A Plug and Play Interface Using Xilinx FPGAs

May, 1995

Application Note BY BILL ALLAIRE AND STEVE KNAPP

Summary

This Application Note describes a Plug and Play ISA interface reference design using a Xilinx XC4003-6PQ100C, or larger, FPGA device. This design implements the features used in a majority of Plug and Play designs but does not implement every option available within the Plug and Play specification.

Table of Contents

Assumptions....1
Overview....1
The Xilinx Solution....1
Plug and Play Card Configuration Sequence....2
Plug and Play Auto-configuration Ports....3
Sending and Verifying the Initiation Key....4
Isolation Protocol....4
Programming Plug and Play Devices....6
Plug and Play Register Summary....7
Control Register Space....7
Plug and Play Isolation Sequence....9
Reading Resource Data....9
Configuring Card Resource Usage....9
Resource Programming....10
Run Time Access to Plug and Play registers....10
High-Volume, Cost-Reduction Strategies....10
Using the Xilinx Plug and Play Design Files....11
Additional Resources....12

Assumptions

This application note assumes that the reader has:

- A copy of the Plug and Play ISA Specification.
- A basic understanding of the Plug and Play ISA functionality.
- An understanding of the ISA bus functionality.
- A working knowledge of the Xilinx XACT development system.

The "Additional Resources" section describes where to find additional information on these topics.

Overview

The ISA bus is the most popular expansion standard in the PC industry. Unfortunately, ISA has no defined hardware or software mechanism for allocating resources among the various ISA cards in a system. Consequently, users typically configure ISA cards using "jumpers" that change the decode maps for memory and I/O space and steer the DMA and interrupt signals to different pins on the bus. Usually, system configuration files need to be updated to reflect these changes. Users typically resolve any sharing conflicts between boards in the system by referring to the documentation provided by each card manufacturer. For most users, this configuration process can be frustrating, time-consuming, and error-prone.

Other common PC bus standards—like Micro Channel, EISA, and PCI—have hardware and software mechanisms to automatically identify the resources requested by a card and to resolve conflicts. Unfortunately, these existing mechanisms are not compatible with the huge installed base of PCs with ISA card slots.

The Plug and Play ISA specification provides a hardware and software mechanism to resolve resource conflicts between various Plug and Play-compatible ISA cards in a system. The Plug and Play software optimally allocates system resources between the Plug and Play ISA cards and other devices in the system *without* user intervention.

A system that uses only Plug Play ISA cards should automatically configure. However, Plug and Play ISA cards will usually co-exist with current generation ISA cards in the same system. In such cases, user interaction may still be necessary, although the configuration is augmented in the BIOS and/or operating system to manage and arbitrate ISA bus resources.

The Plug and Play ISA specification requires that each card support identification, resource usage determination, conflict detection, and conflict resolution. The specification also presents a process for software to automatically configure the new cards without user intervention.

The Xilinx Solution

This application note includes all the logic and state machines needed to fully support the Plug and Play ISA revision 1.0a using a single Xilinx XC4003-6PQ100C FPGA device. The design includes five basic modules:

- Plug and Play Core Module
- I/O Decoder Module
- IRQ Steering Module

- DMA Request Steering Module
- EEPROM Control Module

The design is easily modified to include only those features that are required in a specific application. Additional logical functions may be integrated into the same design to minimize the number of system components. The component cost can be reduced even further via Xilinx HardWire™ gate arrays.

Features

- Fully compatible with Plug and Play ISA revision 1.0a
- Supports 8-bit ISA bus interface—expandable to 16-bits
- Card isolation
- Initialization key detection
- Card resource request
- Card identification
- Card resource configuration
- I/O decode configuration
- Implements all 7 DMA channels
- IRQ channel configuration—all 15 channels supported
- Optional serial interface to industry-standard 93C46-style EEPROM to hold additional Plug and Play resource data
- Supports writes to the optional EEPROM to store customer data

Plug and Play Card Configuration Sequence

The key aspect of making Plug and Play cards easy to install is their ability to auto-configure. The BIOS performs these major auto-configuration steps:

- Issues an initiation key to begin the configuration process. Plug and Play ISA cards power-up in a quiescent state and wait for initiation before activating.
- Puts all Plug and Play ISA cards in configuration mode.
- Isolates one Plug and Play ISA card at a time.
- Assigns a handle and reads the isolated card's resource data structure.
- After determining the resource requirements and capabilities for all cards, the BIOS uses the handle to assign conflict free resources to each card.
- Activates all Plug and Play ISA cards and removes them from configuration mode.

The Plug and Play software identifies and configures devices using a set of commands executed through three, 8-bit I/O ports. 16-bit accesses (assertion of IOCS16#) to the configuration ports are not supported.

A special sequence of data writes to one of the ports enables the logic on all the Plug and Play cards in the system. This sequence is referred to as the ***initiation key***.

All Plug and Play cards respond to the same I/O port addresses. Consequently, the Plug and Play software needs an isolation mechanism to address one particular card at a time. The isolation protocol uses a unique

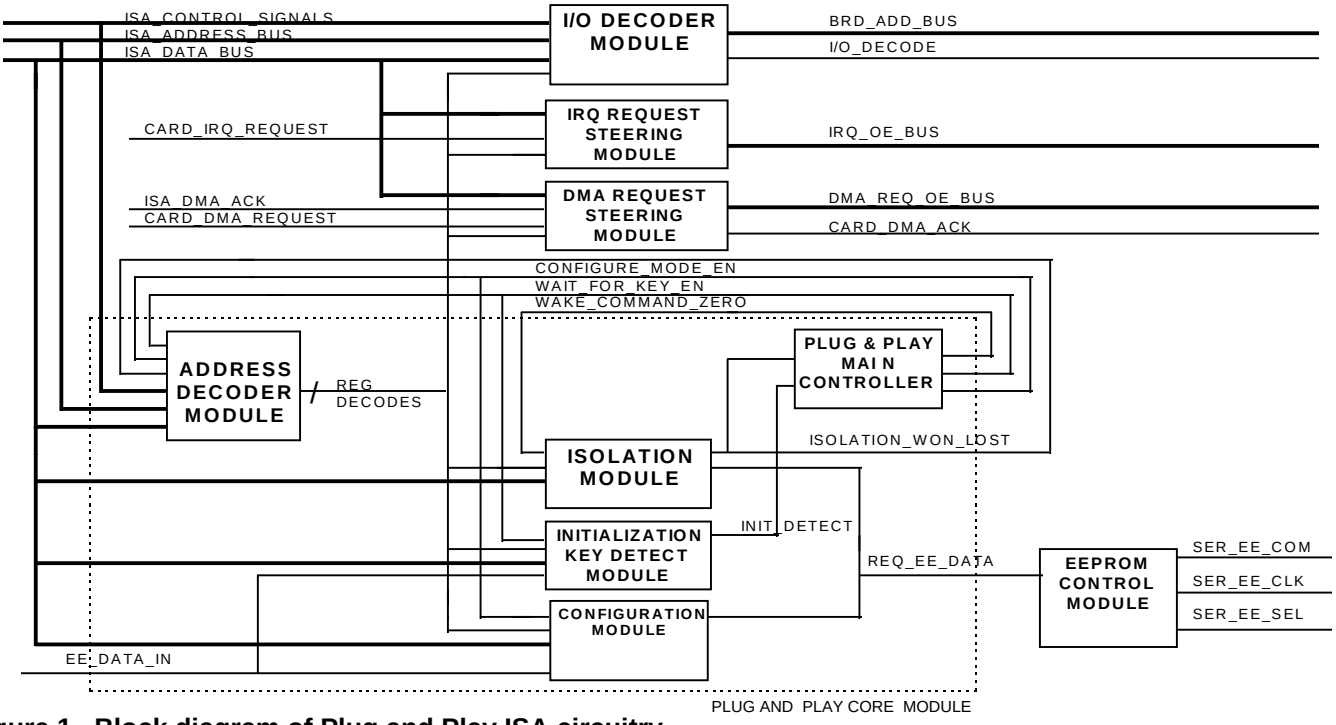


Figure 1. Block diagram of Plug and Play ISA circuitry.

number built into each card—called the ***serial identifier***—to isolate one Plug and Play card at a time. After isolation, the Plug and Play software assigns each card a unique handle, called the ***Card Select Number (CSN)***, which is used to select that Plug and Play card. Using a handle eliminates the need to use a more elaborate and time consuming isolation protocol in order to select a unique card.

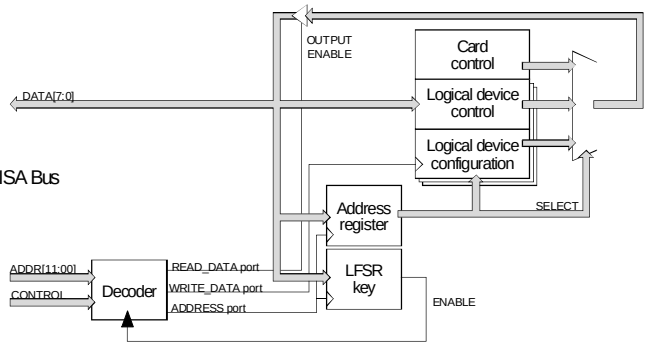


Figure 2. Logic flow for auto-configuration.

Each card supports a readable data structure that describes the resources supported and those requested by the functions on that card. The data structure supports the concept of multiple functions per ISA card. Each function is defined as a logical device. Plug and Play resource information is provided for each logical device and each logical device is independently configured through the Plug and Play standard registers.

Following isolation, the Plug and Play software reads the resource data structure on each card. When all resource capabilities and demands are known, an arbitration process determines the resources allocated to each ISA card.

The configuration of ISA cards is done through the command registers specified for each resource type. However, some ISA functions may not be reconfigurable. In these cases, the resources requested will be equivalent to the resources supported. However, the resource data structure informs the arbiter that it cannot assign these resources to other Plug and Play cards in the system.

After the assignment of resources, an I/O conflict detection mechanism may be invoked. This mechanism provides a means to insure that I/O resources assigned are not in conflict with standard ISA cards.

The command set also supports the ability to activate or deactivate the function(s) on the card.

After configuration is complete, Plug and Play cards are removed from configuration mode. The initiation key needs to be re-issued in order to re-enable configuration mode. This prevents accidental erasure of the configuration information.

Plug and Play Auto-configuration Ports

The system software access the configuration space on each Plug and Play ISA card through three 8-bit ports. The ports are listed in Table 1. The configuration space is implemented as a set of 8-bit registers. The Plug and Play software uses these registers to issue commands, check status, access the resource data information, and configure the Plug and Play hardware. The decoding logic is shown in ***Figure 10—Plug and Play Ports (PP_DEC.1)*** on page 20.

The ports were cleverly chosen so as to avoid conflicts in the installed base of ISA functions, while at the same time minimizing the number of ports needed in the ISA I/O space.

The ADDRESS and WRITE_DATA ports are located at fixed addresses. The WRITE_DATA port is located at an address alias of the ADDRESS port. All three auto-configuration ports use a 12-bit ISA address decode.

Table 1. Auto-configuration Ports

Port Name	Location	Type
ADDRESS	0x0279 (Printer status port)	Write-only
WRITE_DATA	0x0A79 (Printer status port + 0x0800)	Write-only
READ_DATA	Relocatable in range 0x0203 to 0x03FF	Read-only

The READ_DATA port is relocatable within the I/O range from 0x0203h to 0x03FFh. This is the only readable auto-configuration port.

ADDRESS Port

The Plug and Play registers are accessed by

- Writing the address of the desired register to the ADDRESS port, followed by a
- Read of data from the READ_DATA port or a write of data to the WRITE_DATA port.

A write to the ADDRESS port may be followed by any number of WRITE_DATA or READ_DATA accesses to the same register location without the need to write to the ADDRESS port before each access.

The ADDRESS port is also the write destination of the initiation key, which is described later.

Note the extra three flip-flops used to synchronize the write strobe signal, WR_STRB, to the ISA bus clock, CLK. Similar circuitry is used to sychronize the write strobe when writing to the internal registers.

WRITE_DATA Port

The WRITE_DATA port is used to write information to the Plug and Play registers. The destination of the data is determined by the last setting of the ADDRESS port.

READ_DATA Port

The READ_DATA port is used to read information from the Plug and Play registers. The source of the data is determined by the last setting of the ADDRESS port.

The address of the READ_DATA port is set by writing the proper value to a Plug and Play control register. The isolation protocol verifies that the location selected for the READ_DATA port is free of conflict.

Sending and Verifying the Initiation Key

The Plug and Play logic is quiescent on power up and must be enabled by software.

