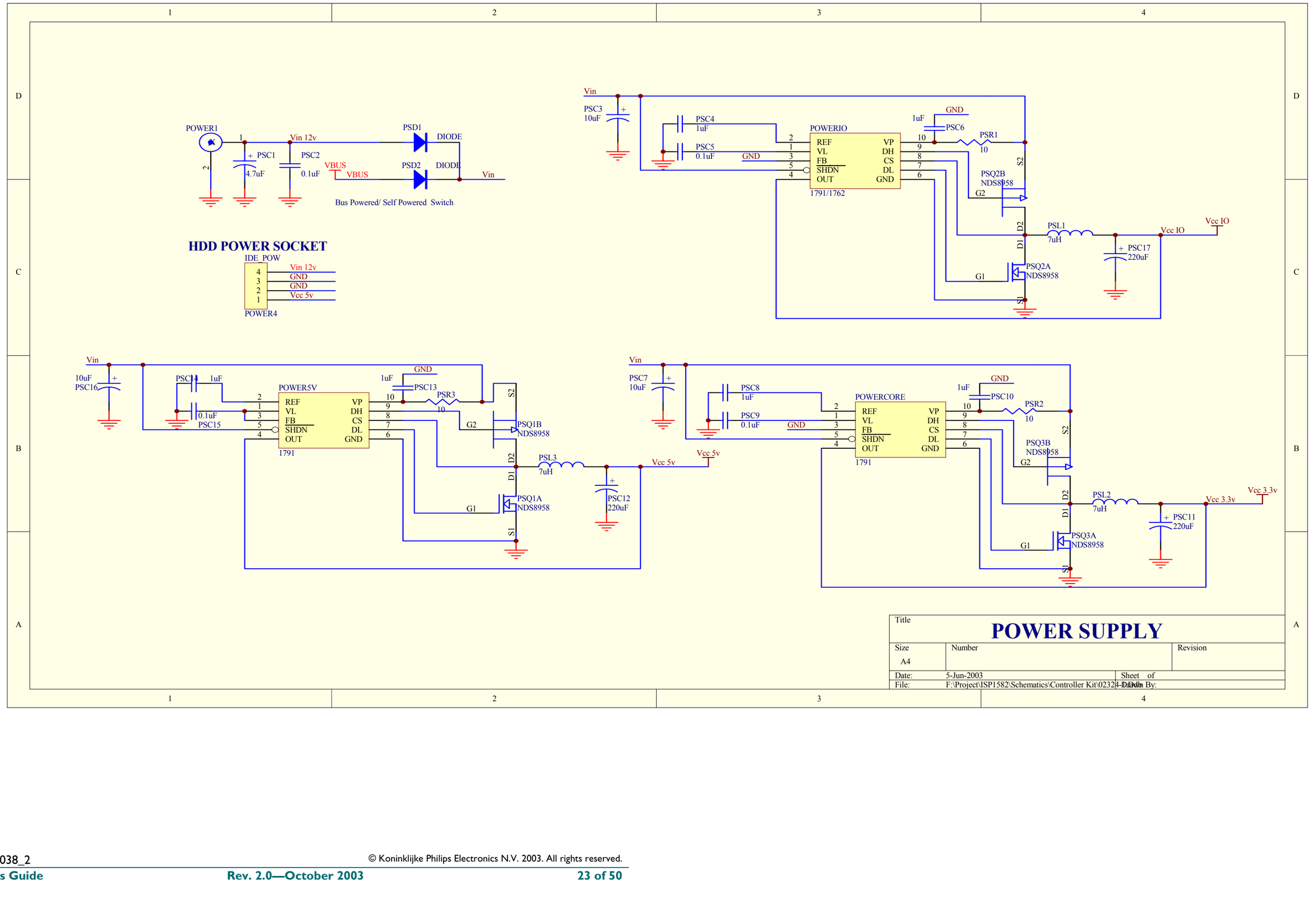
The "Repeat Times" for loop-back test controls the numbers of iterations of loop-back, which is useful for debugging. "-1" means it is infinite.

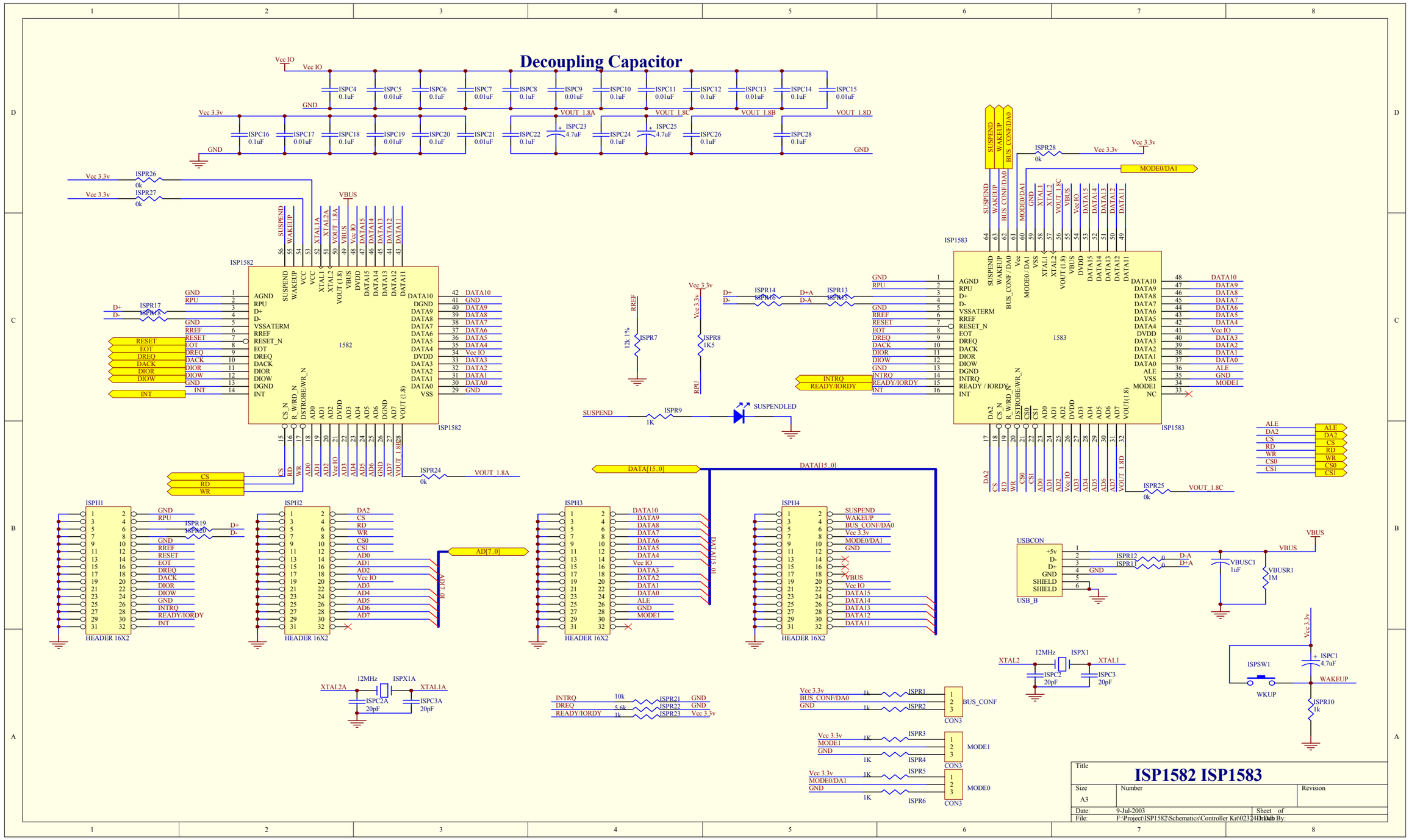
Note: At a particular time, you can only perform either scan or print. Both these operations cannot be performed simultaneously.

