XILINX®

Configuring FPGAs Over a Processor Bus

September 8, 1995

Application Note BY GARY LAWMAN

Summary

This Application Note describes how to configure an FPGA over a processor bus. It also illustrates the source code required to download a configuration bitstream using an IBM PC as a host microprocessor.

Xilinx Family	*Demonstrates*
XC5200 and XC4000 FPGAs	Device configuration and associated data formats.
XC3100A and XC3000A FPGAs under specified circumstances	Device operation during power-on and reset cycles.
Not available for XC2000 and XC3000 FPGAs	Minimum decode circuitry using a single discrete flip-flop.
	An algorithm for ultra-fast reconfiguration schemes.

Table of Contents

Warning

This application note only applies to the families listed above, the configuration circuits in other device families such as ***XC3000*** and ***XC2000*** parts are **not compatible** with the techniques demonstrated.

Introduction

The direct link between a microprocessor and an FPGA enhances debugging and testing, and reduces the product's time to market. In-system device configuration, (also known as **I**n-**S**ystem **P**rogramming, or **ISP**), significantly reduces the development cycle of new products incorporating FPGAs. Once the design is complete, the configuration data can be stored on disk or in ROM and then loaded each time the system powers up or is Reset. The configuration cycle can be transparent to the end user. The interface includes both software and hardware elements, and a program to configure or download the FPGA program data.

Functional Description

The basic operation of all configuration schemes is the same and is composed of the following stages:

1. The device configuration memory is cleared.
2. Configuration data is clocked into the device.
3. After a successful download, the FPGA asserts its DONE signal.

There are a number of different configuration modes available to the designer, each of which require a small number of control lines. The Slave Serial mode is perhaps the simplest requiring only three active signals:

1. **DIN**, the data in pin, also labeled **D0**
2. Configuration Clock, **CCLK**
3. **PROGRAM**

A small number of additional lines must be tied either High or Low. For further details, please refer to the later discussion on configuration signals and the example schematics.

The data itself is broken down into several key blocks as shown in Figure 1:

- The header data indicates the size of the download data, followed by
- multiple frames containing both error check bits and
- the actual circuit configuration data.

```
Xilinx LCA tstpcbd.lca 4003PC84
File tst_isa.rbt
Fri Mar 24 18:05:07 1995
Fri Mar 24 18:05:07 1995
Source
Version
Produced by makebits version 5.1.0
1111111100100000000011010010110101101111
0101111111111110111111101011111110101111
1110101111111010111111101101111111010111
1111010111111101011111110101111111011111
1111010111111111111101111111110111111111
0111111111011111111101111111111011111111
1111010111111111111101111111110111111111
0111111111011111111101111111111011111111
1111010111111101011111110101111111011111
1111010111111111111101111111110111111111
0111111111011111111101111111111011111111
1011111111101111111110111111111011111111
111111110101      .
                  .
                  .
1011111111101111111110111111111011111111
1111010111111111111101111111110111111111
0111111111011111111101111111111011111111
01111111
```

Figure 1. Example Rawbits data file.

There are several features worthy of note in the Rawbits data file and these are highlighted by the bold fonts in Figure 1. The text header, provides a software program audit trail. The actual data frame starts with

- eight dummy bits, (a minimum of eight),
- a pre-amble code of '0010,'
- a 24-bit length count, and finally
- at least 4 more dummy bits to close the preamble before the actual data starts.

Each data frame is completed with four error bits which are normally used to denote the CRC value for the data frame. The number of bits in a frame and number of frames is device dependent. The file is closed with a final post-amble code of eight bits as illustrated in the example. For further details the user should refer to page 2-26 in **The Programmable Logic Handbook** (1994).

Microprocessor Interface

Typically, designers implement the microprocessor-to-FPGA configuration interface using a PAL or numerous discrete logic devices. However, due to the simplicity of the serial data interface, this is unnecessary and a minimum decode circuit can be built using a single discrete flip-flop. The board level circuit is shown in Figure 2. Not shown in Figure 2 are the additional pages which contain the data bus interface, see the VIEWlogic files for details. A downloadable test circuit is shown in Figure 3. The test circuit contains the required LOCK signal and a self-clocked counter that outputs a low frequency signal for visual display using an LED or scope viewing.

The full address map decode may not be necessary during the configuration period. All that is required during the download period is guaranteed exclusive program access to the selected address space for the duration of the download period. On an IBM PC, this is achieved by disabling interrupts and DMA activity for the duration of the configuration period.

A full address map decoder may be included in the users design which is then downloaded as part of the device's configuration bitstream. See Figure 4 for a downloadable full PC I/O address map decoder. This is attractive for many reasons, not least of which is the ability to re-map the design in software according to free address space. The usual PC expansion bus configuration switches may be thus replaced with a few bits of address selection data contained in a very small FPGA based ROM. This ROM may be generated using the Xilinx MEMGEN program. Various optional files may then be supplied to support different address decodes.

Nothing prevents this same technique being applied to other processors, although the inclusion of a separately decoded I/O space of Intel iAPX86 processors provides a significant advantage in simplifying the interface.

To successfully configure an FPGA from a microprocessor bus, the FPGA's memory must cleared, error-free configuration data written into it, and the completed device configuration locked in.

The FPGA is forced to start clearing its configuration memory by asserting a Low on its PROGRAM pin. After the device has completed this initialization, it is ready for new configuration data. After data is clocked into the device, there must be no more activity on the PROGRAM pin. Otherwise, the whole sequence must be aborted and re-started.