Logic Modules Used

The logic that performs this function is included in the following modules:

1. the INIT_DETECT symbol shown in ***Figure 7—Plug and Play Demonstration Design, Sheet 1 (PLGPLY.1)*** on page 14.
2. the underlying logic for INIT_DETECT shown in ***Figure 18—Initiation Key Schematic (INITKEY.1)*** on page 32.
3. the ABEL source code for the initiation key state machine shown in ***Figure 20—ABEL source file for Initiation Key state*** machine (INIT_KEY.ABL) on page 34.
4. the underlying logic for the LFSR initiation key shown in ***Figure 19—Initiation Key LFSR Schematic (LFSR_KEY.1)*** on page 33.
5. the MAIN_CONTROL symbol shown in ***Figure 7—Plug and Play Demonstration Design, Sheet 1 (PLGPLY.1)*** on page 14.
6. the ABEL source for the Plug and Play main controller (states 'wait_init' and 'sleep') shown in ***Figure 17—ABEL source file for Plug and Play*** main state machine on page 30.

The initiation key places the Plug and Play logic into configuration mode. This is done by a predefined series of writes to the ADDRESS port. The write sequence is decoded by the logic and state machine in the INIT_DETECT symbol. If the proper series of I/O writes is detected, then the Plug and Play auto-configuration ports are enabled.

The hardware verification of the initiation key is accomplished using a Linear Feedback Shift Register (LFSR) shown in Figure 19. The LFSR powers-up preloaded with 0x6Ah. Software generates the LFSR sequence, also shown in Figure 19, and writes it to the ADDRESS port as a sequence of 8-bit write cycles. The hardware compares the byte of write data with the value in the shift register at each write. Any time the data does not match, the hardware resets the LFSR to the initial value. Software should reset the LFSR to its initial value by a sequence of two write cycles of 0x00 to the ADDRESS port before the initiation key is sent.

Isolation Protocol

A simple algorithm is used to isolate each Plug and Play card. This algorithm uses the signals on the ISA bus and requires lock-step operation between the Plug and Play hardware and the isolation software. The state diagram for the isolation process is shown in Figure 4.

Logic Modules Used

The logic that performs the isolation protocol is shown in the following modules:

1. the ISOLATION symbol shown in ***Figure 7—Plug and Play Demonstration Design, Sheet 1 (PLGPLY.1)*** on page 14.
2. the underlying logic for ISOLATION shown in ***Figure 15—Isolation Protocol Schematic (ISO_MOD.1)*** on page 26.
3. the underlying ABEL source code for the isolation state machine shown in ***Figure 16—Isolation State Machine Listing (ISOLATE.ABL)*** on page 27.
4. the MAIN_CONTROL symbol shown in ***Figure 7—Plug and Play Demonstration Design, Sheet 1 (PLGPLY.1)*** on page 14.
5. the ABEL source for the Plug and Play main controller shown in ***Figure 17—ABEL source file for Plug and Play*** main state machine on page 30.

Serial Identifier

The key element of this mechanism is that each card contains a unique number, called the *serial identifier*. The serial identifier is a 72-bit unique, non-zero, number composed of two, 32-bit fields and an 8-bit checksum as shown in Figure 3. The first 32-bit field is a vendor identifier. The other 32 bits can be any value, for example, a serial number, part of a LAN address, or a static number, as long as there will never be two cards in a single system with the same 64 bit number. The serial identifier is accessed bit-serially by the isolation logic and is used to differentiate the cards.

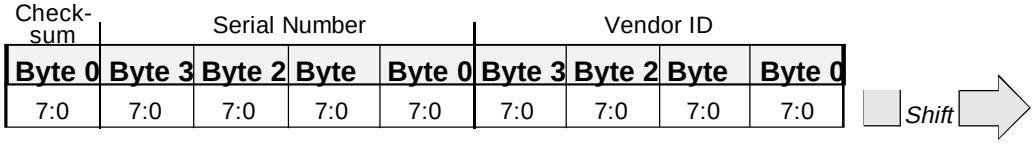


Figure 3. Serial Identifier format

The shift order for all Plug and Play serial isolation and resource data is always defined as bit[0], bit[1], and so on through bit[7].

In this design, the serial identifier is stored in external EEPROM along with the card's resource information. The serial identifier can also be stored internally using CLB ROMs for simple boards with few resources. Using CLB ROMs simplifies the overall logic.

The Plug and Play software sends the initiation key to all Plug and Play cards to place them into configuration mode. The software is then ready to perform the isolation protocol.

Isolating the Cards

The hardware on each card then expects 72 pairs of I/O read accesses to the READ_DATA port. The card's response to these reads depends on the value of each bit of the serial identifier which is being examined one bit at a time. The response is controlled by the isolation ABEL state machine defined in Figure 16.

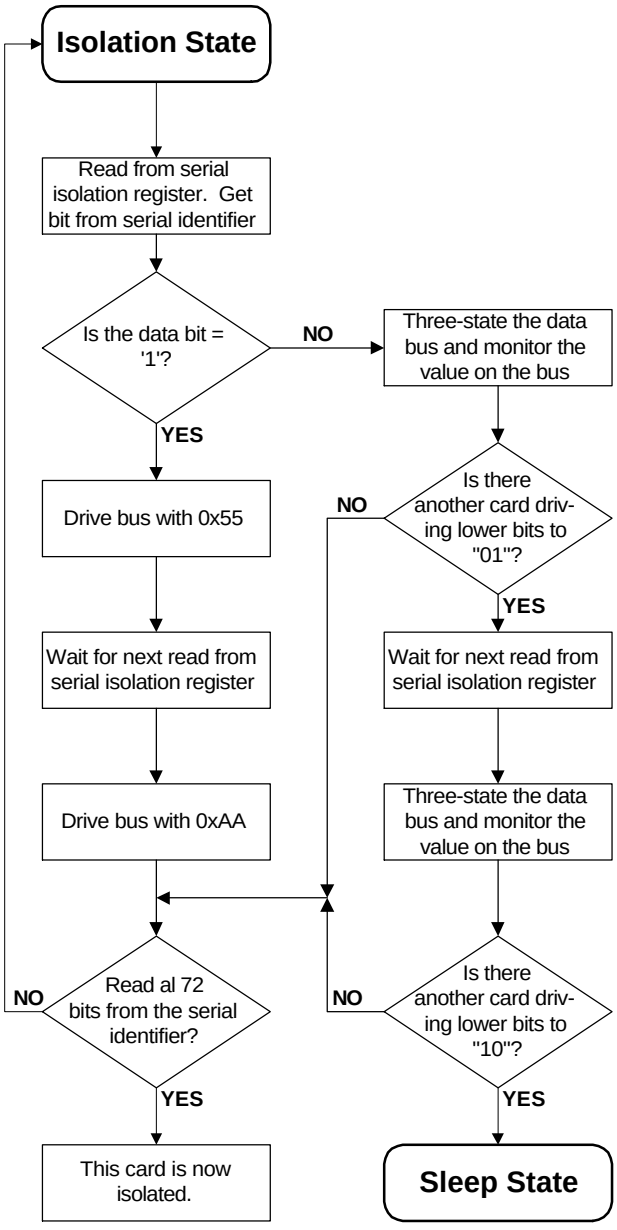


Figure 4. Isolation process state diagram.

- If the current bit of the serial identifier is a "1", then drive the data bus to 0x55 to complete the first I/O read cycle (State 's3' in Figure 16).
- If the bit is "0", then the data bus is high impedance.
- All cards in high impedance check the data bus during the I/O read cycle to sense if another card is driving D[1:0] to "01" (State 's6' in Figure 16).
- During the second I/O read, the card(s) that drove the 0x55 now drive a 0xAA (State 's4' in Figure 16).
- All high impedance cards check the data bus to sense if another card is driving D[1:0] to "10" (State 's8' in Figure 16).

During each read cycle, the Plug and Play hardware drives the entire 8-bit data bus, but only checks the lower 2 bits.

The software checks that it receives 0x55 and 0xAA data returned from each pair of I/O reads. If both 0x55 and 0xAA are read back, then the software assumes that the hardware had a "1" bit in that position. All other results are assumed to be a "0."

If a high impedance card senses another card driving the data bus with the appropriate data during both cycles, then the high impedance card ceases to participate in the current round of card isolation. The testing for another card driving the bus is done during States 's6,' 's7,' and 's8' in Figure 16. If another card is detected, then the card jumps back to State 's0.' All the cards that lose on in the current iteration will participate in future iterations of the isolation protocol.

If a card was driving the bus or if the card was in high impedance and did not sense another card driving the bus, then it should prepare for the next pair of I/O reads. The card shifts the serial identifier by one bit and uses the shifted bit to decide its response. States 's11' and 's12' shown in Figure 16 perform this function.

The above sequence is repeated for the all 72 bits in the serial identifier. During the first 64 bits, software generates a checksum using the received data. The checksum is compared with the checksum read back in the last 8 bits of the sequence. The software checksum algorithm can be found in Appendix B of the **Plug and Play ISA Specification**.

Special Software Considerations

There are two other special considerations for the software protocol. During an iteration, it is possible that:

- The 0x55 and 0xAA combination is never detected.
- The checksum does not match.

If the software encounters either of these cases on the first iteration, it must assumed that there is a conflict on

the READ_DATA. If a conflict is detected, then the READ_DATA port needs to be relocated.

The above process is repeated until a non-conflicting location for the READ_DATA port is found. The entire range between 0x200 and 0x3FF is available, however in practice it is expected that only a few locations will be tried before software determines that no Plug and Play cards are present.

During subsequent iterations, the occurrence of either of these two special cases should be interpreted as the absence of any further Plug and Play cards (i.e.—the last card was found in the previous iteration). This terminates the isolation protocol.

The software must delay 1 msec prior to starting the first pair of isolation reads, and must wait 250 µsec between each subsequent pair of isolation reads. This delay gives the ISA card time to access information from possibly very slow storage devices.

One Card Remains—The Isolation Winner

At the end of isolation process, only one card remains. This card is assigned a handle referred to as the *Card Select Number* (CSN). The CSN will be used later by the software to select the card. The logic that controls the process is contained in the MAIN_CONTROL symbol shown in Figure 7 and in the ABEL state machine listing showin in Figure 17. The specific states involved are named ‘iso_mode’ and ‘wait_csn.’

Cards that have already been assigned a CSN do not participate in subsequent iterations of the isolation protocol: They have already been uniquely identified. A cards must be assigned a CSN before it will respond to other Plug and Play commands.

The protocol permits the 8-bit checksum to be stored in non-volatile memory on the card or generated by on-card logic in real-time. The same LFSR algorithm described in the initiation key section is used in the checksum generation. In this example, the checksum is stored in the EEPROM.

Plug and Play cards must not drive the IOCHRDY signal during serial isolation. However, cards may drive IOCHRDY at any other time.

Programming Plug and Play Devices

This section describes how configuration resource data is read from Plug and Play ISA cards as well as how resource selections are programmed. The Plug and Play state machine and Plug and Play commands are introduced.

The *Card Select Number* (CSN) register is written during the isolation process and is actively involved in most Plug and Play commands. The CSN is an 8-bit register used to select one or more ISA cards and is shown as CSN_REGISTER toward the top of Figure 7.

The CSN is an 8-bit register because it allows a wide variety of devices to manage their configuration and control using this mechanism. The CSN is defined such that all cards power-up with this register set to zero (0x00). Once a card has been isolated, the CSN on that card is assigned a unique, non-zero value. This value allows the Plug and Play software to select this card at later points in the configuration process without going through the isolation protocol again.

Logic Modules Used

1. the CSN_REGISTER symbol shown in ***Figure 7—Plug and Play Demonstration Design, Sheet 1 (PLGPLY.1)*** on page 14.
2. the MAIN_CONTROL symbol shown in ***Figure 7—Plug and Play Demonstration Design, Sheet 1 (PLGPLY.1)*** on page 14.
3. the ABEL source for the Plug and Play main controller shown in ***Figure 17—ABEL source file for Plug and Play*** main state machine on page 30.