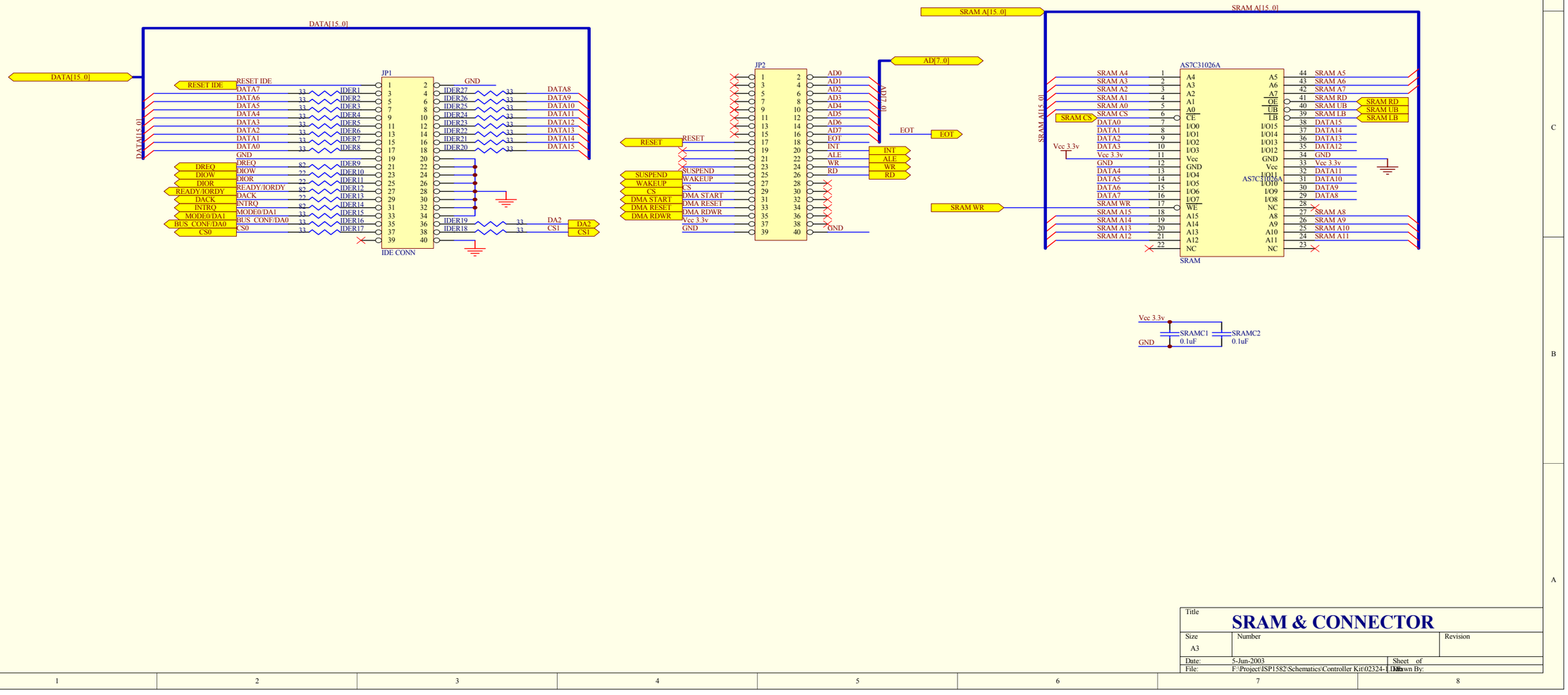
9. Schematics

9.1. ISP1583 Split Bus Eval Board

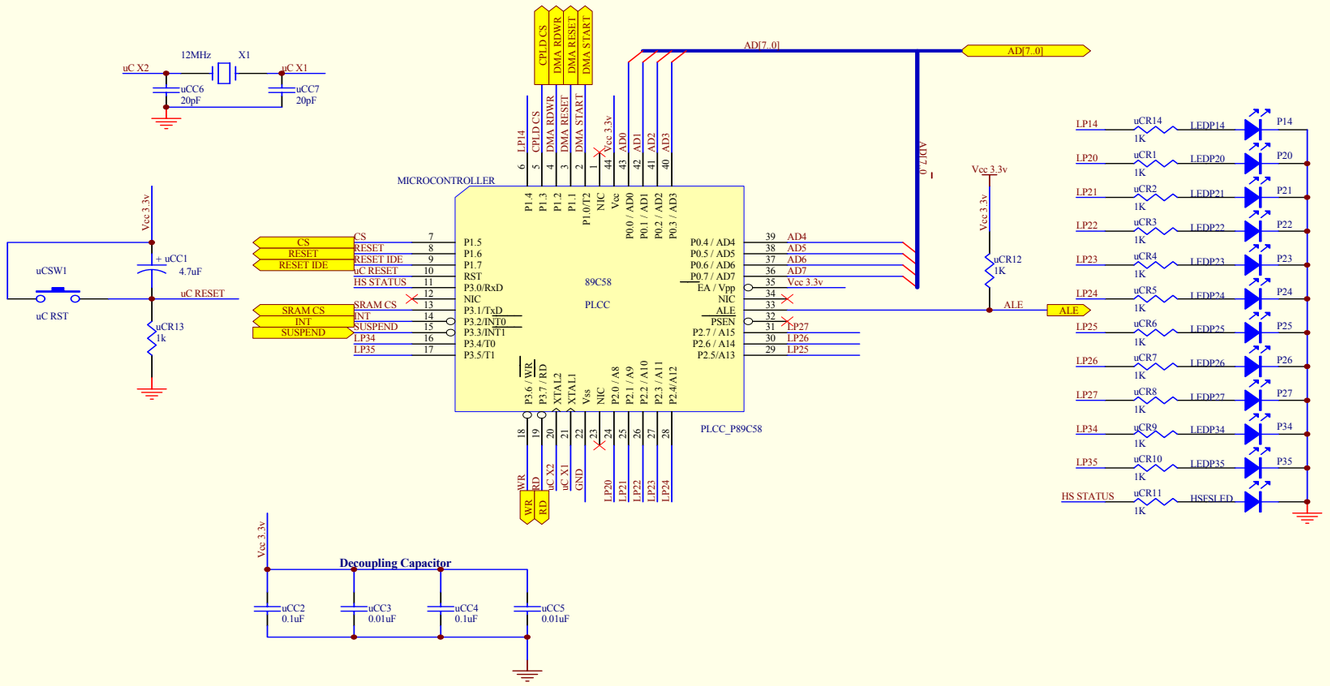






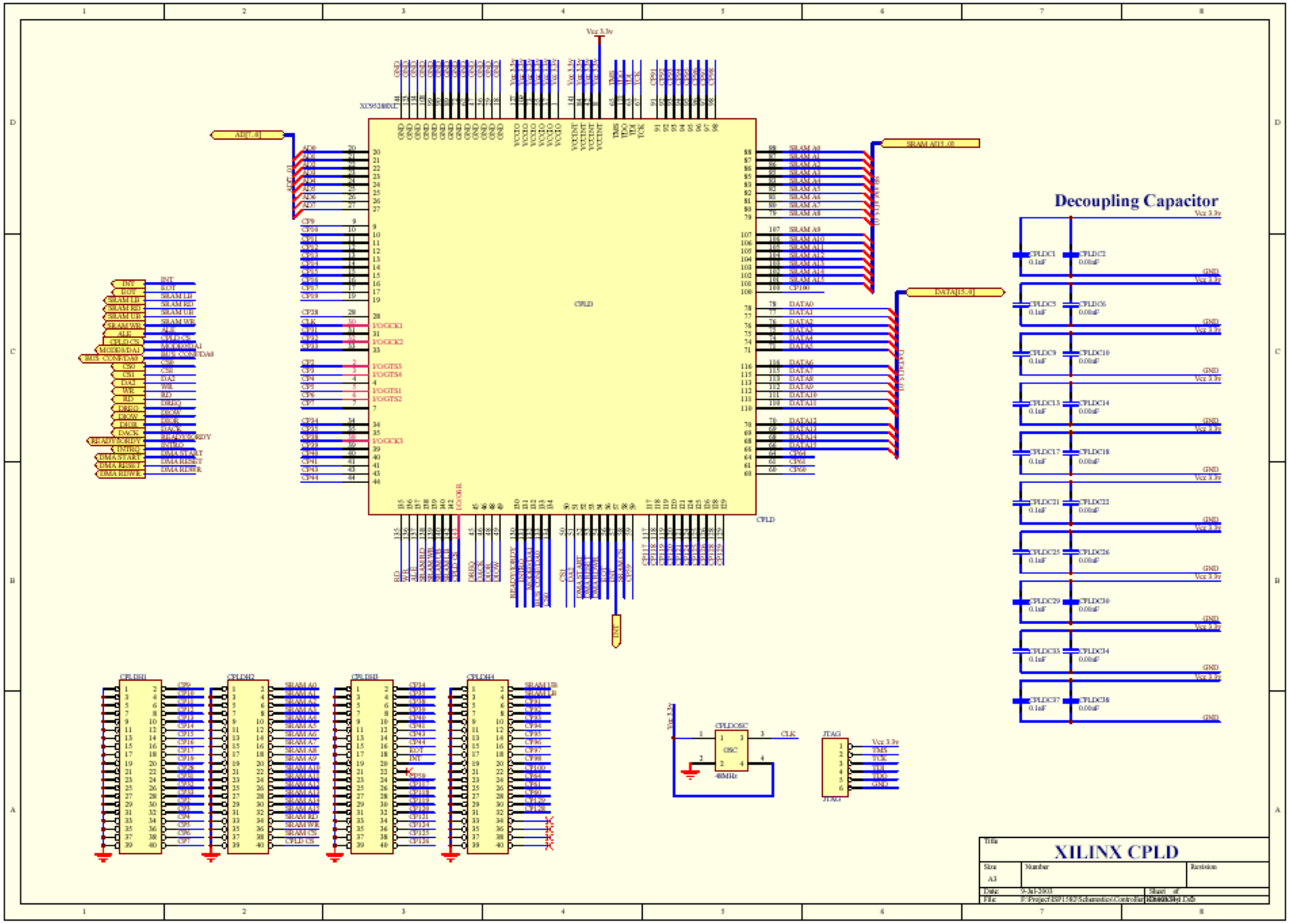


Title	SRAM & CONNECTOR
Size	A3
Number	
Revision	
Date:	5-Jun-2003
Sheet of	
File:	F:\Project\ISP1582\Schematics\Controller Kit\02324-1.Dra
Drawn By:	



Title: **MICRO CONTROLLER**

Size	Number	Revision
A3		
Date: 5-Jun-2003	Sheet of	
File: F:\Project\ISP1582\Schematics\Controller Kit\02324-1.DDB	Drawn By:	



10. Bill Of Material

10.1. ISP1583 Split Bus Eval Board

Table 10-1: Bill of Material of the ISP1583 Split Bus Eval Board

Part Type	Designator	Footprint
0	ISPR11	805
0	ISPR12	805
0.01µF	SC17	603
0.1µF	SC18	603
0.01µF	SC15	603
0.1µF	SC16	603
0.01µF	SC19	603
0.1µF	ISPC4	603
0.01µF	ISPC5	603
0.1µF	SC20	603
0.01µF	SC21	603
0.1µF	SC14	603
0.01µF	SC7	603
0.1µF	SC8	603
0.01µF	SC5	603
0.1µF	SC6	603
0.01µF	SC9	603
0.1µF	SC12	603
0.01µF	SC13	603
0.1µF	SC10	603
0.01µF	SC11	603
0.1µF	ISPC6	603
0.01µF	ISPC19	603
0.1µF	ISPC20	603