This is achieved using a single discrete flip-flop, (FF), with independent control the flip-flop's clock, data and set/preset pins. This flip-flop can then be used to control the PROGRAM pin on the FPGA during the configuration process. Providing that one can also suitably drive the FPGA's configuration clock, (CCLK) and data in, (DIN) this is all the circuitry that is required.

In the IBM PC environment, data can be clocked both into the flip-flop and configuration data into the FPGA using the processor's IOWR, (IO Write), signal.

The flip-flop holds a data bit that drives the PROGRAM pin. Once a Low has been clocked into this flip-flop configuration can only continue if the output is held high. This is achieved when writing of configuration data by always writing a '1' to the data bit driving this register. Remember, that The Slave Serial configuration scheme only drives one data pin on the FPGA, so the two data bits for PROGRAM and DIN may be driven independently. Once the download of configuration data has been successfully completed, a user-selected pin on the FPGA drives the flip-flop's Set input. This prevents any further access to the PROGRAM pin by the microprocessor (LOCK). The circuit driving the LOCK pin must be included in the user's design downloaded into the device.

Because all purely 'user' pins, on the FPGA are tri-stated during configuration, the flip-flop's Set input would float unless suitably tied. Consequently, a resistor defines an appropriate level which may then be overridden by the LOCK bit. The LOCK pin may be driven by any desired combination of signals or registers. If this is made addressable by the microprocessor, future downloads can be initiated in software by strobing this signal. Alternatively, it could be controlled by a watchdog, or other security circuit. In the event that the user chooses to completely lock the configuration against any unauthorized changes, this pin may be driven by a fixed level within the device, so that new downloads can only occur after a system-level power cycle.

Once the device's configuration memory is initialized, all of the configuration data must be written to it. Although the device at this stage will respond to all I/O writes, it is best to address only known free I/O space to avoid spurious data writes to other I/O devices. As already stated, there must be no other I/O accesses during the download period, such as may be caused by interrupts or DMA transactions. In the PC environment, disabling interrupts is achieved using the CLI instruction, and re-enabled using the STI instruction. In embedded applications, the preferred time to download the configuration data is during the initial boot period. As with the PC users, must alternatively be able to disable interrupts and DMA accesses which would otherwise write invalid data into the FPGA's configuration memory.

Additional care and decoding will be required if this technique is used in a processor's memory space. This technique may not be compatible with configuration data held in memory decoded as part of the processor's I/O space. This could happen in the case of configuration data being stored on a PC's disks. This particular problem is readily avoided by reading the configuration data into an array in memory from which it is then accessed for the download process.

In the event that these issues can not be resolved, the memory map for the device's configuration port must be fully decoded implemented with TTL or an EPLD. See Figure 5 for an example.

XC3xxxA Application Notes

There are some differences between the XC3xxxA and XC4xxx/XC52xx configuration signals that must be noted before the technique that has been discussed can be used with parts from this family. The differences may be treated largely as a matter of the naming convention that is used for the relevant signals. In XC3xxxA there are no signals named PROGRAM or DONE. Instead, the equivalent signals are RESET and D/P- respectively.