Main Controller States

The Plug and Play control states are summarized as follows:

- ***Wait for Key*** — State ‘wait_init’ in Figure 17. All cards enter this state after initial power-up or in response to the Wait for Key command. No commands are active in this state until the initiation key is detected on the ISA bus. The ***Wait for Key*** state is the default state for Plug and Play cards during normal system operation. After configuration and activation, software should return all cards to this state.
- ***Sleep*** — State ‘sleep’ in Figure 17. In this state, Plug and Play cards wait for a Wake[CSN] command. This command selectively enables one or more cards to enter either the ***Isolation*** or ***Config*** states based on the write data and the value of the CSN on each card. Cards leave the ***Sleep*** state in response to a Wake[CSN] command when the value of write data bits[7:0] of the command matches the card’s unique CSN. All the cards that have not been assigned a CSN value have CSN=0x00 (the default value). If the write data for the Wake[CSN] command is zero then all cards that have not been assigned a CSN will enter the ***Isolation*** state. If the write data for the Wake[CSN] command is not zero then the one card whose assigned CSN matches the parameter of the Wake[CSN] command will enter the ***Config*** state.
- ***Isolation*** — State ‘iso_mode’ in Figure 17. In this state, Plug and Play cards respond to reads of the Serial Isolation Register as described in the previous section on the isolation process. Once a card is isolated, it is a assigned unique CSN value in State ‘wait_csn’ in Figure 17. This number will later be used by the Wake[CSN] command to select the card.

Once the CSN is written, the card transitions to the ***Config*** state.

- ***Config*** — State 'config' in Figure 17. A card in the ***Config*** state responds to all configuration commands including reading the card's resource configuration information and programming the card's resource selections. Only one card may be in this state at a time.

Plug and Play Register Summary

Plug and Play card standard register space is divided into three parts; card control, logical device control, and logical device configuration.

There is exactly one of each card control register on each ISA card. Card control registers are used for global functions that control the entire card. Logical device control registers and logical device configuration registers are repeated for each logical device. Logical device control registers control device functions, such as enabling the device onto the ISA bus. Logical device configuration registers are used to program the device's ISA bus resource use. There are several vendor defined registers in all three register locations so vendors may configure non-standard ISA resources through the Plug and Play mechanism as well.

As implemented, this design currently supports only one logical device—common for most applications. However, additional logical devices may be supported by connecting the appropriate signals to the LOG_DEV_REG_SEL symbol and the LOGIC_DEVICE_DECODE symbol shown in Figure 8 on page 16.

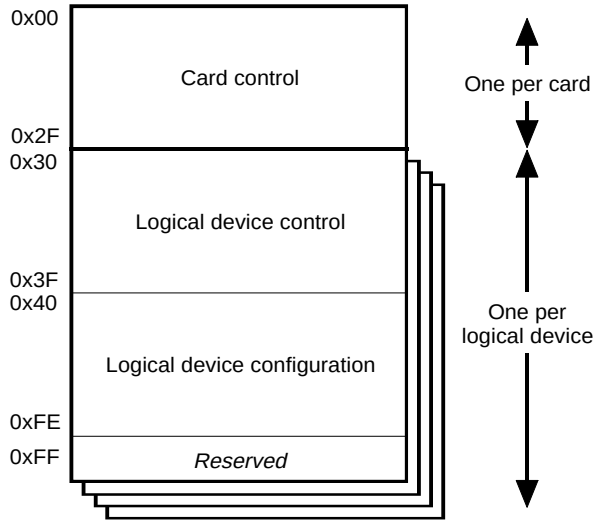


Figure 5. Plug and Play High-Level Register Map

Control Register Space

Plug and Play cards respond to commands written to Plug and Play registers as well as certain ISA bus conditions. The address decoding for these registers and commands is located in the REGISTER_DECODE symbol in Figure 7 and the underlying logic shown in Figure 10. These associated commands are summarized below:

- **RESET_DRV** — This is the ISA bus reset signal. In this application, the ISA_RESET directly drives the GSR input of the STARTUP symbol shown in Figure 8. When the card detects this signal, the Plug and Play main controller state machine enters the ***Wait for Key*** state (State 'wait_init' in Figure 18). All CSNs are reset to 0x00. The configuration registers for all logical devices are loaded with their power-up values from non-volatile memory or jumpers. All non-boot logical devices become inactive. Boot devices become active using their power-up ISA resources. Note: The software must delay 1 msec after RESET_DRV before accessing the auto-configuration ports.
- **Config control register** — The Config Control register consists of three independent commands which are activated by writing a "1" to their corresponding register bits. The logic is contained in the CONFIG_CNTRL symbol in Figure 7 and the underlying logic is shown in Figure 14. These bits are asserted automatically reset by the logic two clock cycles after the commands execute.
  - **Reset command** — The Reset command is sent to the Plug and Play cards by writing a value of 0x01 to the CONFIG_CNTRL register. All Plug and Play cards in any state, *except* **Wait for Key**, respond to this command. This command performs a reset function on all logical devices. This resets the contents of configuration registers to their default state. The configuration registers for all logical devices are loaded with their power-up values from non-volatile memory or jumpers. ***The READ_DATA port, CSN and Plug and Play state are preserved.*** Note: The software must delay 1 msec after issuing the reset command before accessing the auto-configuration ports.
  - **Wait for Key command** — The Wait for Key command is sent to the Plug and Play cards by writing a value of 0x02 to the CONFIG_CNTRL register. All Plug and Play cards in *any* state will respond to this command. This command forces all Plug and Play cards to enter the ***Wait for Key*** state. The CSNs are preserved and no logical device status is changed. This command is accomplished by resetting the Plug and Play main controller state machine to State 'wait_init' shown in Figure 17.
  - **Reset CSN command** — The Reset CSN command is sent to the Plug and Play cards by writing a value of 0x04 to the CONFIG_CNTRL register. All Plug and Play cards in any state, *except* **Wait for Key**, will reset their CSN to 0x00. This command is accomplished by resetting the CSN_REGISTER symbol in Figure 7.

NOTE: On a CTRL-ALT-DEL key sequence, the BIOS issues a reset of all logical devices, restores configuration registers to their default values, and returns all cards to the **Wait for Key** state (i.e.—write a value of 0x03 to the Config Control register). This retains the CSNs and READ_DATA port and will eliminate the need to go through the isolation sequence again. A write to this register with all three bits set is equivalent to a RESET_DRV event.

- **Set RD_DATA Port command** — This command sets the address of the READ_DATA Port for all Plug and Play cards. Write data bits[7:0] is used as ISA I/O bus address bits[9:2]. The ISA bus address bits[1:0] is fixed at binary "11." The ISA bus address bits[15:10] is fixed at binary "000000." This command can *only* be used in the ***Isolation*** state. The exact method for setting the read data port is:
  - Issue the Initiation Key
  - Send command `Wake[0]`
  - Send command `Set RD_DATA Port`

  Note: After a RESET_DRV or Reset CSN command, this register is considered uninitialized and must be reinitialized.
- **Serial Isolation register** — A read from the Serial Isolation register causes Plug and Play cards in the ***Isolation*** state to respond to the ISA bus read cycle as described in the Isolation Protocol section above. Cards that "lose" the isolation protocol will enter the ***Sleep*** state.
- **Card Select Number** — A Card Select Number is uniquely assigned to each Plug and Play card when the card has been isolated and is the only card in the ***Isolation*** state. A Card Select Number of zero represents an unidentified card, the default state. Valid Card Select Numbers for identified ISA cards range from 1 to 255 and must be assigned sequentially starting from 1. The Card Select Number is used to select a card via the Wake[CSN] command as described above. The Card Select Number on all ISA cards is set to zero on a RESET_DRV command. *The CSN is never set to zero using the CSN register.*
- **Wake[CSN] command** — This command is used to bring ISA cards in the ***Sleep*** state to either the ***Isolation*** state or the ***Config*** state.
  - A Wake[CSN] command with a parameter of *zero* will force all cards without a CSN to enter the **Isolation** state.
  - A Wake[CSN] command with a parameter *other than zero* will force the card with the matching CSN to enter the ***Config*** state.
  - Any card in the ***Isolation*** or ***Config*** states that receives a Wake[CSN] command with a parameter that *does not match its CSN* will transition to the ***Sleep*** state.

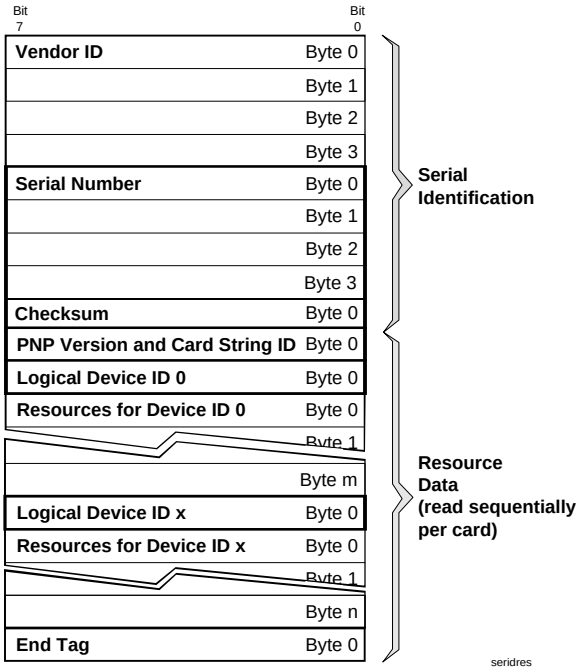


Figure 6. Serial Identifier and Resource Data

All Plug and Play cards function as if their 72-bit serial identifier and their resource data come from a single serial device. The pointer to this data is reset to the beginning whenever a card receives any Wake[CSN] command.

- **Resource Data register** — A read of the Resource Data register will return one byte of resource data from the Plug and Play card in the ***Config*** state. Resource data is always returned byte sequentially. The Status register must always be read to confirm that resource data is available before reading the Resource Data register.
- **Status register** — Bit[0] of the Status register indicates that the next byte of resource data is available to be read. If this bit is one, then data is available otherwise resource data is not yet available. The Plug and Play software will poll this location until bit[0] is set, then the next data byte from the Resource Data register is read.
- **Logical Device Number register** — The logical device number register is used to select which logical device the following configuration commands will operate on. Cards may contain more than one logical device, in which case the logical device is selected by writing the 8-bit logical device number into this register. The logical device number is determined by the order in which the logical devices are read from the resource data. The first logical device number is 0, the second is 1, and so on.
- **I/O Range Check register** — The I/O Range Check register is not implemented in this design. This

command is optional and is not implemented on cards that do not have configurable I/O port ranges.

- **Activate register** — The Activate register is a read/write register that is used to activate a logical device. An active logical device responds to all ISA bus cycles as per its normal operation. An inactive logical device does not respond to nor drive any ISA bus signals. Bit[0] is the active bit, if it is set to "1" then the logical device is active, otherwise it is inactive.

Plug and Play Isolation Sequence

On power up, all Plug and Play cards detect RESET_DRV, set their CSN to 0, and enter the ***Wait for Key*** state. There is a required 1 msec delay from either a RESET_DRV or ResetCmd to any Plug and Play port access to allow a card to load initial configuration information from a non-volatile device.

Cards in the ***Wait for Key*** state do not respond to any access to their auto-configuration ports until the initiation key is detected. Cards ignore all ISA accesses to their Plug and Play interface.

When the cards have received the initiation key, they enter the ***Sleep*** state. In this state, the cards listen for a Wake[CSN] command with the write data set to 0x00. This Wake[CSN] command will send all cards to the ***Isolation*** state and reset the serial identifier/resource data pointer to the beginning.

The first time the cards enter the ***Isolation*** state it is necessary to set the READ_DATA port address using the Set RD_DATA port command.

Next, 72 pairs of reads are performed to the Serial Isolation register to isolate a card as described previously. If the checksum read from the card is valid, then this means one card has been isolated. The isolated card remains in the ***Isolation*** state while all other cards have failed the isolation protocol and have returned to the ***Sleep*** state. The CSN on this card is set to a unique number. Writing this value causes this card to transition to the ***Config*** state. Sending a Wake[0] command causes this card to transition back to ***Sleep*** state and all cards with a CSN value of zero to transition to the ***Isolation*** state. This entire process is repeated until no Plug and Play cards are detected.

If a conflict is detected on the READ_DATA port, a Wake[0] command is issued to cause all the cards that are in the ***Isolation*** state to reset their serial identifer data pointer to the beginning while remaining in the ***Isolation*** state. Further, after a read port conflict has been detected and a Wake[0] has been issued, the software must wait 1 msec before beginning the next 72 pairs of serial isolation read cycles.

Reading Resource Data

Card resource data may only be read from cards in the ***Config*** state. A card may get to the ***Config*** state by one of two different methods:

1. A card enters the ***Config*** state in response to the card "winning" the serial isolation protocol and having a CSN assigned.
2. The card also enters the ***Config*** state in response to receiving a Wake[CSN] command that matches the card's CSN.