0.01µF	ISPC17	603
0.1µF	ISPC18	603
0.01µF	ISPC21	603
0.1µF	ISPC26	603
0.1µF	ISPC28	603
0.1µF	ISPC22	603
0.1µF	ISPC24	603
0.1µF	ISPC16	603
0.01µF	ISPC9	603
0.1µF	ISPC10	603
0.01µF	ISPC7	603
0.1µF	ISPC8	603
0.01µF	ISPC11	603
0.1µF	ISPC14	603

Part Type	Designator	Footprint
0.01µF	ISPC15	603
0.1µF	ISPC12	603
0.01µF	ISPC13	603
0.1µF	SC4	603
0.01µF	CPLDC10	805
0.01µF	CPLDC6	805
0.01µF	CPLDC2	805
0.1µF	CPLDC1	805
0.01µF	CPLDC18	805
0.01µF	CPLDC14	805
0.1µF	PSC15	805
0.1µF	PSC5	805
0.01µF	uCC3	805
0.1µF	uCC2	805
0.1µF	PSC9	805
0.01µF	uCC5	805
0.1µF	uCC4	805
0.1µF	CPLDC5	805
0.1µF	CPLDC25	805
0.1µF	CPLDC21	805
0.01µF	CPLDC38	805
0.1µF	CPLDC37	805
0.1µF	CPLDC33	805
0.1µF	CPLDC29	805
0.01µF	CPLDC34	805
0.1µF	CPLDC17	805
0.1µF	CPLDC13	805
0.1µF	CPLDC9	805
0.01µF	CPLDC30	805
0.01µF	CPLDC26	805
0.01µF	CPLDC22	805
0.1µF	SC1A	805
0.01µF	SC1B	805
0.1µF	PSC2	805
0.1µF	SRAMC1	805
0.1µF	SRAMC2	805
0.1µF	SC11	805
0.01µF	SC1D	805
0.1µF	SC1C	805
0.01µF	SC1F	805
0.01µF	SC1H	805
0.1µF	SC1G	805
0k	ISPR24	805
0k	ISPR25	805