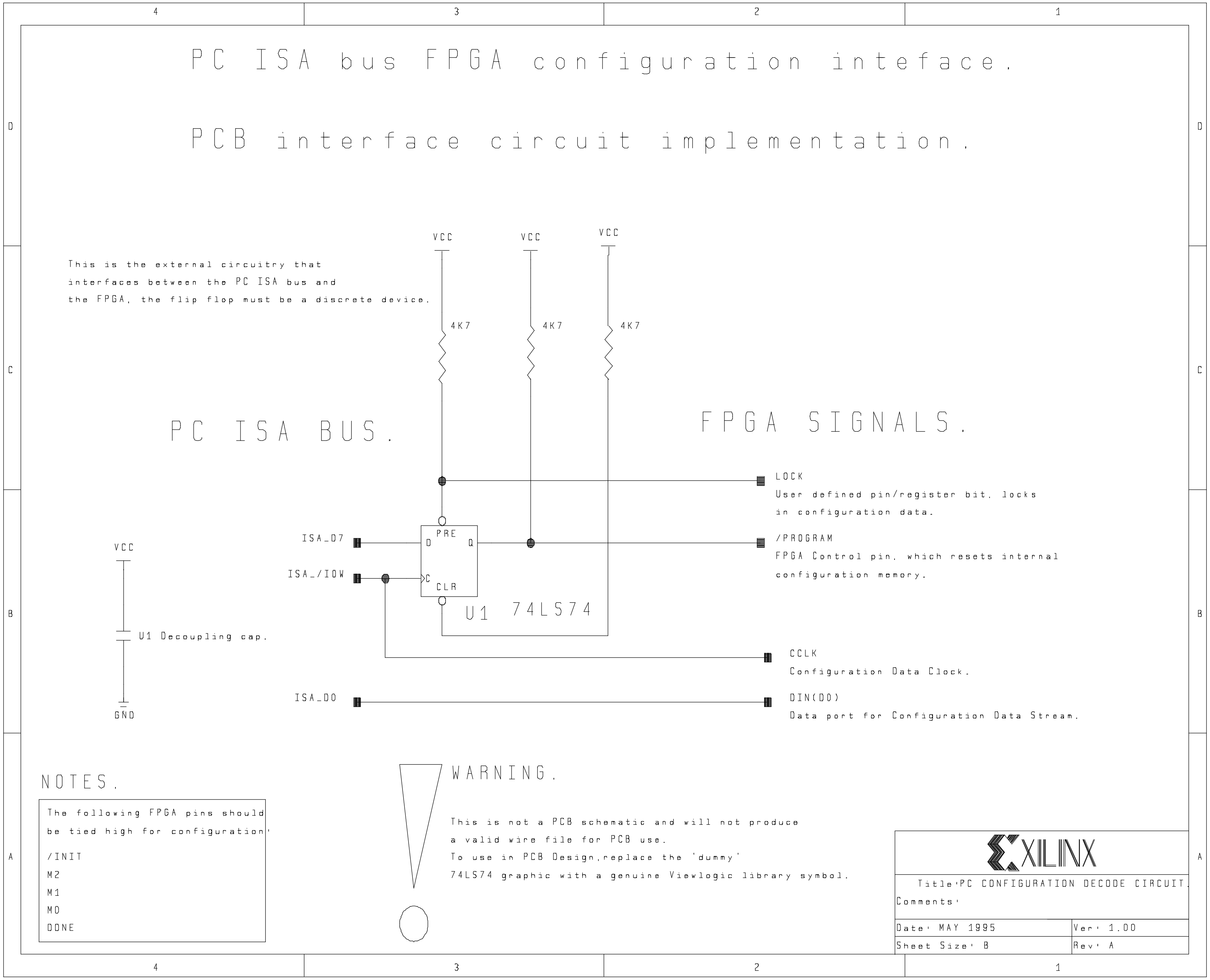

PC ISA bus FPGA configuration inteface.
PCB interface circuit implementation.
This is the external circuitry that interfaces between the PC ISA bus and the FPGA, the flip flop must be a discrete device.
VCC
VCC
VCC
4K7
4K7
4K7
PC ISA BUS.
FPGA SIGNALS.
LOCK
User defined pin/register bit, locks in configuration data.
ISA_D7
PRE
D
Q
/PROGRAM
FPGA Control pin, which resets internal configuration memory.
ISA_/IOW
C
CLR
U1 74LS74
VCC
U1 Decoupling cap.
GND
CCLK
Configuration Data Clock.
ISA_D0
DIN(D0)
Data port for Configuration Data Stream.
NOTES.
The following FPGA pins should be tied high for configuration:
/INIT
M2
M1
M0
DONE
WARNING.
This is not a PCB schematic and will not produce a valid wire file for PCB use.
To use in PCB Design, replace the 'dummy' 74LS74 graphic with a genuine Viewlogic library symbol.
XILINX
Title:PC CONFIGURATION DECODE CIRCUIT.
Comments:
Date: MAY 1995
Ver: 1.00
Sheet Size: B
Rev: A

Figure 2. Board-level schematic interface.