As shown in Figure 6, all Plug and Play cards function as if their 72-bit serial identifier and their resource data both come from a single byte-serial device (an industry-standard EEPROM, in this design). As stated earlier, the pointer to the byte-serial device is reset in response to any Wake[CSN] command. This implies that if a card enters the ***Config*** state directly from ***Sleep*** state in response to a Wake[CSN] command, the 9-byte serial identifier must be read first before the card resource data is accessed. The Vendor ID and Unique Serial Number are valid. However, the checksum byte is not valid when read in this way. For a card that enters the ***Config*** state from the ***Isolation*** state (i.e. after the isolation protocol has been run and all 72 bits of the serial identifier have been read), the first read of the Resource Data register will return resource data.

Card resource data is read by first polling the Status register and waiting for bit[0] to be set. When this bit is set it means that one byte of resource data is ready to be read from the Resource Data register. After the Resource Data register is read, the Status register must be polled before reading the next byte of resource data. This process is repeated until all resource data is read. The format of resource data is described in the following section.

The above operation implies that the hardware is responsible for accumulating 8 bits of data in the Resource Data register. When this operation is complete, the status bit[0] is set. When a read is performed on the Resource Data register, the status bit[0] is cleared, eight more bits are accumulated in the Resource Data register, then the status bit[0] is set again.

Configuring Card Resource Usage

Plug and Play cards support the following registers which are used for configuring the card's standard ISA resource usage per logical device.

- Memory Address Base registers (up to four non-contiguous ranges in the Plug and Play specification, ***not implemented in this design***)
- I/O Address Base registers (up to eight non-contiguous ranges in the Plug and Play specification, ***a single contiguous range in this design***)

- Interrupt Level Select registers (up to two separate interrupt levels in the Plug and Play specification, ***a single level in this design***)
- DMA Channel Select registers (up to two DMA channels in the Plug and Play specification, ***a single channel in this design***)

These registers are read/write and always reflect the current operation of all logical devices on the Plug and Play card. If a resource is not programmable, then the configuration register bits are read-only.

Resource Programming

Plug and Play cards are programmed by sending the card a Wake[CSN] command with the write data set to the card's CSN. This will force the one card with the matching CSN into the ***Config*** state and force all other cards into the ***Sleep*** state. Next, the logical device to be programmed is selected by writing the logical device number to the Logical Device Number register. If the card has only one logical device, this step may be skipped.

Resource configuration for each logical device is programmed into the card using the registers for I/O, memory, IRQ, and DMA selection defined in Appendix A in the **Plug and Play ISA Specification**. Each and every resource requested by a logical device must be programmed, even if the resource is not assigned. Each resource type is described below.

- **Memory Configuration** — ***NOT SUPPORTED IN THIS DESIGN.*** Memory space resource use is programmed by writing the memory base address to the memory base address registers. Next, the memory control is written with the correct 8/16/32 bit memory operation value and the decode select option. If the memory decode option was set to range length, then the range length is written to the memory upper limit/range length registers. If the memory decode option was set to upper limit, then the upper limit memory address is written to the upper limit/range length register. If no memory resource is assigned, the memory base address registers must be set to zero and the upper limit/range length registers must be set to zero.
- **I/O Space Configuration** — I/O space resource use is programmed by writing the I/O base address[15:0] to the I/O port base address registers. The logic for the base registers is shown in Figure 11. If a logical device indicated it uses 10-bit I/O space decoding, then bits [15:10] of the I/O address are not implemented on the card. If no I/O resource is assigned, the I/O base address registers must be set to zero.
- **Interrupt Request Level** — The interrupt request level for a logical device is selected by writing the interrupt request level number to the Interrupt Level Select register. This select number represents the number of the interrupt on the ISA bus. The interrupt request logic is shown in Figure 13. The reference design only uses IRQ2 through IRQ5 though all 15 are available.

  The edge/level and high/low active state of the interrupt must be written to the Interrupt Request Type register. If no interrupt is assigned, the Interrupt Level Select register must be set to 0. *The IRQ2 signal on the ISA bus is routed to IRQ 9 on the 8259 interrupt controller. To select IRQ 2 on the ISA bus, the Interrupt Level Select register must be set to 2, not 9.*
- **DMA Channel** — The DMA channel for a logical device is selected by writing the DMA channel number to the DMA Channel Select register. The DMA logic is shown in Figure 12. The select number represents the number of the DRQ/DACK channel on the ISA bus. If no DMA channel is assigned, this register must be set to 4.

The last step in the programming sequence is to set the logical device's activate bit. This forces the logical device to become active on the ISA bus at its assigned resources. When finished programming configuration registers, all cards must be set to the ***Wait for Key*** state.

Run Time Access to Plug and Play registers

Read access to Plug and Play configuration is available at all times with no impact to the function(s) on the card. However, write accesses to Plug and Play registers must be done with the full knowledge of the device driver controlling the device and the operating system environment. Even though it is possible to re-assign the CSNs during run time, this is not necessary since CSNs for all Plug and Play cards are assigned during initialization. The only exception to this case is for docking stations, hot-insertion capability or if power management is supported. It is required that prior to changing the value of any registers, the logical device be deactivated, the resource register re-programmed, and the logical device is activated again. When finished accessing Plug and Play registers, all cards must be returned to the ***Wait for Key*** state.

High-Volume, Cost-Reduction Strategies

Plug and Play ISA cards typically will be shipped in high production volumes. While the Plug and Play ISA specification is still in a state of flux, programmable logic is probably the best solution for implementing the interface.

However, as the specification matures, there are a few strategies for reducing the overall cost. In this application note, the design is implemented using an XC4003-6PQ100C. This provides a relatively low cost solution while retaining the benefits of flexible, changeable programmable logic. As of this writing (February 1995), Xilinx recently introduced a new FPGA family called the

XC5200. The attributes of the XC5200 family important for this design are:

- Pin-compatibility with corresponding XC4000 devices.
- A revolutionary, low-cost architecture by exploiting advanced 0.6µ triple-layer metal CMOS technology.

This design can be migrated to the lower-cost XC5200, when available, without changing the PC board pinout. The XC5200 offers a cost reduction over the XC4003 while remaining fully re-programmable.

Once the Plug and Play ISA specification has solidified, another option for higher-volume applications is to covert the programmable XC4003 design into a mask-programmed Xilinx XC4303 or XC4403 HardWire™ gate array. The HardWire gate array offers:

- 100% pin- and design-compatibility with the XC4003
- Low NRE charge
- "Design Once" — A low-risk conversion process using the same design files as used in the XC4003 without requiring test or simulation vectors
- 100% fault coverage without requiring user-written test vectors
- Reduced production costs

Using the Xilinx Plug and Play Design Files

This design is made available on diskette. 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.
- Xilinx XACT 5.0, or later, FPGA development system.
- VIEWdraw or VIEWdraw-LCA schematic editor with Xilinx XC4000 Unified libraries. This is required in order to make modifications to the schematics.
- Xilinx X-ABEL or Data I/O ABEL 5.0 or later. This is required in order to make modifications to the various state machine design implemented using ABEL.
- QBASIC BASIC intrepreter usually included with MS-DOS. This is only required for executing the various BASIC test programs.

Using the Design on Your System

1. Create a new directory called **PLGPLY** on your hard disk.
2. Copy the file called **PLUGPLAY.EXE** into the **PLGPLY** directory.
3. Type **PLUGPLAY.EXE** on the command line. This extracts a **README.TXT** file, and a hierarchical archive of the design files called **PLGPLY.ZIP**.
4. Invoke **PKUNZIP -D PLYGPLY.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 the XDM program.
7. Set the part type for a XC4003-6PQ100C.
8. Run XMAKE on **PLGPLY.MAK** to process the design. The schematic files named **PLGPLY.1** through **PLGPLY.3** include the top-level schematics.

Design Directories

The files are installed in separate sub-directories as shown in Table 2.

Table 2. Design Sub-directories.

Sub-Directory	Files
/SCH	VIEWdraw schematic designs
/SYM	VIEWdraw symbols used in the schematics
/WIR	VIEWdraw wire files
/BASIC	QBASIC routines used to test and access the Plug and Play design
/CMD	VIEWsim command files for simulation

Limitations and Restrictions

WARNING: THIS IS AN UNTESTED DESIGN.

Xilinx, Inc. does not make any representation or warranty regarding this design or any item based on this design. Xilinx disclaims all express and implied warranties, including but not limited to the implied fitness of this design for a particular purpose and freedom from infringement. Without limiting the generality of the foregoing, Xilinx does not make any warranty of any kind that any item developed based on this design, or any portion of it, will not infringe any copyright, patent, trade secret or other intellectual property right of any person or entity in any country. It is the responsibility of the user to seek licenses for such intellectual property rights were applicable. Xilinx shall not be liable for any damages arising out of or in connection with the use of the design including liability for lost profit, business interruption, or any other damages whatsoever.

Design Support and Feedback

This application note, like the Plug and Play ISA specification itself, is still under development. Consequently, there will most likely be future changes and additions.

If you would like to be updated with new versions of this application note, or if you have questions, comments, or suggestions, or if you discover potential problems, please send an E-mail to

PnP@xilinx.com

or a FAX addressed to "Corporate Applications: Plug and Play Application Note" sent to

1+(408) 879-4442.

Updates to this document can be received via E-mail. Send an E-mail message to **xdocs@xilinx.com** with **send 21610** in the subject header.

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.

Additional Resources

The following additional information is available on Plug and Play:

Plug and Play ISA 1.0a Specification

Available via the ComuServe Plug and Play forum (GO PLUGPLAY). Requires a CompuServe account. The specification is available as a downloadable Microsoft Word for Windows 6.0 .DOC file.

Plug and Play X-NOTES

Contains additional basic background information on the Plug and Play market and design solutions. Available from your local Xilinx sales office.

Acknowledgments

Portions reprinted by permission of Intel Corporation, Copyright/Intel Corporation 1994.

NOTES:

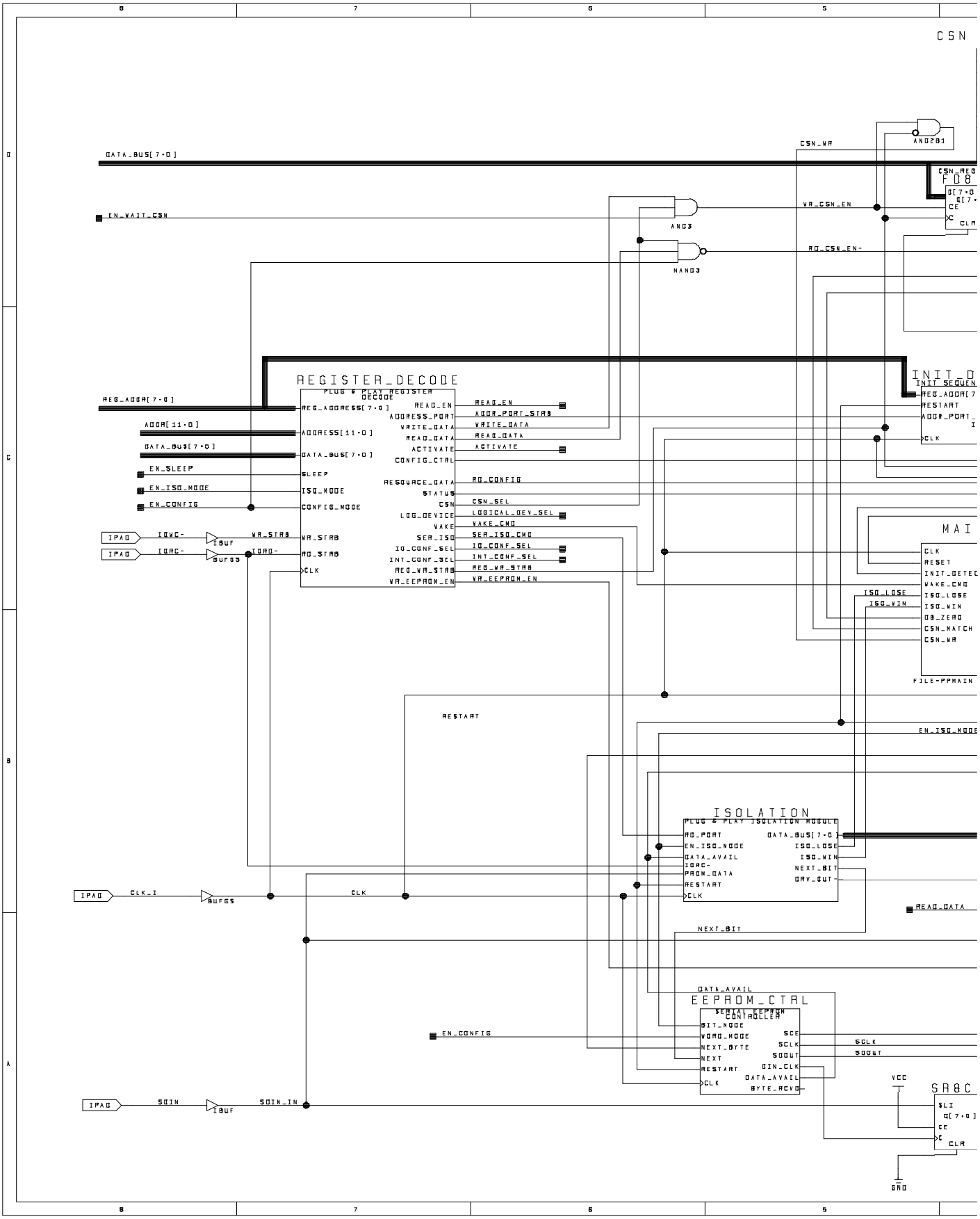
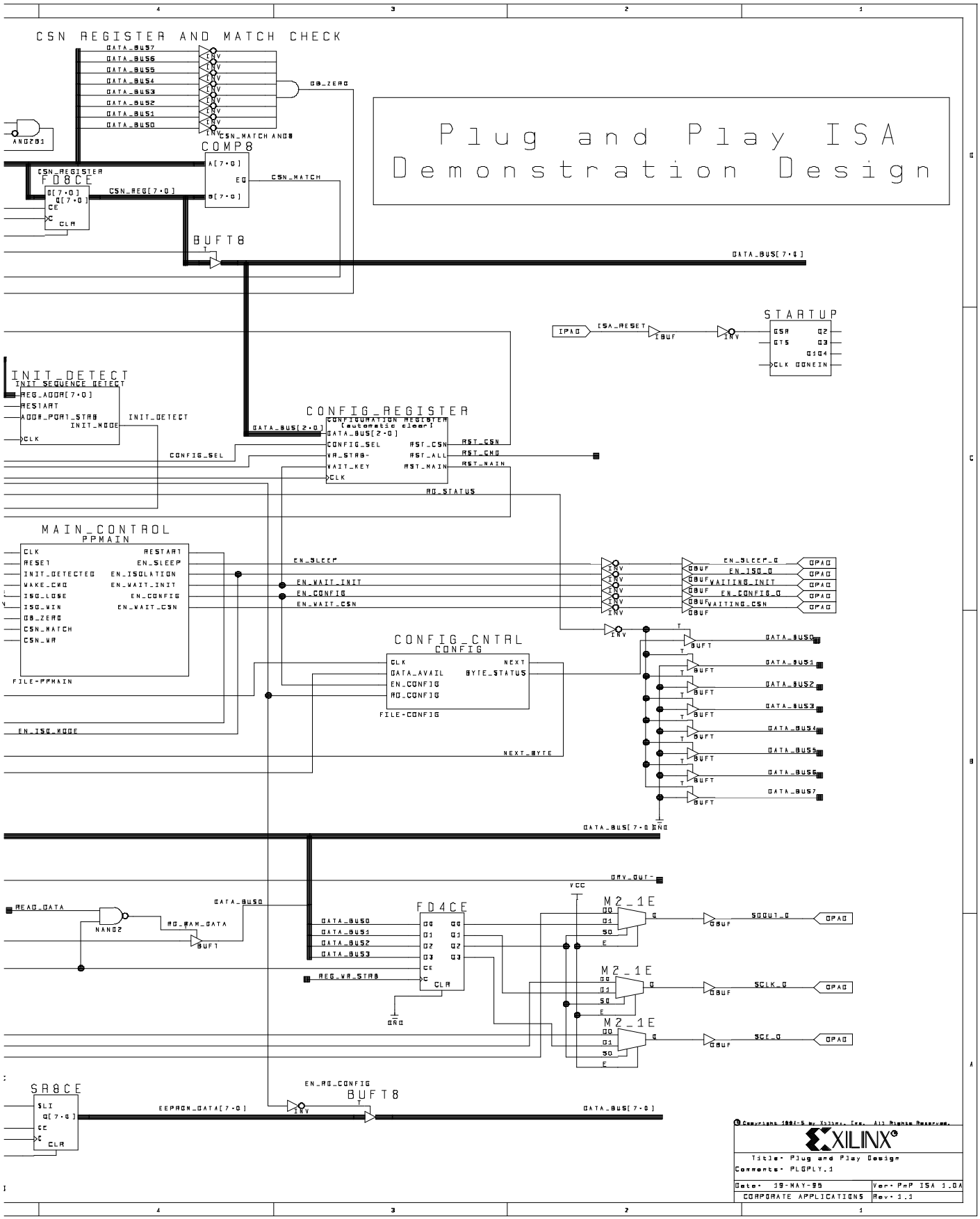


Figure 7—Plug and Play Demonstration Design, Sheet 1 (PLGPLY.1)


CSN REGISTER AND MATCH CHECK
Plug and Play ISA
Demonstration Design
COMP8
FD8CE
BUFT8
STARTUP
INIT_DETECT
CONFIG_REGISTER
MAIN_CONTROL
PPMAIN
CONFIG_CNTRL
CONFIG
FD4CE
M2_1E
SR8CE
FILE-PPMAIN
FILE-CONFIG
Title- Plug and Play Design
Comments- PLGPLY.1
Date- 19-MAY-95
Ver- PnP ISA 1.0A
CORPORATE APPLICATIONS
Rev- 1.1

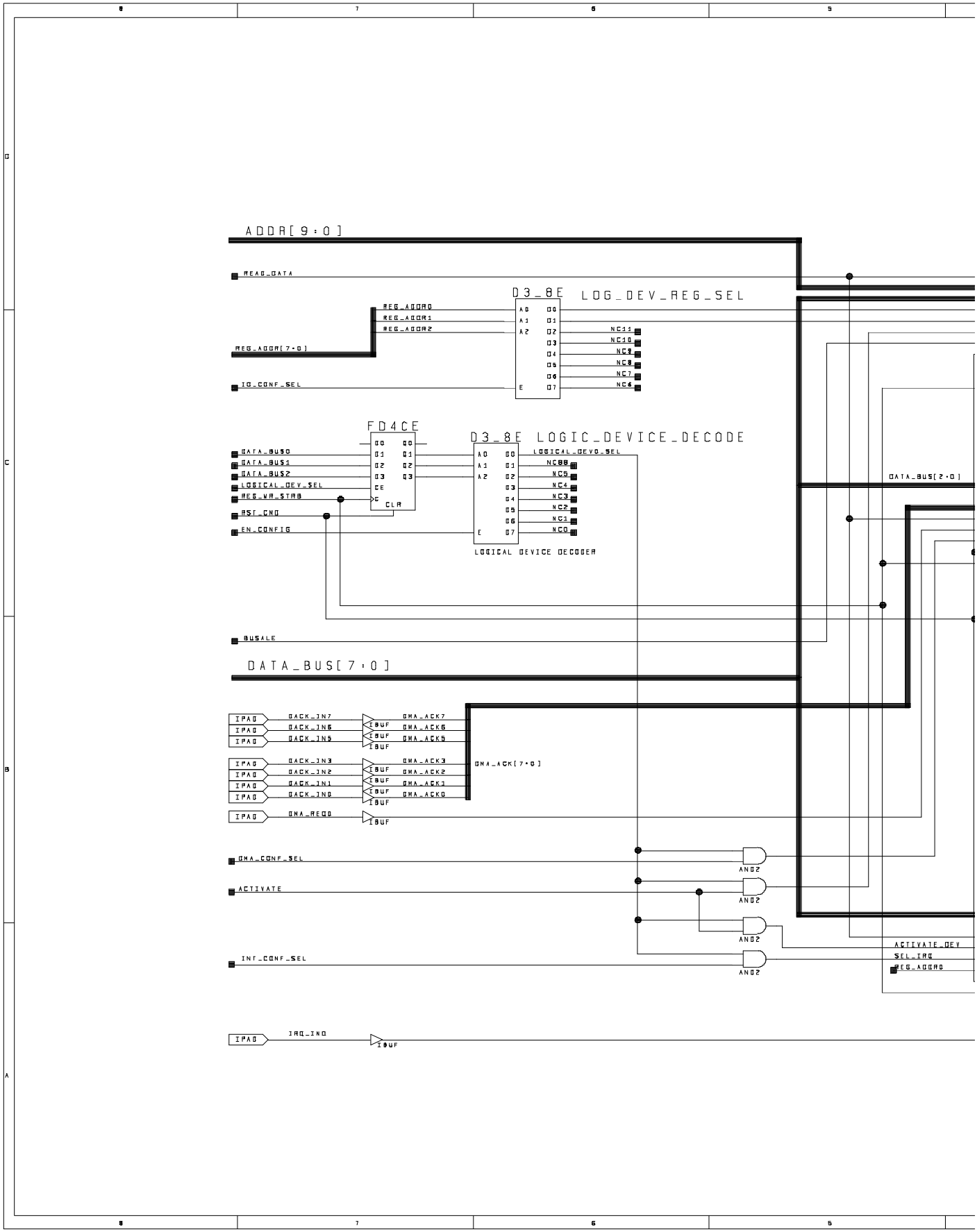
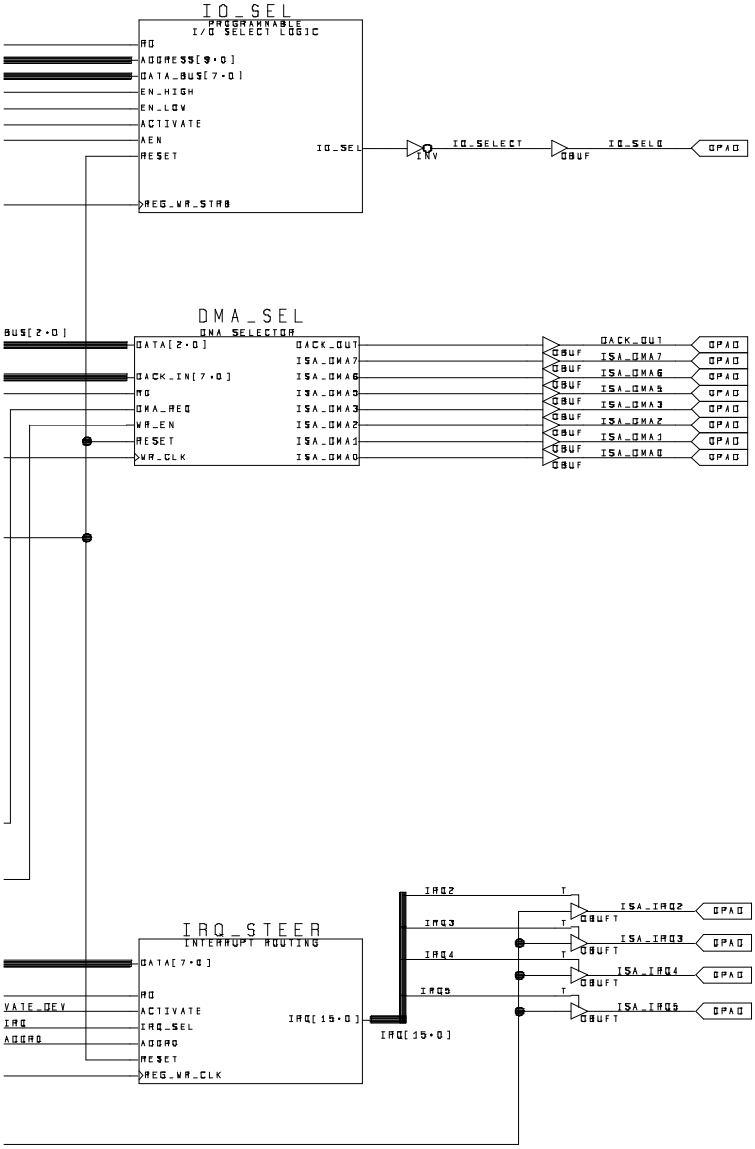


Figure 8—Plug and Play Demonstration Design, Sheet 2 (PLGPLY.2)



XILINX®

Title: Plug and Play Design (Sheet 2)