Part Type	Designator	Footprint
0k	ISPR28	805
0k	ISPR27	805
0k	ISPR26	805
1K5	ISPR8	805
1K	uCR12	805
1K	ISPR3	805
1K	uCR11	805
1K	uCR3	805
1K	uCR2	805
1K	uCR1	805
1K	ISPR4	805
1K	uCR5	805
1K	uCR7	805
1K	uCR6	805
1K	ISPR5	805
1K	ISPR6	805
1K	uCR4	805
1K	uCR14	805
1K	uCR9	805
1K	ISPR9	805
1K	uCR10	805
1K	uCR8	805
1M	VBUSR1	
1k	ISPR2	805
1k	ISPR1	805
1k	ISPR10	805
1k	uCR13	805
1k	ISPR23	805
1µF	VBUSC1	
1µF	PSC14	805
1µF	PSC10	805
1µF	PSC13	805
1µF	PSC6	805
1µF	PSC4	805
1µF	PSC8	805
4.7µF	ISPC23	
4.7µF	ISPC25	
4.7µF	ISPC1	CASE A
4.7µF	uCC1	CASE A
4.7µF	PSC1	CASE A
5.6k	ISPR22	805
7uH	PSL3	CDRH125
7uH	PSL1	CDRH125
7uH	PSL2	CDRH125

Part Type	Designator	Footprint
10	PSR2	805
10	PSR3	805
10	PSR1	805
10k	ISPR21	805
10µF	PSC16	CASE B
10µF	PSC3	CASE B
10µF	PSC7	CASE B
12MHz	X1	XTAL-CSM4A
12MHz	SX1	XTAL-HC49/4H
12MHz	SX1B	XTAL-HC49/4H
12MHz	ISPX1	XTAL-HC49/4H
12MHz	ISPX1A	XTAL-HC49/4H
12k 1%	ISPR7	805
20pF	SC3B	805
20pF	SC2B	805
20pF	ISPC2A	805
20pF	SC3	805
20pF	SC2	805
20pF	ISPC3A	805
20pF	uCC7	805
20pF	ISPC3	805
20pF	ISPC2	805
20pF	uCC6	805
22	IDER11	805
22	IDER10	805
22	IDER13	805
33	IDER21	805
33	IDER22	805
33	IDER19	805
33	IDER18	805
33	IDER7	805
33	IDER3	805
33	IDER4	805
33	IDER5	805
33	IDER2	805
33	IDER8	805
33	IDER6	805
33	IDER1	805
33	IDER23	805
33	IDER17	805
33	IDER20	805
33	IDER16	805
33	IDER15	805
33	IDER25	805

Part Type	Designator	Footprint
33	IDER24	805
33	IDER27	805
33	IDER26	805
48MHz	CPLDOSC	XTAL-SG615
82	IDER9	805
82	IDER14	805
82	IDER12	805
220µF	PSC17	CASE D
220µF	PSC12	CASE D
220µF	PSC11	CASE D
1791	POWER5V	1791
1791	POWERCORE	1791
1791/1762	POWERIO	1791
CON3	MODE1	CON3
CON3	MODE0	CON3
CON3	BUS_CONF	CON3
CPLD	XC95288XL	TQFP_144
DIODE	PSD2	SMA
DIODE	PSD1	SMA
HEADER 16X2	ISPH3	HEADER 16X2
HEADER 16X2	ISPH1	HEADER 16X2
HEADER 16X2	ISPH2	HEADER 16X2
HEADER 16X2	ISPH4	HEADER 16X2
HEADER 16X2	H1	HEADERB 16X2
HEADER 16X2	H2	HEADERB 16X2
HEADER 16X2	H3	HEADERB 16X2
HEADER 16X2	H4	HEADERB 16X2
IDE CONN	JP1	HEADERB 20X2
ISP1582	ISP1582	HVQFN56-SMT
ISP1582	SOCKET1582	SOCKET56
ISP1583	ISP1583	LQFP64-SMT
ISP1583	SOCKET1583	SOCKET64
JTAG	JTAG	HEADER 6
NDS8958	PSQ1	NDS8958
NDS8958	PSQ3	NDS8958
NDS8958	PSQ2	NDS8958
P14	LEDP14	LED
P20	LEDP20	LED
P21	LEDP21	LED
P22	LEDP22	LED
P23	LEDP23	LED
P24	LEDP24	LED
P25	LEDP25	LED
P26	LEDP26	LED

Part Type	Designator	Footprint
P27	LEDP27	LED
P34	LEDP34	LED
P35	LEDP35	LED
PLCC_P89C58	MICROCONTROLLER	PLCC44
POWER4	IDE_POW	POWER4
SRAM	AS7C31026A	TSOP44
USB_B	USBCON	USB_A
WKUP	ISPSW1	SW-TACT
uC RST	uCSW1	SW-TACT

11. Xilinx® XC95288XL DMA Controller VHDL Code

The following code integrates the DMA slave, DMA master and PIO.

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

entity topLevel is
    port (

--            Global Clock Input
              CLK:          in STD_LOGIC;
              DBG_CLK:      out STD_LOGIC;
              DBG_STATE:    out STD_LOGIC_VECTOR(1 downto 0);

--            Microcontroller burst and mode access
              uC_AD:        inout STD_LOGIC_VECTOR (7 downto 0);
              CS:           in STD_LOGIC;
              ALE:          in STD_LOGIC;
              uC_WRITE:     in STD_LOGIC;

--            DMA burst and mode selection
              WIDTH:        out STD_LOGIC;                          -- debug pin
              MODE_OUT:     OUT STD_LOGIC_VECTOR (1 downto 0);      -- debug pin

--            DMA control signal
-- ***************************************************************************
--  ISP158x_MS * DMA_START * DMA_RDWR * DMA_RESET *   Operation
--  ==========================================================================
--      1           1           1             1    D14 Master Read, SRAM Read
--      1           1           0             1    D14 Master Write, SRAM Write
--      0           1           1             1    D14 Slave Read, SRAM Read
--      0           1           0             1    D14 Slave Write, SRAM Write
--      x           x           x             0    D14 DMA Reset
--      x           0           x             1    D14 DMA STOP
-- ***************************************************************************

               ISP1581_MS:        out STD_LOGIC;             -- debug pin

               DBG_DMA_START:     out STD_LOGIC;             -- debug pin
               DBG_DMA_RDWR:      out STD_LOGIC;             -- debug pin
               DBG_DMA_RESET:     out STD_LOGIC;             -- debug pin

              EOT:                OUT STD_LOGIC;

--            DMA bus and control signal

               DIOW:              inout STD_LOGIC;           -- dma control pin
               DIOR:              inout STD_LOGIC;           --     ""
               DACK:              inout STD_LOGIC;           --     ""
               DREQ:              inout STD_LOGIC;           --     ""

--            SRAM and flash control signal and bus
               SRAM_ADDR:         out STD_LOGIC_VECTOR(15 downto 0);
               SRAM_WR:           out STD_LOGIC;
               SRAM_RD:           out STD_LOGIC;
               SRAM_UB:           out STD_LOGIC;
               SRAM_LB:           out STD_LOGIC;
               SRAM_DATA_MSB:     inout STD_LOGIC_VECTOR(7 downto 0);
              SRAM_DATA_LSB:      inout STD_LOGIC_VECTOR(7 downto 0);
               UC_READ:           in  STD_LOGIC