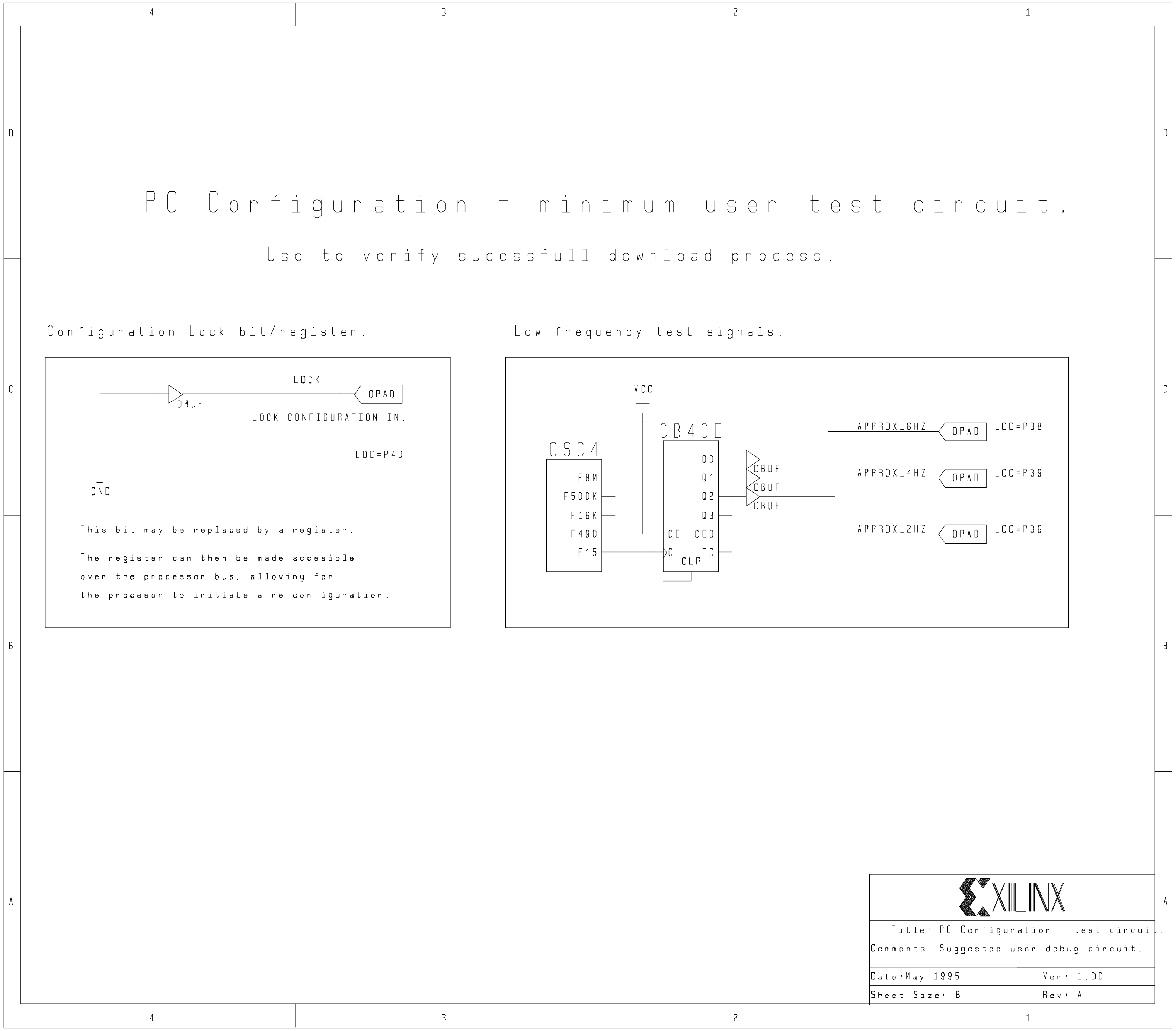

PC Configuration - minimum user test circuit.
Use to verify sucessfull download process.
Configuration Lock bit/register.
Low frequency test signals.
LOCK
OPAD
OBUF
LOCK CONFIGURATION IN.
LOC=P40
GND
This bit may be replaced by a register.
The register can then be made accesible
over the processor bus, allowing for
the procesor to initiate a re-configuration.
VCC
CB4CE
OSC4
F8M
F500K
F16K
F490
F15
Q0
Q1
Q2
Q3
CE
CEO
C
TC
CLR
OBUF
OBUF
OBUF
APPROX_8HZ
APPROX_4HZ
APPROX_2HZ
OPAD
OPAD
OPAD
LOC=P38
LOC=P39
LOC=P36
XILINX
Title: PC Configuration - test circuit.
Comments: Suggested user debug circuit.
Date:May 1995
Ver: 1.00
Sheet Size: B
Rev: A

Figure 3. Downloadable test circuit for the flip-flop decode (FPGA).