Comments: PLGPLY.2

Date: 19-MAY-95	Ver: PnP ISA 1.0A
CORPORATE APPLICATIONS	Rev: 1.1

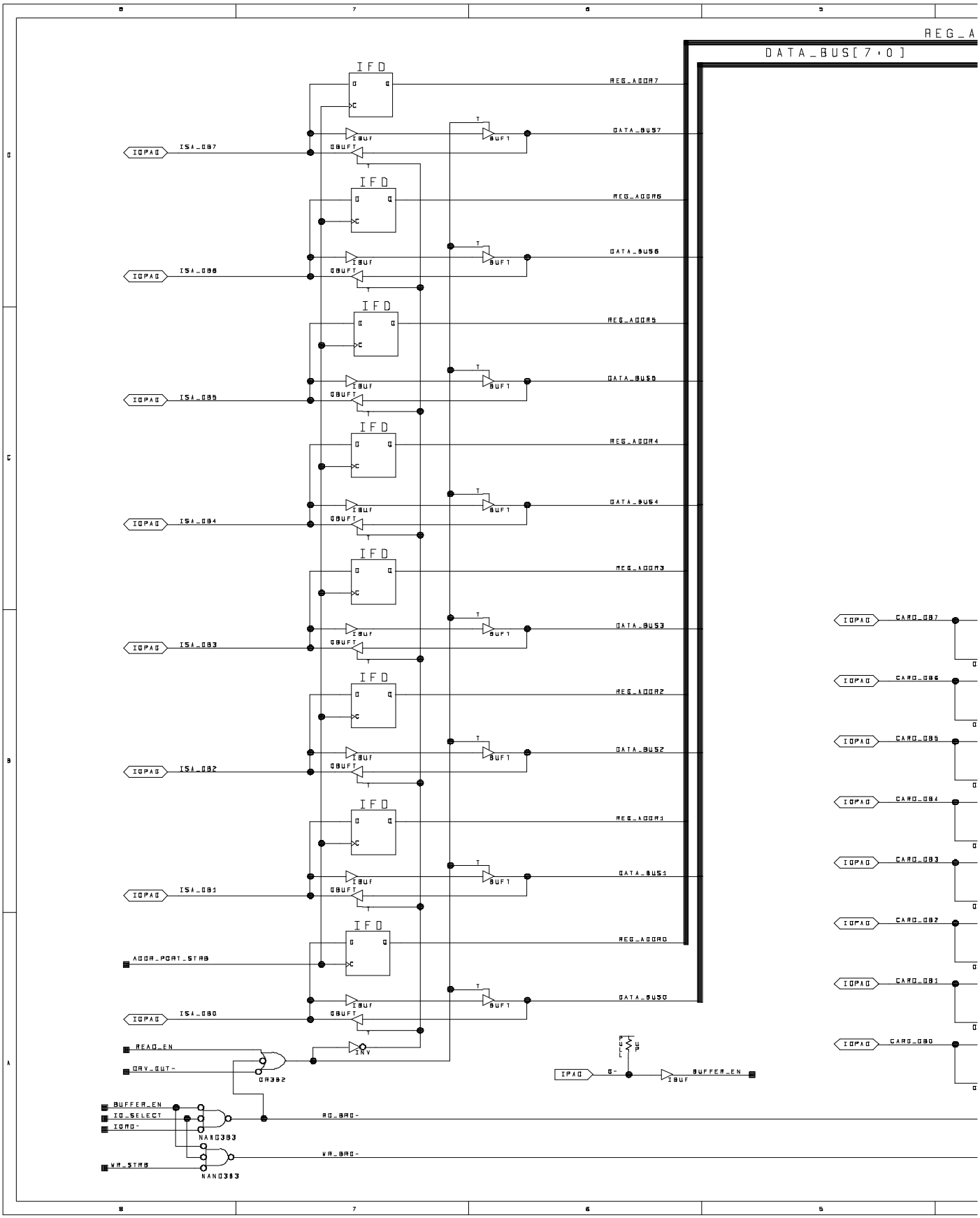
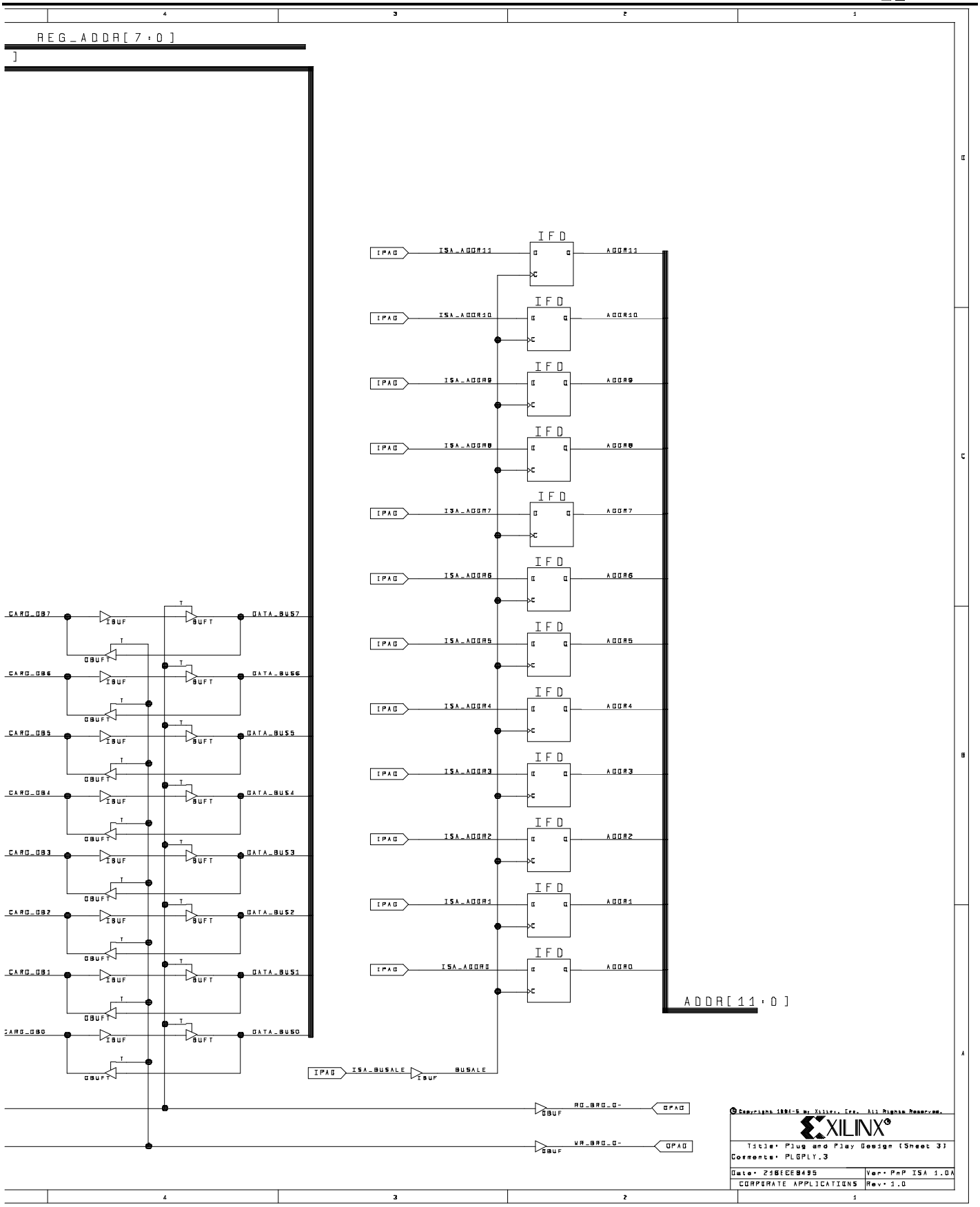


Figure 9—Plug and Play Demonstration Design, Sheet 3 (PLGPLY.3)


REG_ADDR[7:0]
]
IFD
IPAD
ISA_ADDR11
ADDR11
ISA_ADDR10
ADDR10
ISA_ADDR9
ADDR9
ISA_ADDR8
ADDR8
ISA_ADDR7
ADDR7
ISA_ADDR6
ADDR6
ISA_ADDR5
ADDR5
ISA_ADDR4
ADDR4
ISA_ADDR3
ADDR3
ISA_ADDR2
ADDR2
ISA_ADDR1
ADDR1
ISA_ADDR0
ADDR0
ADDR[11:0]
CARD_D87
CARD_D86
CARD_D85
CARD_D84
CARD_D83
CARD_D82
CARD_D81
CARD_D80
IBUF
OBUFT
DATA_BUS7
DATA_BUS6
DATA_BUS5
DATA_BUS4
DATA_BUS3
DATA_BUS2
DATA_BUS1
DATA_BUS0
ISA_BUSALE
BUSALE
RD_BRD_0-
WR_BRD_0-
OPAD
Title: Plug and Play Design (Sheet 3)
Comments: PLGPLY.3
Date: 2?6EC8495
Ver: PnP ISA 1.0A
CORPORATE APPLICATIONS
Rev: 1.0

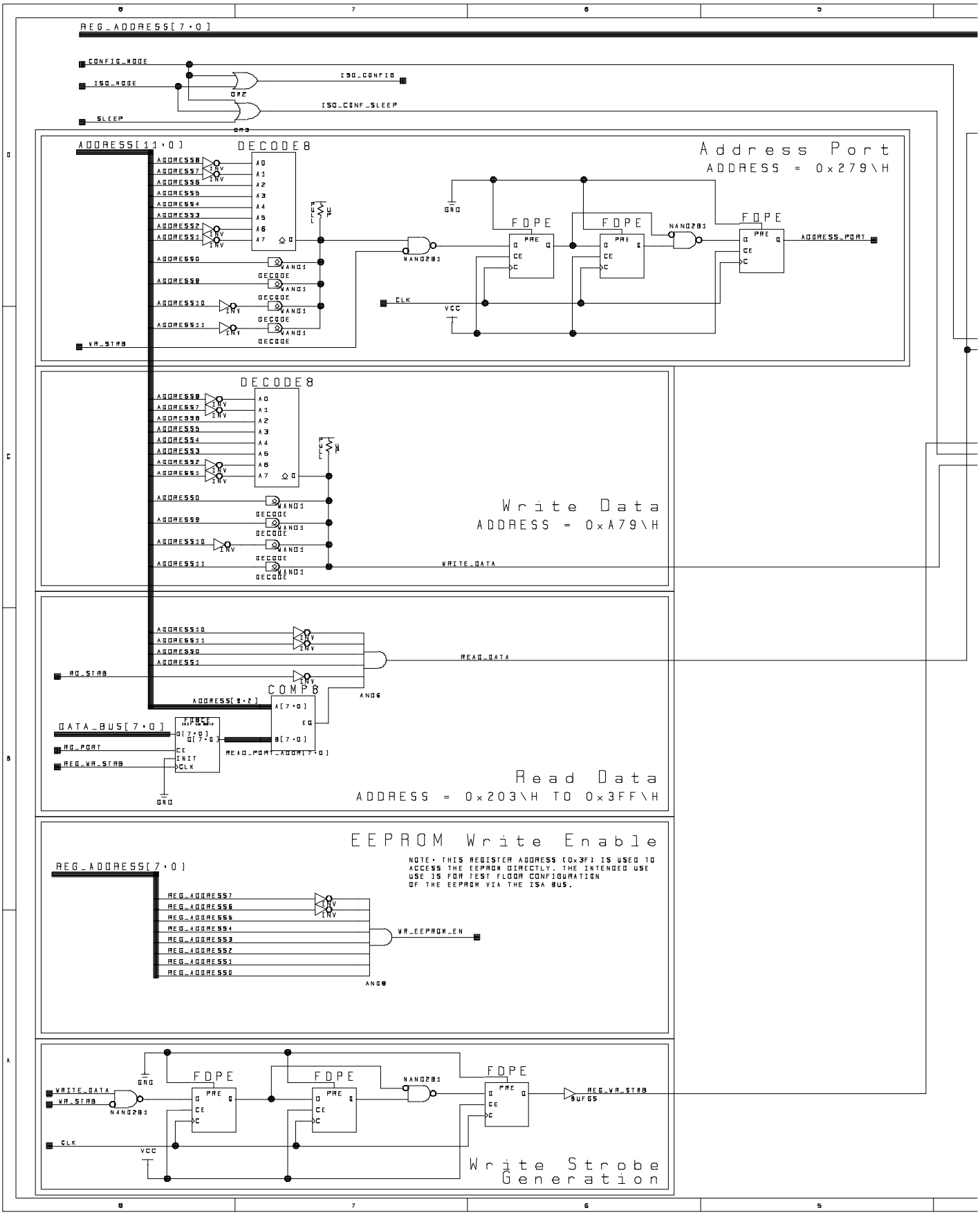
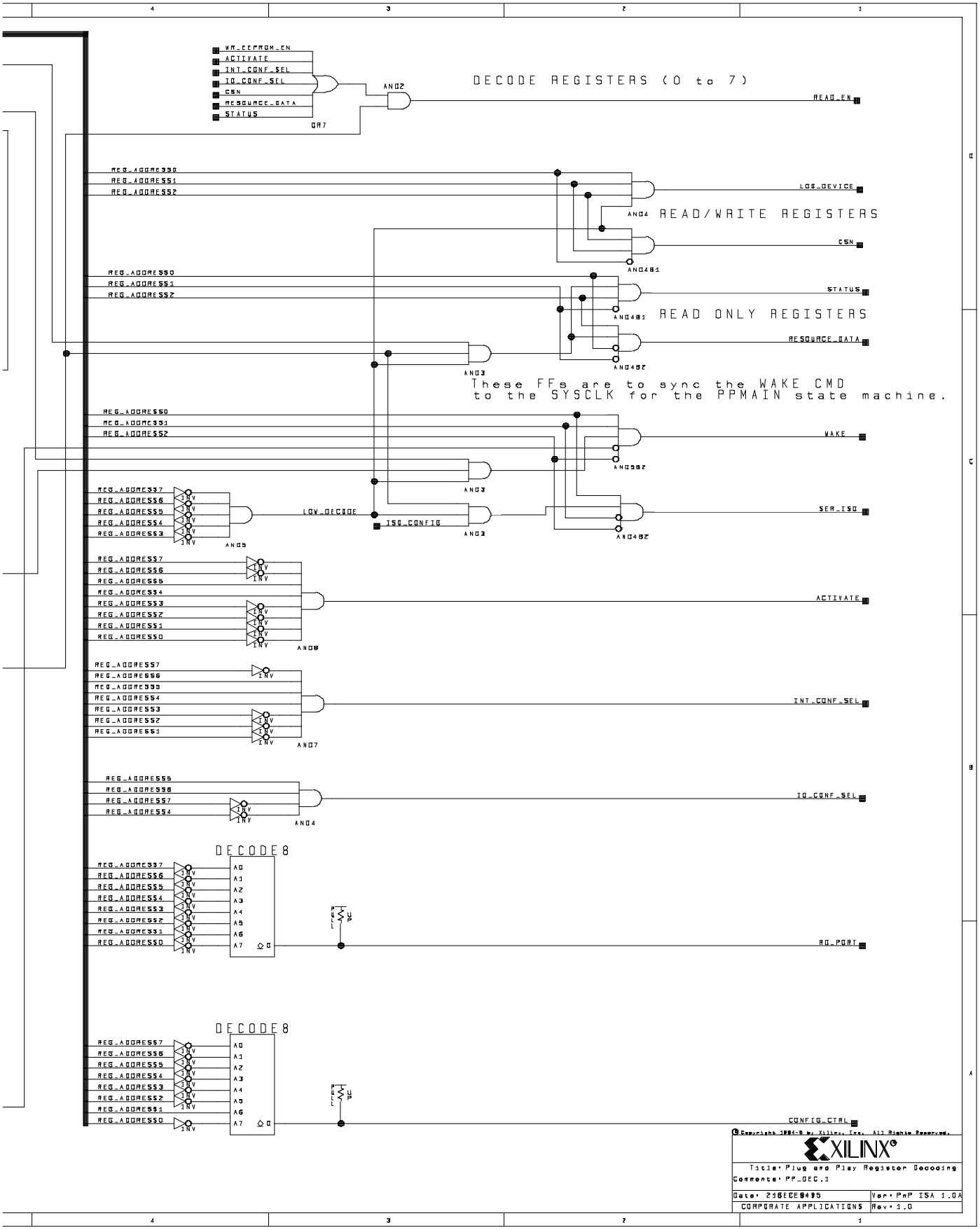