    );
end toplevel;
```

```
architecture rtl of toplevel is

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

signal ms:                       STD_LOGIC;
signal DMA_START:                STD_LOGIC;
signal DMA_RDWR:                 STD_LOGIC;
signal DMA_RESET:                STD_LOGIC;
signal update_dma_sig:           STD_LOGIC;
signal dma_status:               STD_LOGIC_VECTOR(5 downto 0);

signal PIO:                      STD_LOGIC;

signal MS_z:                     STD_LOGIC;
signal MS_zz:                    STD_LOGIC;
signal DMA_START_z:              STD_LOGIC;
signal DMA_START_zz:             STD_LOGIC;
signal DMA_RDWR_z:               STD_LOGIC;
signal DMA_RDWR_zz:              STD_LOGIC;
signal DMA_RESET_z:              STD_LOGIC;
signal DMA_RESET_zz:             STD_LOGIC;
signal DMA_RESET_zzz:            STD_LOGIC;

signal dma_set:                  std_logic;
signal mode:                     std_logic_vector(3 downto 0);
signal mode_z:                   std_logic_vector(3 downto 0);
signal mode_zz:                  std_logic_vector(3 downto 0);
signal transfer_counter:         std_logic_vector(15 downto 0);
signal transfer_set_lsb:         std_logic;
signal transfer_set_msb:         std_logic;


signal SRAM_ADDR_COUNTER:        integer range 0 to 1048575;
signal MSB_WR_FLAG,LSB_WR_FLAG:  STD_LOGIC;
signal COUNTER_FLAG:             STD_LOGIC;
signal MSB_RD_FLAG,LSB_RD_FLAG:  STD_LOGIC;
signal sram_wr_a1:               std_logic;
signal sram_wr_a2:               std_logic;
signal sram_wr_a3:               std_logic;
signal reset_address_flag:       std_logic;
signal sram_rd_a1:               std_logic;
signal sram_rd_a2:               std_logic;
signal sram_rd_a3:               std_logic;
signal sram_rd_a4:               std_logic;
signal sram_rd_a5:               std_logic;

type op_code is (S0,S1,S2,S3);
signal STATE:                    op_code;
signal ADDR_COUNTER:             integer range 0 to 1048575;
signal m_ADDR_COUNTER:           integer range 0 to 1048575;
signal DACK_IN:                  std_logic;
signal DREQ_IN:                  STD_LOGIC;
signal DREQ_z:                   STD_LOGIC;
signal DREQ_zz:                  STD_LOGIC;

signal DREQ_OUT,INCREMENT:       STD_LOGIC;
signal COUNTER:                  STD_LOGIC_VECTOR(15 downto 0);
signal dma_start_enable:         STD_LOGIC;
signal sram_a1:                  std_logic;
signal sram_a2:                  std_logic;
signal sram_a3:                  std_logic;
signal sram_a4:                  std_logic;
signal sram_a5:                  std_logic;
signal sram_a6:                  std_logic;
signal sram_a7:                  std_logic;
signal DIOR_IN:                  STD_LOGIC;
signal DIOR_z, DIOR_zz:          STD_LOGIC;
signal DIOW_z, DIOW_zz:          STD_LOGIC;
signal DACK_z, DACK_zz:          STD_LOGIC;
signal DIOW_IN:                  STD_LOGIC;
```

```
signal dack_out:                std_logic;
signal PIO_reset:               std_logic;

signal read_flag:               std_logic;
signal write_flag:              std_logic;


begin

       EOT <= deasserted ;
       DREQ_z <= DREQ;

       PIO_reset <= DMA_STATUS(5);
       PIO <= DMA_STATUS(4);
       MS_z <= DMA_STATUS(3);
       DMA_START_z  <= DMA_STATUS(2);
       DMA_RDWR_z  <= DMA_STATUS(1);
       DMA_RESET_z <= DMA_STATUS(0);


       DBG_CLK      <= CLK;                    -- debug output
       isp1581_ms <= ms;                       -- debug output
       DBG_DMA_RESET <= DMA_RESET;             -- debug output
       DBG_DMA_START <= DMA_START;             -- debug output
       DBG_DMA_RDWR  <= DMA_RDWR;              -- debug output
       MODE_OUT <= MODE(2 downto 1);           -- debug output
       WIDTH <= MODE(3);                       -- debug output


ALE_Process:
-- address decoding flag set in this process
process(CS, ALE)
begin

       if CS = deasserted  then
              if falling_edge(ALE) then

                     case uC_AD is

                     when "10010101" =>               -- 95 dma signal
                            update_dma_sig <= asserted;
                            DMA_SET <= deasserted;
                            TRANSFER_SET_LSB <= deasserted;
                            TRANSFER_SET_MSB <= deasserted;
                            LSB_WR_FLAG <= deasserted;
                            MSB_WR_FLAG <= deasserted;
                            LSB_RD_FLAG <= deasserted;
                            MSB_RD_FLAG <= deasserted;
                     when "11110010" =>               -- F2 dma mode
                            DMA_SET <= asserted;
                            update_dma_sig <= deasserted;
                            TRANSFER_SET_LSB <= deasserted;
                            TRANSFER_SET_MSB <= deasserted;
                            LSB_WR_FLAG <= deasserted;
                            MSB_WR_FLAG <= deasserted;
                            LSB_RD_FLAG <= deasserted;
                            MSB_RD_FLAG <= deasserted;
                     when "11110000"=>                -- F0 LSB of transfer count
                            TRANSFER_SET_LSB <= asserted;
                            TRANSFER_SET_MSB <= deasserted;
                            update_dma_sig <= deasserted;
                            DMA_SET <= deasserted;
                            LSB_WR_FLAG <= deasserted;
                            MSB_WR_FLAG <= deasserted;
                            LSB_RD_FLAG <= deasserted;
                            MSB_RD_FLAG <= deasserted;
                     when "11110001" =>               -- F1 MSB of transfer count
                            TRANSFER_SET_MSB <= asserted;
                            TRANSFER_SET_LSB <= deasserted;
```

```
                        update_dma_sig <= deasserted;
                        DMA_SET <= deasserted;
                        LSB_WR_FLAG <= deasserted;
                        MSB_WR_FLAG <= deasserted;
                        LSB_RD_FLAG <= deasserted;
                        MSB_RD_FLAG <= deasserted;
                    when "11110100"=>                -- F4 LSB of PIO WR
                        LSB_WR_FLAG <= asserted;
                        MSB_WR_FLAG <= deasserted;
                        LSB_RD_FLAG <= deasserted;
                        MSB_RD_FLAG <= deasserted;
                        update_dma_sig <= deasserted;
                        DMA_SET <= deasserted;
                        TRANSFER_SET_LSB <= deasserted;
                        TRANSFER_SET_MSB <= deasserted;
                    when "11110101"=>                -- F5 MSB of PIO WR
                        LSB_WR_FLAG <= deasserted;
                        MSB_WR_FLAG <= asserted;
                        LSB_RD_FLAG <= deasserted;
                        MSB_RD_FLAG <= deasserted;
                        update_dma_sig <= deasserted;
                        DMA_SET <= deasserted;
                        TRANSFER_SET_LSB <= deasserted;
                        TRANSFER_SET_MSB <= deasserted;
                    when "11110110"=>                -- F6 LSB of PIO RD
                        LSB_WR_FLAG <= deasserted;
                        MSB_WR_FLAG <= deasserted;
                        LSB_RD_FLAG <= asserted;
                        MSB_RD_FLAG <= deasserted;
                        update_dma_sig <= deasserted;
                        DMA_SET <= deasserted;
                        TRANSFER_SET_LSB <= deasserted;
                        TRANSFER_SET_MSB <= deasserted;
                    when "11110111"=>                -- F7 MSB of PIO RD
                        LSB_WR_FLAG <= deasserted;
                        MSB_WR_FLAG <= deasserted;
                        LSB_RD_FLAG <= deasserted;
                        MSB_RD_FLAG <= asserted;
                        update_dma_sig <= deasserted;
                        DMA_SET <= deasserted;
                        TRANSFER_SET_LSB <= deasserted;
                        TRANSFER_SET_MSB <= deasserted;