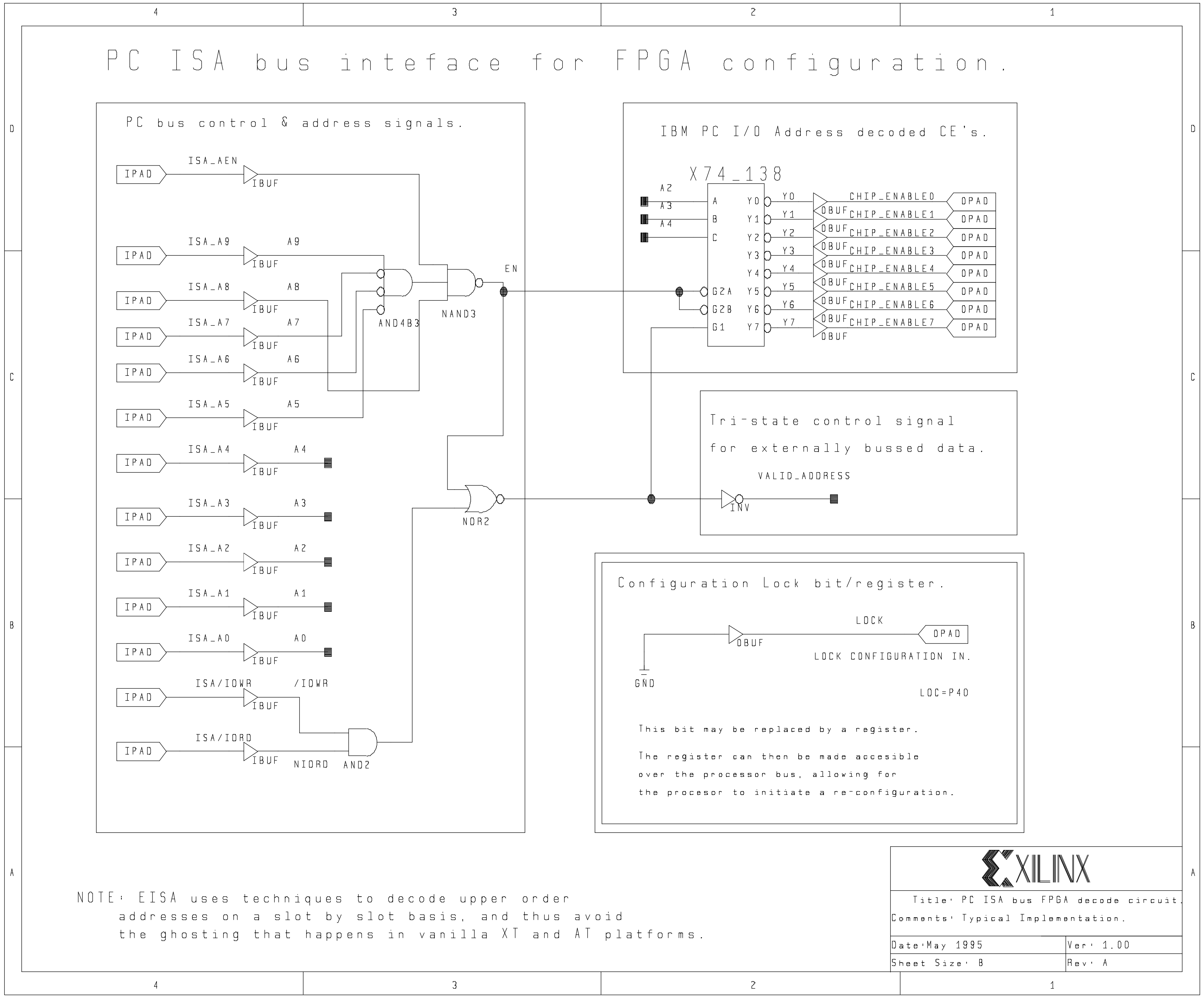
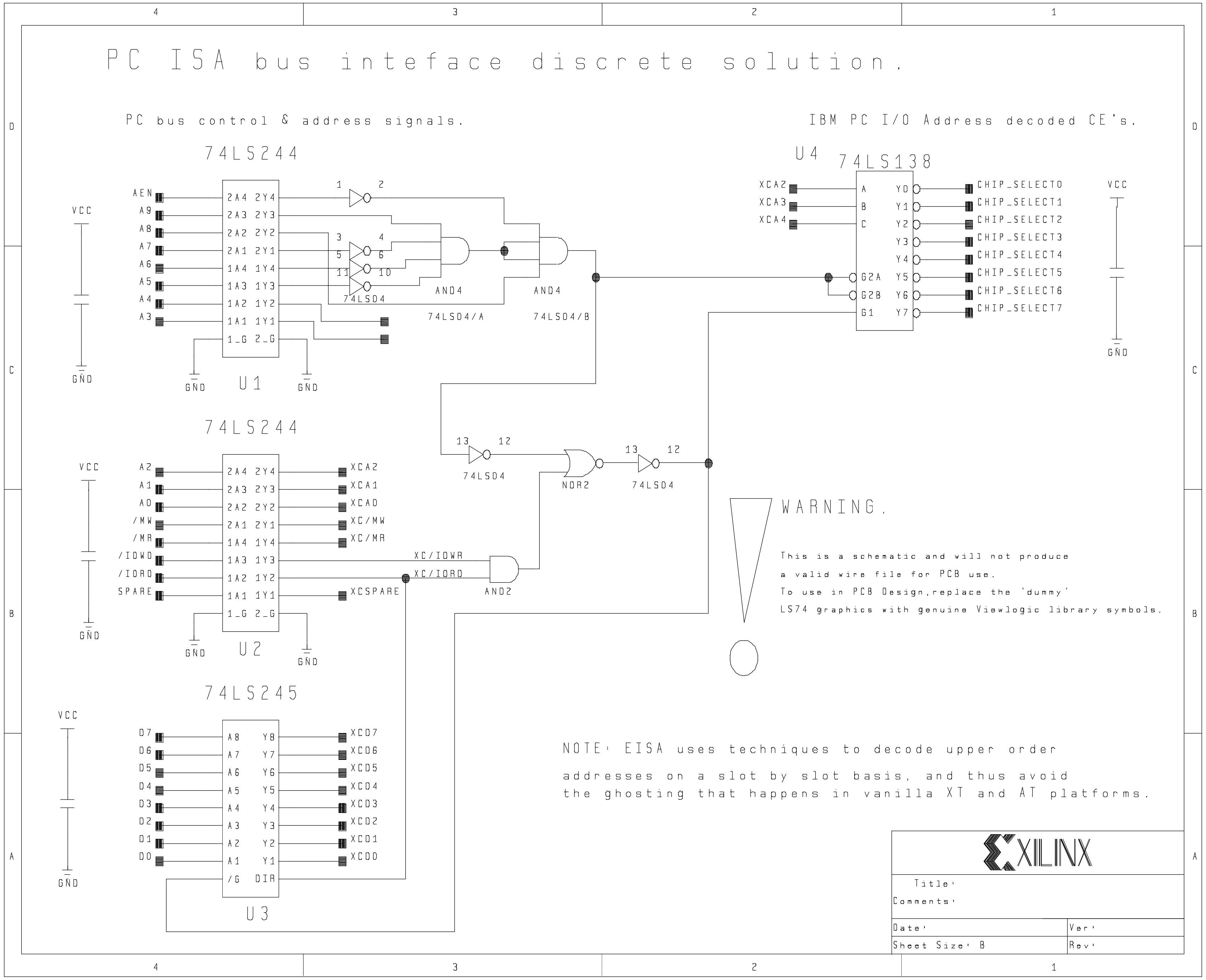