Figure 10—Plug and Play Ports (PP_DEC.1)


DECODE REGISTERS (0 to 7)
READ_EN
LOG_DEVICE
READ/WRITE REGISTERS
CSN
STATUS
READ ONLY REGISTERS
RESOURCE_DATA
These FFs are to sync the WAKE CMD
to the SYSCLK for the PPMAIN state machine.
WAKE
SER_ISO
ACTIVATE
INT_CONF_SEL
IO_CONF_SEL
DECODE8
IO_PORT
DECODE8
CONFIG_CTRL
Title: Plug and Play Register Decoding
Comments: PP_DEC.1
Date: 21ECE9495
Ver: PnP ISA 1.0A
CORPORATE APPLICATIONS
Rev: 1.0

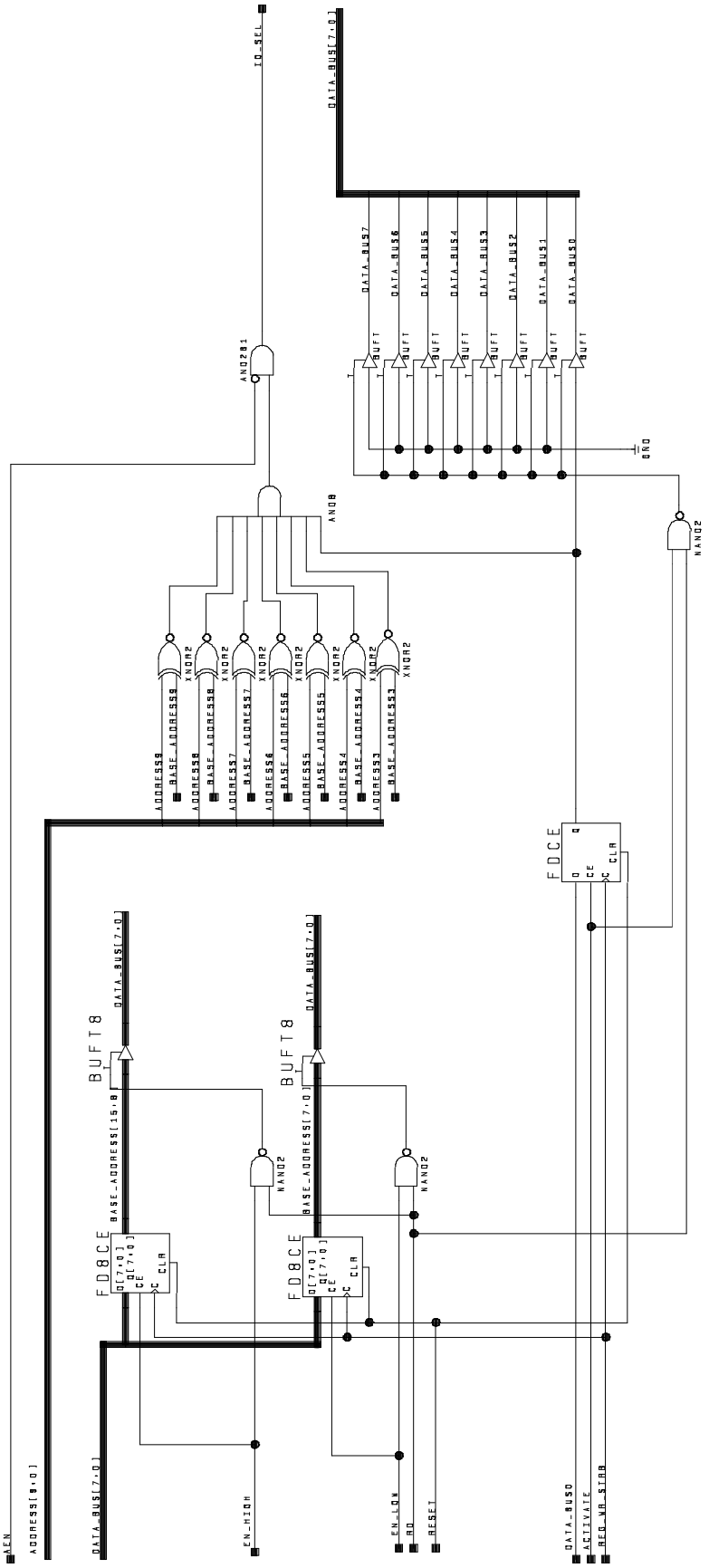


Figure 11—I/O Range Selector Schematic (IO_DEC.1)

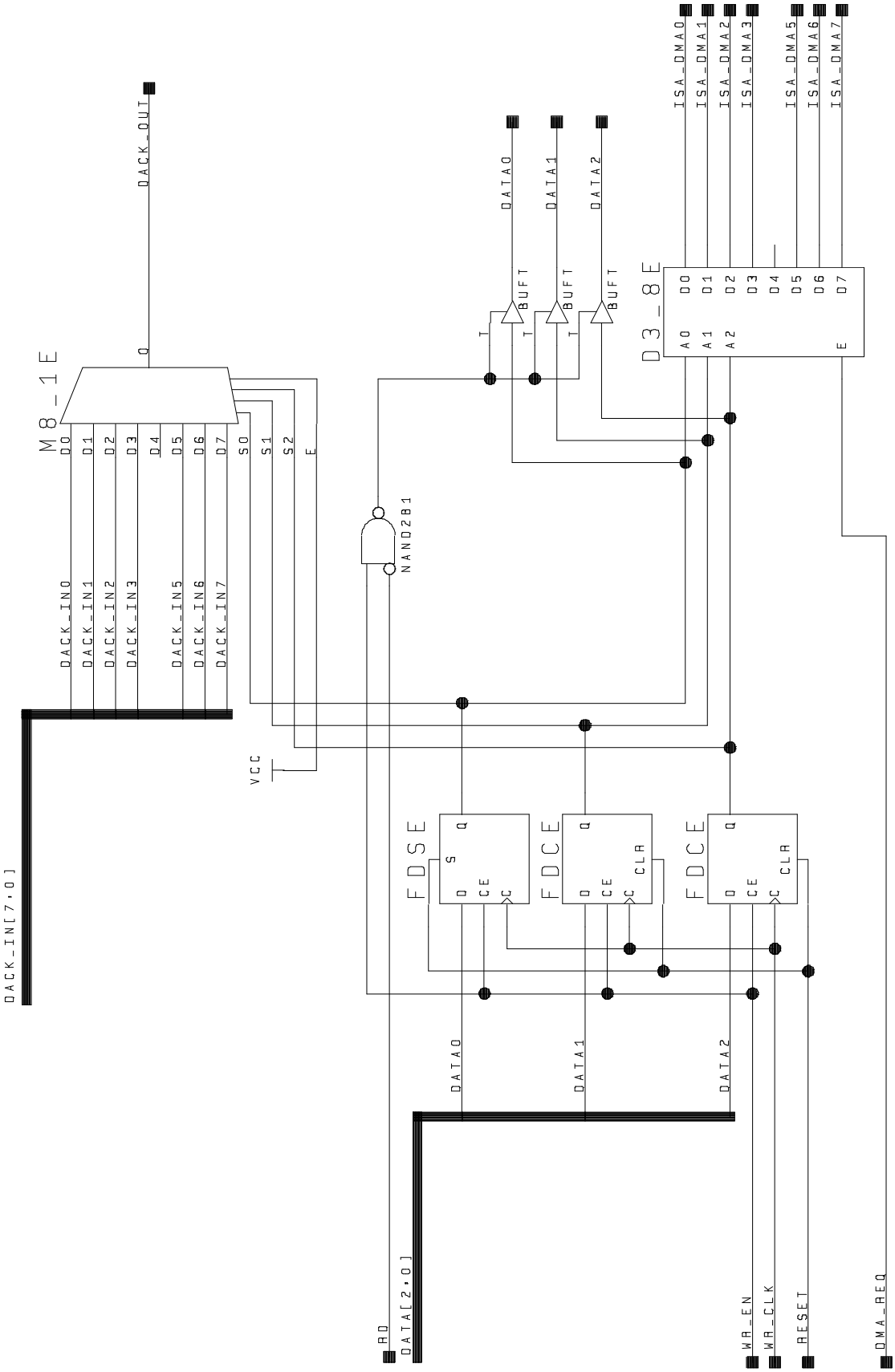


Figure 12—DMA Selector Schematic (DMA_SEL.1)