                    when others =>
--reset all variables used in this process

                        update_dma_sig <= deasserted;
                        DMA_SET <= deasserted;
                        TRANSFER_SET_LSB <= deasserted;
                        TRANSFER_SET_MSB <= deasserted;
                        LSB_WR_FLAG <= deasserted;
                        MSB_WR_FLAG <= deasserted;
                        LSB_RD_FLAG <= deasserted;
                        MSB_RD_FLAG <= deasserted;

                    end case;
                end if;

            else
                -- reset all variables used in this process
                update_dma_sig <= deasserted;
                DMA_SET <= deasserted;
                TRANSFER_SET_LSB <= deasserted;
                TRANSFER_SET_MSB <= deasserted;
                LSB_WR_FLAG <= deasserted;
                MSB_WR_FLAG <= deasserted;
                LSB_RD_FLAG <= deasserted;
                MSB_RD_FLAG <= deasserted;

            end if;
end process;
```

```
update_DMA_parameters:
--decoding transfer counter, dma mode and dma signal(DMA Reset, DMA RdWr, DMA Start, -
--Ms)

process(CS, uC_WRITE)
begin

              if CS = deasserted then


                     if rising_edge(uC_WRITE) then

                            if update_dma_sig = asserted then
                                   dma_status <= uC_AD(5 downto 0);
                            end if;
                            if DMA_SET = asserted then
                                   MODE_z <= uC_AD(3 downto 0);
                            end if;
                            if TRANSFER_SET_LSB = asserted then
                                   TRANSFER_COUNTER(7 downto 0) <= uC_AD;
                            end if;

                            if TRANSFER_SET_MSB = asserted then
                                   TRANSFER_COUNTER(15 downto 8) <= uC_AD;
                            end if;

                            if DMA_RESET = deasserted  then
                                   TRANSFER_COUNTER <= "0000000000000000";
                            end if;
                     end if;
              end if;

end process;


update_addr:
-- update sram address
process(pio, sram_addr_counter, addr_counter, m_addr_counter, ms)
begin
       if pio = asserted then
              SRAM_ADDR <= CONV_STD_LOGIC_VECTOR(SRAM_ADDR_COUNTER,16);

       elsif ms = asserted then
              SRAM_ADDR <= CONV_STD_LOGIC_VECTOR(m_ADDR_COUNTER,16);

       else
              SRAM_ADDR <= CONV_STD_LOGIC_VECTOR(ADDR_COUNTER,16);

       end if;

end process;


Read_Process:
-- SRAM read and write
Process (pio, CS, MSB_RD_FLAG, LSB_RD_FLAG, uC_WRITE, MSB_WR_FLAG, LSB_WR_FLAG, clk,
ale, sram_a1)
variable DELAY: STD_LOGIC;
begin

              if falling_edge(CLK) then
                     if pio = asserted and CS = deasserted then
                     if MSB_WR_FLAG=asserted then

                            SRAM_RD <= asserted;
                            SRAM_LB <= asserted;
                            sram_wr_a1 <= uC_WRITE;
                            sram_wr_a2 <= sram_wr_a1;
                            sram_wr_a3 <= sram_wr_a2;
                            SRAM_UB <= (sram_wr_a3 OR (sram_wr_a2));
```

```
                SRAM_WR <= (sram_wr_a3 OR (sram_wr_a2));

        elsif LSB_WR_FLAG=asserted then

                SRAM_RD <= asserted;
                SRAM_UB <= asserted;
                sram_wr_a1 <= uC_WRITE;
                sram_wr_a2 <= sram_wr_a1;
                sram_wr_a3 <= sram_wr_a2;
                SRAM_LB <= (sram_wr_a3 OR (sram_wr_a2));
                SRAM_WR <= (sram_wr_a3 OR (sram_wr_a2));

        elsif MSB_RD_FLAG = asserted then
                SRAM_WR <= asserted;
                sram_rd_a1 <= uC_READ;
                sram_rd_a2 <= sram_rd_a1;
                sram_rd_a3 <= sram_rd_a2;
                sram_rd_a4 <= sram_rd_a3;
                sram_rd_a5 <= sram_rd_a4;
                SRAM_LB <= asserted;
                SRAM_UB <= (sram_rd_a4 OR (not(sram_rd_a5)));
                SRAM_RD <= (sram_rd_a4 OR (not(sram_rd_a5)));

        elsif LSB_RD_FLAG=asserted then
                SRAM_WR <= asserted;
                sram_rd_a1 <= uC_READ;
                sram_rd_a2 <= sram_rd_a1;
                sram_rd_a3 <= sram_rd_a2;
                sram_rd_a4 <= sram_rd_a3;
                sram_rd_a5 <= sram_rd_a4;
                SRAM_LB <= (sram_rd_a4 OR (not (sram_rd_a5)));
                sram_rd <= (sram_rd_a4 OR (not(sram_rd_a5)));
                SRAM_UB <= asserted;

        else
                SRAM_LB <= asserted;
                SRAM_UB <= asserted;
                SRAM_WR <= asserted;
                SRAM_RD <= asserted;
                sram_rd_a1 <= uC_READ;
                sram_rd_a2 <= sram_rd_a1;
                sram_rd_a3 <= sram_rd_a2;
                sram_rd_a4 <= sram_rd_a3;
                sram_rd_a5 <= sram_rd_a4;

        end if;

        end if;

if ms = deasserted and pio = deasserted then
        SRAM_LB <= deasserted;
        SRAM_UB <= deasserted;

        if DMA_RESET = deasserted then
                SRAM_WR <= asserted;
                SRAM_RD <= asserted;
                ADDR_COUNTER <= 0;
                DELAY := asserted;
        end if;

        if DMA_RDWR = WRITE then

                case STATE is

                when S2 =>

                        if DREQ_IN = asserted then
                                SRAM_RD <= deasserted;
                        else
                                SRAM_RD <= asserted;
                        end if;
```

```
                when S3 =>

                        SRAM_RD <= asserted;

                when S0 =>

                        if DACK_out = deasserted then
                                ADDR_COUNTER <= ADDR_COUNTER + 1;
                        else
                                ADDR_COUNTER <= ADDR_COUNTER;
                        end if;

                when others => NULL;

                end case;

        else

                case STATE is

                when S0 =>

                        if DACK_out = deasserted then
                                SRAM_WR <= deasserted;
                        else
                                SRAM_WR <= asserted;
                                DELAY := asserted;

                        end if;

                when S1 =>

                        SRAM_WR <= asserted;

                when S2 =>

                        if DELAY = deasserted then
                                ADDR_COUNTER <= ADDR_COUNTER + 1;
                        else
                                ADDR_COUNTER <= ADDR_COUNTER;
                                DELAY := deasserted;
                        end if;

                when others => NULL;

                end case;
        end if;


end if;

if ms = asserted and pio = deasserted then
        SRAM_LB <= deasserted;
        SRAM_UB <= deasserted;
        if DMA_RESET = deasserted  then

                SRAM_RD <= asserted;
                SRAM_WR <= asserted;
                INCREMENT <= asserted;

        else

                case MODE (2 downto 1) is

                when "00" =>


                        if DMA_RDWR = WRITE then
                                if diow_in = deasserted then
                                        increment <= deasserted;
```