PC ISA bus inteface for FPGA configuration.
PC bus control & address signals.
IPAD
IBUF
ISA_AEN
ISA_A9
A9
ISA_A8
A8
ISA_A7
A7
ISA_A6
A6
ISA_A5
A5
ISA_A4
A4
ISA_A3
A3
ISA_A2
A2
ISA_A1
A1
ISA_A0
A0
ISA/IOWR
/IOWR
ISA/IORD
NIORD
AND2
AND4B3
NAND3
EN
NOR2
IBM PC I/O Address decoded CE's.
X74_138
A
B
C
G2A
G2B
G1
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
OBUF
CHIP_ENABLE0
CHIP_ENABLE1
CHIP_ENABLE2
CHIP_ENABLE3
CHIP_ENABLE4
CHIP_ENABLE5
CHIP_ENABLE6
CHIP_ENABLE7
OPAD
Tri-state control signal
for externally bussed data.
VALID_ADDRESS
INV
Configuration Lock bit/register.
LOCK
OPAD
LOCK CONFIGURATION IN.
LOC=P40
GND
This bit may be replaced by a register.
The register can then be made accesible
over the processor bus, allowing for
the procesor to initiate a re-configuration.
NOTE: EISA uses techniques to decode upper order
addresses on a slot by slot basis, and thus avoid
the ghosting that happens in vanilla XT and AT platforms.
XILINX
Title: PC ISA bus FPGA decode circuit.
Comments: Typical Implementation.
Date:May 1995
Ver: 1.00
Sheet Size: B
Rev: A

Figure 4. Downloadable full PC I/O address map decoder.



WARNING.

This is a schematic and will not produce a valid wire file for PCB use.
To use in PCB Design,replace the 'dummy' LS74 graphics with genuine Viewlogic library symbols.

NOTE: EISA uses techniques to decode upper order addresses on a slot by slot basis, and thus avoid the ghosting that happens in vanilla XT and AT platforms.

Figure 5. Discretely implemented full PC I/O address map decoder.

Figure 6. Configuration program listing.

```
/*************************************************************************************/
/*                                                                                  */
/*  Rev 1.0 Config.  (c) 1995 Xilinx Inc.                                           */
/                                                                                   */
/*  uP Download program, requires an ASCII bit configuration data file.             */
/*                                                                                  */
/*  Program source and executeables are not supported.                              */
/*  No guarantees are provided or implied by this program or sources                */
/*  Xilinx customers may use this program and sources for configuring               */
/*  Xilinx Parts without royalty.                                                   */
/*                                                                                  */
/*  This program must be used with the Application circuit given in the             */
/*  accompanying note.                                                              */
/*                                                                                  */
/*  WARNING THIS PROGRAM'S CONFIGURATION CIRCUIT SHOULD ONLY BE USED                */
/*  WITH THE FOLLOWING PART FAMILIES: XC31xxA, XC4xxx & A, XC52xx                   */
/*                                                                                  */
/*                                                                                  */
/* Complied with Borland C++ version 3.0, not tested with other compilers           */
/*                                                                                  */
/*************************************************************************************/

#include  <stdio.h>
#include  <io.h>
#include  <conio.h>
#include  <string.h>

#define IOPORTADDR 0x300      /* This must be an unused I/O address for    */
                                     /* the IBM PC.                                                */
                                     /* Other options are:                                         */
                                     /*                                                            */
                                     /*                                                            */
                                     /*                                                            */
                                     /*                                                            */

#define MAXSIZEBITS 55000     /* Number of device configuration data bits.         */
                                     /* Dependent upon actual device used.         */
                                     /* See the data book for details.             */
#define DEVICEFACTOR 900000   /* Waiting factor, this is a device and              */
                                     /* host machine dependent delay factor.       */
                                     /* unfortunately we can not use the PC's      */
                                     /* timers due to them being I/O mapped.       */
                                     /* See App Note for further details.          */

FILE *fp;                            /* Pointer to the ASCII config data file.    */
char dataarray[MAXSIZEBITS];  /* Memory array for the config data.                 */
char filechar;                /* ASCII bit value from config data file.            */
unsigned int addr ,addrcount; /* Misc counters.                                    */

void delay(long loopval)      /* We need to allow time for the device to   */
{                                    /* finish various operations.                 */
long j,k;
     for (j = 1; j <= loopval; ++j )
          {
          k = inportb(IOPORTADDR);

          /* Users must ensure that their compiler does not  */
          /* optimise this loop out as the value of k is     */
          /* never used.                                     */

          /* This is a loop delay - as we can not use timers */
          /* located in I/O space or interrupts, during the  */
          /* call.                                   */

          /* This has to allow for the FPGA device size and  */
          /* CPU, the only way to achieve some CPU timing    */
          /* independence for the loop period is to use      */
          /* I/O reads.  This gives the loop a fixed timing  */
          /* overhead, as the I/O is clocked at 8 MHz.       */
          /* Note however that some systems can run I/O at   */
          /* 10 or 12 MHz, through user options.             */
          }
}
```

```
void ReadConfigFileData(char filename[11])
{
    addr = 0;
    fp = fopen(filename, "r");
    printf("%s", filename);
    while (!feof(fp))
     {
         fscanf(fp, "%c", &filechar);
         dataarray[addr] = filechar;
         ++addr;
        }
    printf(" Length of File read is %u", addr-1, " bytes");
    /* Note that addr is incremented 1 too many times, hence -1 above.  */
}

void InitFPGA(long delayval)
{
    outportb(IOPORTADDR, 0x00);                      /* Initialise the config memory.      */
    outportb(IOPORTADDR, 0xff);                      /* both output's required.            */

    delay(delayval);          /* Period to allow device to do internal clear.      */
                                      /* This has to be a program loop, as interrupts are  */
                                      /* now switched off, so we can't read the timers.    */
                                      /* Delayval must be calculated for each device.      */
                                      /* Alternatively we can assume worst case delay.     */
                                      /* ie for largest device, and then add a margin.     */
}

void WriteConfigDataToFPGA(long delayvalue)
{
  addrcount = 0;
  while (addrcount != addr)
    {
       if (dataarray[addrcount] ==  '1') outportb(IOPORTADDR,0xff);
       else outportb(IOPORTADDR,0xfe); /* '0' case selected.    */
       /* We assume a valid data bit either ASCII '1' or '0'.   */
       /* A default case could be provided for error checking.  */
     addrcount++;
    }
  delay(delayvalue);          /* Allow the device time to complete the             */
                              /* cofiguration.                                     */
}


int main()
{
    printf("\n \n");
    printf(" Started Configuration Process.  \n");
    printf(" File data read, now press any key to do download.  \n ");

    while (!kbhit());
    ReadConfigFileData("bit_data.stp");              /* This file must be a stripped      */
                                                             /* version of a Makebits              */
                                                             /* generated rawbits data file.       */
    asm cli;                                         /* Disable interrupts - any I/O such */
                                                             /* as disk access                     */
                                                             /* will kill the download.            */

    InitFPGA(DEVICEFACTOR);                          /* Strobe the program line, start    */
                                                             /* device init                        */

    WriteConfigDataToFPGA(DEVICEFACTOR);      /* Write data to device.             */

    asm sti;                                  /* Re-enable interrupts, then we     */
                                                             /* can use the PC                     */
                                                             /* again, otherwise it is locked up.  */
    printf("\n Finished %d \n");
    fclose(fp);
    return 0;                                        /* Main completed OK.                         */
}
```