```
                else
                        increment <= asserted;
                end if;

                sram_a1 <= diow_in;
                sram_a2 <= sram_a1;
                sram_a3 <= sram_a2;
                sram_a4 <= sram_a3;
                sram_a5 <= sram_a4;
                sram_a6 <= sram_a5;
                sram_a7 <= sram_a6;
                sram_wr <= ((NOT sram_a7) OR sram_a6);


        else
                if dior_in = deasserted then
                        increment <= deasserted;
                else
                        increment <= asserted;
                end if;
                sram_a1 <= dior_in;
                sram_a2 <= sram_a1;
                sram_a3 <= sram_a2;
                sram_a4 <= sram_a3;
                sram_a5 <= sram_a4;
                sram_a6 <= sram_a5;
                sram_a7 <= sram_a6;
                sram_rd <= ((NOT sram_a7) OR sram_a6);


        end if;


when "01" =>

        if DMA_RDWR = WRITE then
                if dack_in = deasserted then
                        increment <= deasserted;
                else
                        increment <= asserted;
                end if;

                sram_a1 <= dack_in;
                sram_a2 <= sram_a1;
                sram_a3 <= sram_a2;
                sram_a4 <= sram_a3;
                sram_a5 <= sram_a4;
                sram_a6 <= sram_a5;
                sram_a7 <= sram_a6;
                sram_wr <= ((NOT sram_a7) OR sram_a6);


        else

                if dior_in = deasserted then
                        increment <= deasserted;
                else
                        increment <= asserted;
                end if;


                sram_a1 <= dior_in;
                sram_a2 <= sram_a1;
                sram_a3 <= sram_a2;
                sram_a4 <= sram_a3;
                sram_a5 <= sram_a4;
                sram_a6 <= sram_a5;
                sram_a7 <= sram_a6;
                sram_rd <= ((NOT sram_a7) OR sram_a6);

        end if;
```

```
                        when "10" =>

                                if DMA_RDWR = WRITE then

                                        sram_a1 <= dack_in;
                                        sram_a2 <= sram_a1;
                                        sram_a3 <= sram_a2;
                                        sram_a4 <= sram_a3;
                                        sram_a5 <= sram_a4;
                                        sram_a6 <= sram_a5;
                                        sram_a7 <= sram_a6;
                                        sram_wr <= ((NOT sram_a7) OR sram_a6);


                                else
                                        sram_a1 <= dack_in;
                                        sram_a2 <= sram_a1;
                                        sram_a3 <= sram_a2;
                                        sram_a4 <= sram_a3;
                                        sram_a5 <= sram_a4;
                                        sram_a6 <= sram_a5;
                                        sram_a7 <= sram_a6;
                                        sram_rd <= ((NOT sram_a7) OR sram_a6);

                                end if;

                                if dack_in = deasserted then
                                        increment <= deasserted;
                                else
                                        increment <= asserted;
                                end if;


                        when others => NULL;

                        end case;
                end if;
        end if;

        if DMA_RESET = deasserted then

                ADDR_COUNTER <= 0;
                DELAY := asserted;
                INCREMENT <= asserted;
        end if;

    end if;

end process;


WRITE_READ_Process:
-- direction control of AD to read and write mode

process(pio, CS, MSB_WR_FLAG, LSB_WR_FLAG, SRAM_DATA_MSB, SRAM_DATA_LSB, ALE, uC_AD,
    MSB_RD_FLAG, LSB_RD_FLAG, uC_READ, CLK)
begin
    if pio = deasserted then
        uC_AD <= "ZZZZZZZZ";
        SRAM_DATA_MSB <= "ZZZZZZZZ";
        SRAM_DATA_LSB <= "ZZZZZZZZ";

    else
        if CS = deasserted  then

            if falling_edge(CLK) then

                if MSB_WR_FLAG=asserted and uC_WRITE = deasserted then
                    SRAM_DATA_MSB<= uC_AD;
                    SRAM_DATA_LSB<="ZZZZZZZZ";
```

```
                                uC_AD <= "ZZZZZZZZ";
                            elsif LSB_WR_FLAG=asserted and uC_WRITE = deasserted then
                                SRAM_DATA_LSB<=uC_AD;
                                SRAM_DATA_MSB<="ZZZZZZZZ";
                                uC_AD <= "ZZZZZZZZ";
                            elsif MSB_RD_FLAG=asserted and uC_READ=deasserted then
                                uC_AD <= sram_DATA_MSB;
                                SRAM_DATA_MSB <= "ZZZZZZZZ";
                                SRAM_DATA_LSB <= "ZZZZZZZZ";
                            elsif LSB_RD_FLAG=asserted and uC_READ=deasserted then
                                uC_AD <= sram_DATA_LSB;
                                SRAM_DATA_LSB <= "ZZZZZZZZ";
                                SRAM_DATA_MSB <= "ZZZZZZZZ";
                            else
                                uC_AD <= "ZZZZZZZZ";
                                SRAM_DATA_MSB <= "ZZZZZZZZ";
                                SRAM_DATA_LSB <= "ZZZZZZZZ";
                            end if;

                        end if;
                end if;

        end if;

end process;


ADDRESS_PROCESS:
--PIO address logic and address reset logic
--the address is reset whenever there is a change from read to write or vice versa

process(pio, CS, ALE, uC_AD, uC_WRITE, reset_address_flag, sram_addr_counter,
counter_flag,
        uC_READ,read_flag,write_flag)

begin
        if pio = asserted then
                if CS = deasserted  then
                        if falling_edge(ALE) then

                                case uC_AD is


                                when "11110100"=>                       -- F4 LSB WR
                                        if COUNTER_FLAG = deasserted then
                                                SRAM_ADDR_COUNTER <= SRAM_ADDR_COUNTER;
                                                COUNTER_FLAG<=deasserted;

                                        else
                                                SRAM_ADDR_COUNTER <= SRAM_ADDR_COUNTER + 1;
                                                COUNTER_FLAG<=deasserted;
                                        end if;


                                when "11110110"=>                       -- F6 LSB RD
                                        if COUNTER_FLAG = deasserted then
                                                SRAM_ADDR_COUNTER <= SRAM_ADDR_COUNTER;
                                                COUNTER_FLAG<=deasserted;

                                        else
                                                SRAM_ADDR_COUNTER <= SRAM_ADDR_COUNTER + 1;
                                                COUNTER_FLAG<=deasserted;
                                        end if;


                                when "11110101"=>                       -- F5 MSB WR
                                        COUNTER_FLAG<=asserted;

                                when "11110111"=>                       -- F7 MSB RD
                                        COUNTER_FLAG<=asserted;
```

```
                              when others =>
                                    SRAM_ADDR_COUNTER <= SRAM_ADDR_COUNTER;

                              end case;

                        end if;


                        if uC_READ = deasserted then
                              read_flag <= deasserted;
                        end if;

                        if uC_WRITE = deasserted then
                              write_flag <= deasserted;
                        end if;

                        if read_flag = deasserted and uC_WRITE = deasserted then
                              -- reset address
                              SRAM_ADDR_COUNTER<=0;
                              COUNTER_FLAG<=deasserted;
                              read_flag <= asserted;
                        elsif write_flag = deasserted and uC_READ = deasserted then
                              -- reset address
                              SRAM_ADDR_COUNTER<=0;
                              COUNTER_FLAG<=deasserted;
                              write_flag <= asserted;
                        end if;
                  end if;
            end if;
      end process;

      STATE_MACHINE_Process:

      process(DMA_RESET, state, clk, ms, pio)
      begin
            if ms = deasserted then
                  if DMA_RESET = deasserted then
                        STATE <= S0;
                        DBG_STATE <= "00";

                  elsif falling_edge(CLK) then

                         if ((DREQ_IN = asserted) or
                           (DREQ_IN = deasserted and DACK_out = deasserted) or
                           (STATE /= S0)) then