Program Files and Data Formats

The program supplied with this Application Note requires that the design bitstream is generated using a program called “Makebits” using the “rawbits” option. The output file is an ASCII file with each character a ‘0’ or a ‘1’. This file must be further processed to strip the header information. This can either be done by hand editing the file or using a utility program to strip the unnecessary text. The final data is then written into the FPGA during configuration, and is the input required by the example program, “config.c”.

Download Program Overview

1. Read bitstream data file into a memory based array.
    - Initialize array.
    - Read from file and count # bits*.
    - Write data to array, # bits*.
2. Switch off IRQ and DMA.
3. Is device already programmed?
    - If yes, then reset its LOCK bit.
    - When resetting this bit, the ‘LOCK’ signal sense is a user option.
4. Double check status, by attempting to read test registers. The data should be invalid.
5. Reset FPGA device by strobing the **/PROGRAM** pin low.
6. Wait for device to clear memory. Wait for N mS where N is the number of frames in the device. Refer to the Data Book for exact details. Must not use system timers, due to disabled interrupts.
7. While keeping the **/PROGRAM** pin high, write all of the array data into the device.
    - Write data bits from array, # bits*.
8. Clock the device one to two times more to enter the user mode. It may be necessary to refer to the Data Book for exact details of what is required to bring the device out of the configuration state and make it operational as this is different for each family of devices.

NOTE: * Of course if a parallel mode is used then the code must handle bytes.

When developing configuration schemes and circuits, the signals detailed inTable 1 and Table 2 are available. Signals that are required as part of the device control during configuration are designated as ‘Mandatory’, and the required states for these signals is given. Other signals that provide status, (often useful in user circuits), and other completely optional signals are also described.

Ultra-fast Re-Configuration Scheme

An adaptive clocking scheme can be used to determine an optimal clock frequency for applications requiring maximum configuration speed for downloading numerous different configurations.

The configuration error protection used in Xilinx devices is very robust. If a device configures, one can have a very high degree of confidence that the device contains the correct downloaded data. Novel initialization schemes that self time the device can be used to find the fastest possible clock rates for configuration:

1. Choose an arbitrarily high clock rate.
2. Follow the standard configuration process.
3. If the device does not configure the process is repeated with a reduced configuration clock
4. Once a successful clock rate is found, the clock frequency should be de-rated to allow for temperature changes, e.g. 10%. Future configuration attempts should then be driven at this lower clock frequency.

Using this scheme can provide significant improvements in configuration performance. The XC5200’s new high speed ‘Express’ configuration mode combined with this scheme can provide configuration data rates in excess of 200 Mbits/second.

Table 1. Interface Description

Name	Requirements	Bit #	Mnemonic	Status	Description
LOCK.	Mandatory	0	LOCK	R/W*	Internal and external signal pin. This register's o/p must be set to prevent re-initialization by undecoded IO accesses which would otherwise pulse the /PROGRAM pin of the FPGA. This register forces the external FF's pre-set control so that the FPGAs /PROGRAM pin can not be toggled. As a register it may be accessed as part of the processors address space. * NOTE: This signal must be included in the User's circuit. The register may be replaced with a hardwired level to provide the locking mechanism.
Configuration Test	Optional	1,2	TEST	R/W	These registers are read/write and allow the user to write any two bit values and read them back for test purposes, as such verification of a successful download. This allows the user to double check a successful download is available and also that the I/O address allocation is a valid assignment.
Activate User Interface	Optional	3	AUI	R/W	This register bits may be used to enable the rest of the User circuitry. This is recommended for debugging designs by isolating the design from the bus interface. In a development environment serious design errors may bring down the system CPU.
I/O Address Space	Mandatory for a PC interface.		IOADDR	R R/W	These Bits may be hardwired as equivalents of the commonly used DIP switches normally used in IBM PC expansion cards for I/O Address setup. Automatic I/O Address allocation Alternatively these bits may be configured dynamically as part of a user defined initialization process, after successful completion of the download process. In this case the I/O address for the FPGA Bus interface may be written into the device, perhaps after an earlier user defined program or procedure has interrogated the I/O Address space to determine what is free for allocation. The development of this code Idea is not presented in this Application Note and is left for the interested user.