                              case   STATE is

                              when S0 =>
                                    STATE <= S1;
                                    DBG_STATE <= "01";
                              when S1 =>
                                    STATE <= S2;
                                    DBG_STATE <= "10";
                              when S2 =>
                                    STATE <= S3;
                                    DBG_STATE <= "11";
                              when S3 =>
                                    STATE <= S0;
                                    DBG_STATE <= "00";
                              when others =>
                                    STATE <= S0;
                                    DBG_STATE <= "00";

                              end case;
                        else
                              STATE <= S0;
                        end if;
                  end if;
            end if;
      end process;
```

```
DACK_Process:
-- cpld master dack logic
process(DMA_RESET, CLK, ms, pio)
begin
       if ms = deasserted  and pio = deasserted then
              if DMA_RESET = deasserted then

                     DACK_OUT <= asserted;
                     DACK <= asserted;

              elsif falling_edge(CLK) then

                     case MODE(2 downto 1) is

                     when "00" =>

                            if STATE = S2 then
                                   DACK_out <= not DREQ_IN;
                            else
                                   DACK_out <= DACK_out;
                            end if;

                     when others =>

                            if DMA_RDWR = READ and MODE(2 downto 1) = "01" then

                                   if STATE = S2 then
                                          DACK_out <= not DREQ_IN;
                                   else
                                          DACK_out <= DACK_out;
                                   end if;

                            else

                                   case STATE is

                                   when S3 =>

                                          if DREQ_IN = asserted then
                                                 DACK_out <= deasserted;
                                          else
                                                 DACK_out <= asserted;
                                          end if;

                                   when S1 =>

                                          if DREQ_IN = asserted then

                                                 DACK_out <= asserted;

                                          elsif DREQ_IN = deasserted then

                                                 DACK_out <= deasserted;

                                          end if;

                                   when S2 =>

                                          if DREQ_IN = deasserted then

                                                 DACK_out <= asserted;

                                          else
                                                 DACK_out <= DACK_out;

                                          end if;

                                   when others => NULL;
```

```
                                end case;
                        end if;
                end case;
                -- delay the output by one clock
                DACK <= DACK_out;
        end if;

    else
        dack <= 'Z';

    end if;
end process;



DIOR_DIOW_Process:
-- cpld master diow and dior logic

process(DMA_RESET, CLK, ms, pio)
begin
    if ms = deasserted and pio = deasserted then

        if DMA_RESET = deasserted then

            DIOR <= asserted;
            DIOW <= asserted;

        elsif falling_edge(CLK) then

            if DACK_out = deasserted then

            case MODE(2 downto 1) 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 S1 =>

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

                when S0 =>

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

                when S2 =>

                    if DMA_RDWR = WRITE then
                        DIOW <= asserted;
```

```
                                        DIOR <= asserted;
                                  else
                                        DIOW <= asserted;
                                        DIOR <= asserted;
                                  end if;

                            when others => NULL;

                            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 S1 =>

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

                            when S0 =>

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

                            when S2 =>

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

                            when others => NULL;

                            end case;

                      when others => NULL;

                      end case;

                      end if;

                end if;

          else
                dior <= 'Z';
                diow <= 'Z';

          end if;
    end process;
```

```
SRAM_ADDR_Process:
-- master mode address logic

process(DMA_RESET, INCREMENT, ms, pio)
begin
       if ms = asserted and pio = deasserted then
              if DMA_RESET = deasserted then

                     m_ADDR_COUNTER <= 0;

              elsif rising_edge(INCREMENT) then
                     if dma_start = asserted then
                            m_ADDR_COUNTER <= m_ADDR_COUNTER + 1;
                     end if;

              end if;
       end if;
end process;


DMA_COUNTER:
-- master mode counter
process(DMA_RESET, dreq_out, INCREMENT, ms, pio)
begin

       if ms = asserted and pio = deasserted then
              if DMA_RESET = deasserted or DREQ_OUT = deasserted  then
                     COUNTER <= "0000000000000000";

              elsif falling_edge(INCREMENT) then
                     if MODE(3) = asserted then
                            COUNTER <= COUNTER + 2;
                     else
                            COUNTER <= COUNTER + 1;
                     end if;
              end if;
       end if;

end process;

DREQ_Process:
-- master mode dreq logic

process(DMA_RESET, MS, CLK, pio)
begin
       if ms = asserted and pio = deasserted then
              if DMA_RESET = deasserted  then

                     DREQ <= deasserted;
                     DREQ_OUT <= deasserted;
                     dma_start_enable <= asserted;


              elsif falling_edge(CLK)

                     if ((COUNTER(15 downto 0) >= TRANSFER_COUNTER(15 downto 0)) and
                         COUNTER(15 downto 0) /= "0000000000000000") then

                            DREQ <= deasserted;
                            DREQ_OUT <= deasserted;
                            dma_start_enable <= deasserted;
                     elsif DMA_START = asserted and dma_start_enable = asserted then

                            DREQ <= asserted;
                            DREQ_OUT <= asserted;
                     elsif DMA_START = deasserted then
                            dma_start_enable <= asserted;

                     end if;
```

```
                end if;
        else

                dreq <= 'Z';
                DREQ_OUT <= deasserted;
                dma_start_enable <= asserted;

        end if;
end process;

--------              synchronising the inputs                --------

sync_process1:
process(CLK, DMA_RESET_z)
begin
        if DMA_RESET_z = deasserted then
                DMA_RESET_zz <= deasserted;
                DMA_RESET_zzz <= deasserted;
        elsif falling_edge(CLK) then

                MS_zz <= MS_z;
                MS <= MS_zz;

                DMA_START_zz  <= DMA_START_z;
                DMA_START  <= DMA_START_zz;

                DMA_RDWR_zz  <= DMA_RDWR_z ;
                DMA_RDWR  <= DMA_RDWR_zz ;

                DMA_RESET_zz <= DMA_RESET_z ;
                DMA_RESET_zzz <= DMA_RESET_zz ;

        end if;
end process;

        -- sync dma_reset
        DMA_RESET <= DMA_RESET_zzz and DMA_RESET_z;


sync_PROCESS2:
process(CLK, DMA_RESET)
begin
        if DMA_RESET = deasserted then
                DIOR_z <= deasserted;
                DIOR_zz <= deasserted;
                DIOW_z <= deasserted;
                DIOW_zz <= deasserted;
                DACK_z <= deasserted;
                DACK_zz <= deasserted;
                DREQ_zz <= deasserted;
                DREQ_IN <= deasserted;
        elsif falling_edge(CLK) then
                MODE_zz <= MODE_z;
                MODE <= MODE_zz;
                DIOR_z <= DIOR;
                DIOR_zz <= DIOR_z;
                DIOW_z <= DIOW;
                DIOW_zz <= DIOW_z;
                DACK_z <= DACK;
                DACK_zz <= DACK_z;
                DREQ_zz <= DREQ_z;
                DREQ_IN <= DREQ_zz;
        end if;
end process;

        DIOR_IN <= DIOR AND DIOR_zz;
        DIOW_IN <= DIOW AND DIOW_zz;
        DACK_IN <= DACK AND DACK_zz;

end rtl;
```

12. References

- *ISP1583 Hi-Speed Universal Serial Bus interface device* datasheet
- *ISP1582/83 Software Guide*

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

PHILIPS