Table 2. Configuration Signal Functional Description.

Name	Type	I/O	Config. Modes	Required Action	Description
Mandatory Control Signals					All of these signals must be used during configuration.
M0 M1 M2	Control	I	(see mode table below)		

M2	M1	M0	Mode.	Comments.
0	0	0	Master Serial	Bit Serial mode, CCLK can be used to drive other devices
1	1	1	Slave Serial	Bit Serial mode, CCLK is an input CCLK must be generated from another source
1	0	0	Master Parallel Up.	Byte Wide mode, address starts at 000...0 and ⇧
1	1	0	Master Parallel Down.	Byte Wide mode, address starts at FFF...F and ⇩
0	1	1	Peripheral Synchronous.	Byte Wide mode, (a parallel slave mode)
1	0	1	Peripheral Asynchronous.	Byte Wide mode
0	1	0	Reserved	
0	0	1	Reserved	

Name	Type	I/O	Config. Modes	Required Action	Description
PROGRAM (cf RESET in XC3xxxA)	Control	I	All	Pulsed Low, at the start of all new downloads	A falling edge on this pin forces the start of an internal configuration memory clear. This pin must be raised high before the clear cycle is finished for configuration to continue. If this pin is still low at the end of the current clear cycle, the device will initialize a new clear cycle. This mechanism may be used to delay configuration indefinitely. In situations where delayed configuration is desirable, asserting a Low on the **INIT** pin is preferred. The time that is required for the internal memory clear cycle is FPGA size dependent.
DIN/D0	Data	I	All	Data input	Data In. This is the data input for all serial modes. In Parallel modes, it is D0.
CCLK	Clock	I	Slave Synch Peripheral.	Configuration Clock	The rising edge of **CCLK** writes the data on **DIN** into the device. After all configuration data is clocked into the device
		O	Master Async Peripheral.		Internally generated configuration clock, used to drive either additional daisy chained FPGAs and or external serial memory devices containing configuration data. This clock may also be used to feed other FPGAs, which must then operate with their own clocks as inputs as detailed above.
Signals with Mandatory Input conditions					All of these signals must be conditioned in some way during configuration.
DONE (cf D/P in XC3xxxA)	Control	I	All	Must be pulled high with a resistor, (2-8 kΩ)	Assertion of a low level can be used to delay the global logic initialization or the enabling of outputs.
	Status	O	All		A high level on this pin Indicates the successful download of configuration data. This is only active if the data stream passed all Error checks, using a minimum of 200 error check bits. This signal only becomes asserted after a minimum of one additional CCLK edge applied after the last bit of download configuration data has been clocked into the device. For exact details please refer to the data book.
INIT	Control	I		Must be pulled high with a resistor, (2-8 kΩ)	A low level applied to this pin may be may be used to delay configuration indefinitely. In situations where delayed configuration is desirable use of this pin rather than the mechanism available using the **PROGRAM** is preferred.
	Status	O	All		Output of a low level indicates that an error occurred during the download of configuration data.

Bibliography

The following Xilinx documents provide information on configuration:

The Xilinx Data Book.

IBM PC AT Technical Reference Manual.

Xilinx - Configuration Guidelines.

Xilinx - Debugging Configuration Schemes.

Using the Design Files

This design and source files are available from the Xilinx BBS (see “Xilinx Technical Bulletin Board” in Section 6 of the **Xilinx Data Book**. This section describes what software is required to run the design and the steps involved. Also, please read through the ***Limitations and Restrictions*** section.

Software Requirements

The following software is required to process this design:

- PKUNZIP 2.04e, or later, unarchiving pro-gram.
- VIEWdraw or VIEWdraw-LCA schematic editor. This software is required in order to make modifications to the schematics.
- Xilinx XACT 5.0 FPGA development system, including the PPR place and route program and the X-BLOX module generator.

Using the Design on Your System

1. Create a new directory called **`PC_ISA`** on your hard disk.
2. Copy the file called **`PC_ISA.EXE`** into the **`PC_ISA`** directory.
3. Type **`PC_ISA.EXE`** on the command line. This extracts a **`README.TXT`** file, and a hierarchical archive of the design files called **`PC_ISA.ZIP`**.
4. Invoke **`PKUNZIP -D PC_ISA.ZIP`** to extract the files, including their hierarchical path names, onto your disk.
5. Edit the **`VIEWDRAW.INI`** file. Make sure that the VIEWlogic design library pointers are set appropriately for your machine. You will find the library pointers near the end of the file.
6. Invoke XDM.
7. Set the part type to XC4003-6PQ100C.
8. Run XMAKE on PC_ISA.MAK to process the design. The schematic files are named PC_ISA.1 through PC_ISA.XXX

Limitations and Restrictions

WARNING: THIS IS AN UNTESTED DESIGN.

Design Support and Feedback

This application note may undergo future revisions and additions. If you would like to be updated with new versions of this application note, or if you have questions, comments, or suggestions please send an E-mail to

`applications@xilinx.com`

or a FAX addressed to "PC_ISA Application Note Developers" sent to

1+(408) 879-4442.

IMPORTANT: Please be sure to include which version of the application note you are using. The version number is in the lower right-hand corner of page 1.