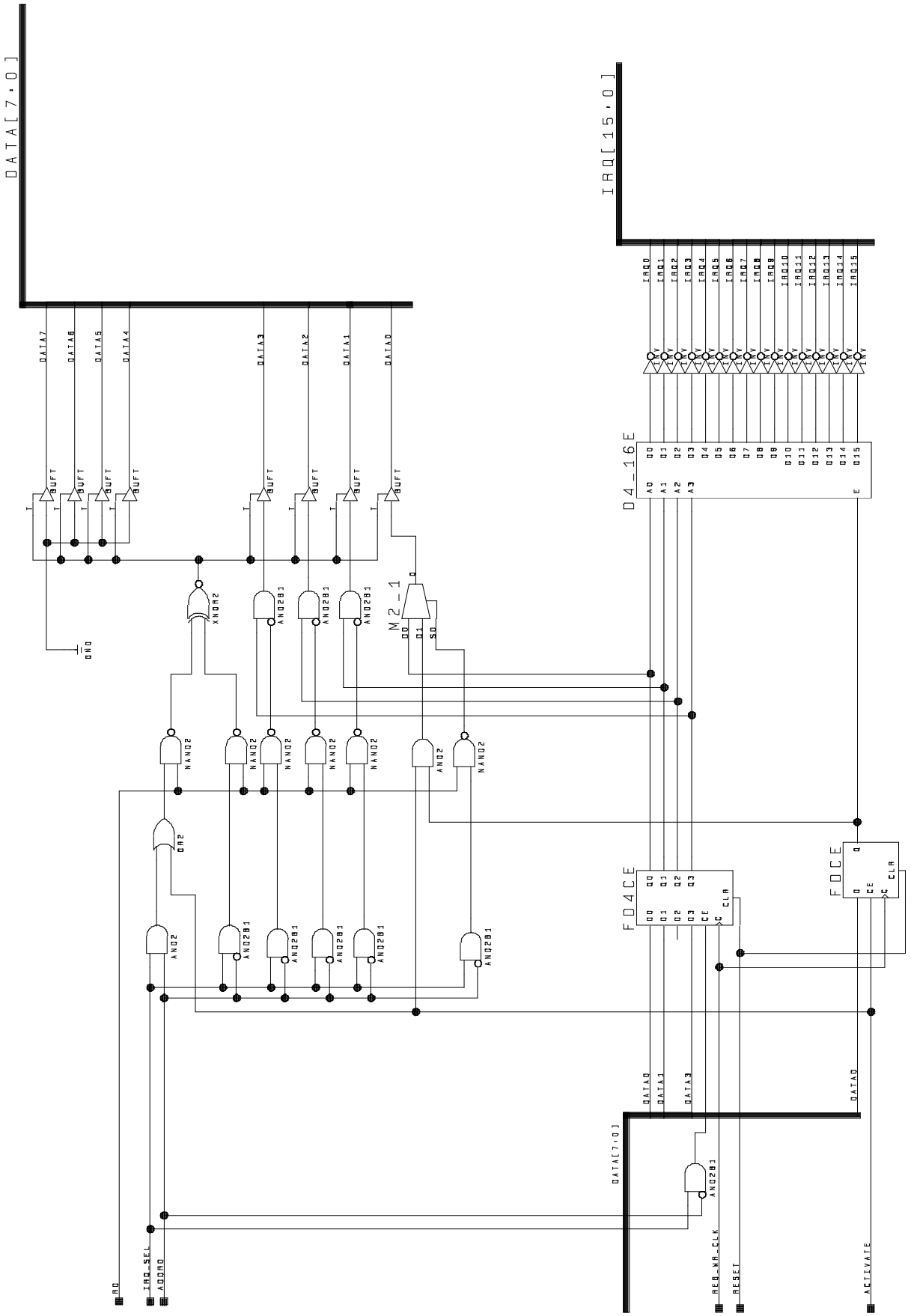


Figure 13—Interrupt Steering Schematic (IRQ_SEL.1)

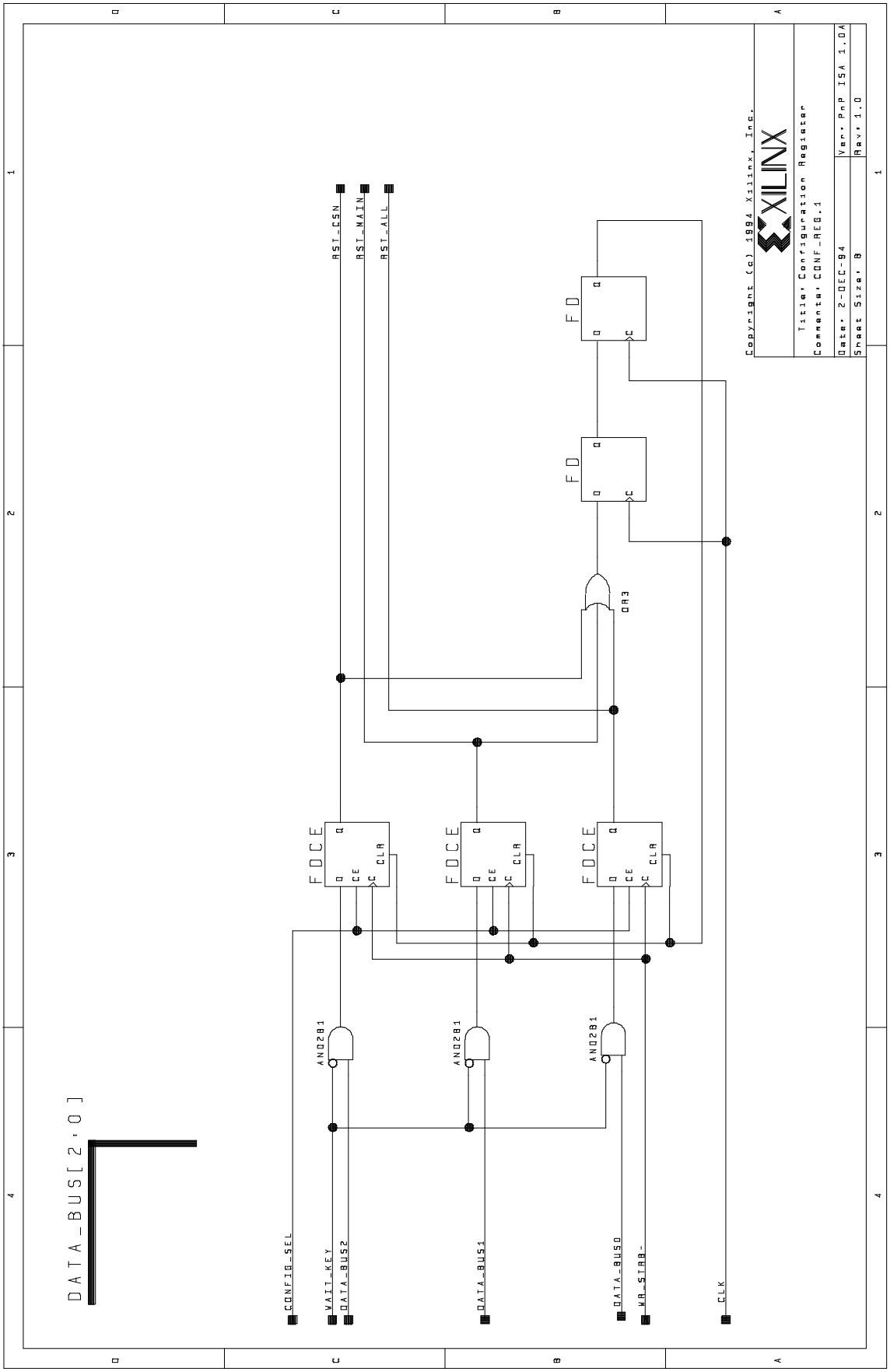


Figure 14—Configuration Register Schematic (CONF_REG.1)

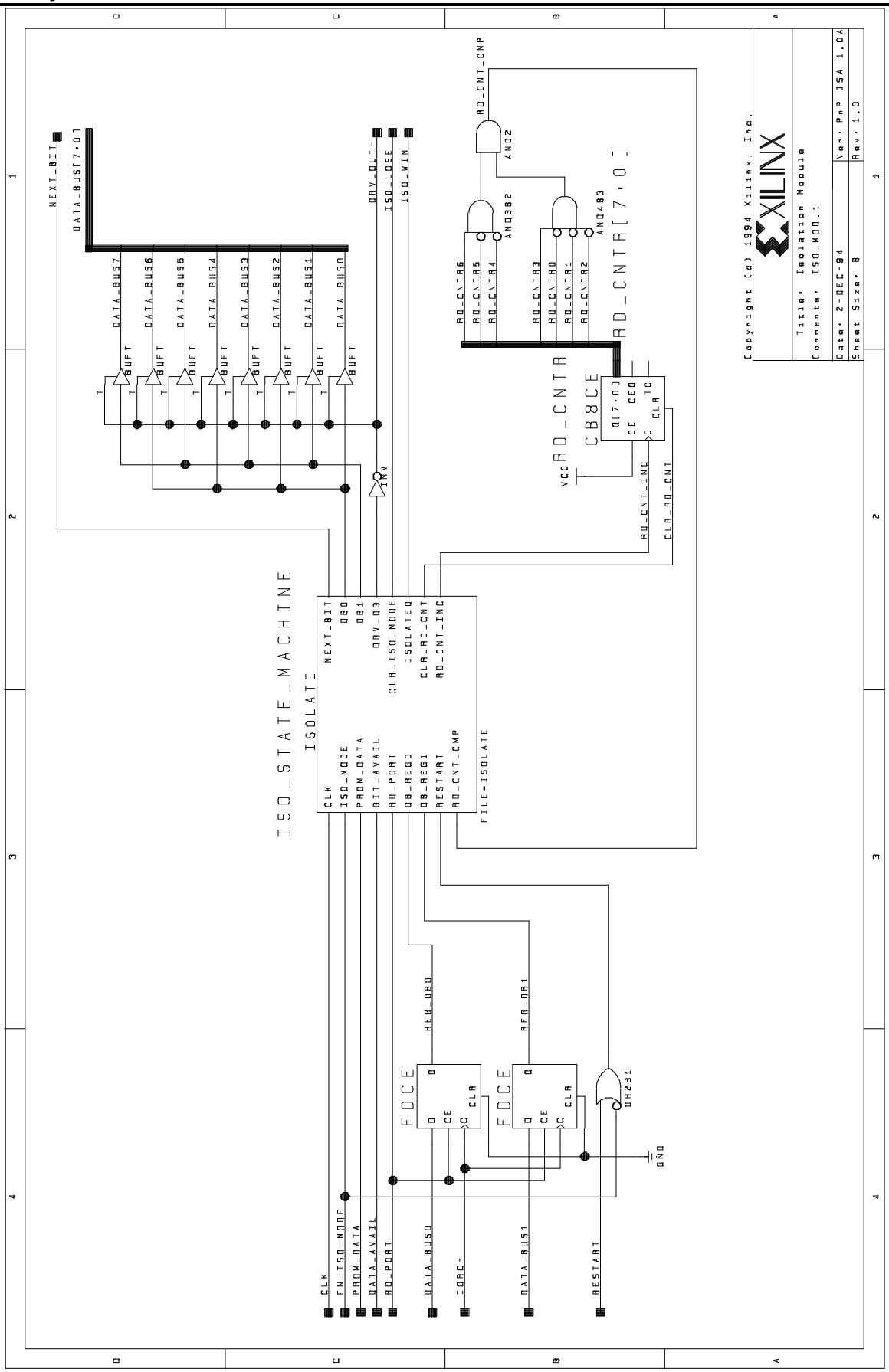


Figure 15—Isolation Protocol Schematic (ISO_MOD.1)

Figure 16—Isolation State Machine Listing (ISOLATE.ABL)

```
" Copyright (c) 1994 Xilinx, Inc.

module isolate
title 'Plug & Play isolation state machine'

" clock
     clk           pin;

" inputs
     iso_mode      pin;    "card in isolation mode

     prom_data     pin;    "ID data one bit at a time
     bit_avail     pin;    "bit available from PROM

     rd_port       pin;    "ISA I/O port read to isolation address

     db_reg0       pin;    "Registered databus bit 0
     db_reg1       pin;    "Registered databus bit 1

     rd_cnt_cmp    pin;    "when 72 isolation reads completed goes true

     restart       pin;    "signal used to force back to state s0

" outputs
     next_bit      pin istype 'reg_D'; "requests another bit of data from PROM

     db0           pin istype 'reg_D'; "output data 0
     db1           pin istype 'reg_D'; "output data 1
     drv_db        pin istype 'reg_D'; "output enable for data bus driver

     clr_iso_mode  pin istype 'reg_D'; "clears the register that indicates in isolation mode
     clr_rd_cnt    pin istype 'reg_D'; "isolation read count clear signal
     rd_cnt_inc    pin istype 'reg_D'; "increment the isolation read counter

     isolated      pin istype 'reg_D'; "sucess -- 72 isolation reads done


" state diagram label definitions
     sbit          STATE_REGISTER istype 'reg_D';
     s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12     STATE;

" The power-on initial state is 's0'
xilinx property 'Initialstate s0';

" label definitions
     db_in           =[db_reg1, db_reg0];
     db_out          =[db1, db0];

     HIGH = 1;
     LOW  = 0;

EQUATIONS

     next_bit.clk      = clk;
     db0.clk           = clk;
     db1.clk           = clk;
     drv_db.clk        = clk;
     clr_iso_mode.clk  = clk;
     clr_rd_cnt.clk    = clk;
     rd_cnt_inc.clk    = clk;
     isolated.clk      = clk;
     sbit.clk          = clk;

" STATE MACHINE DESCRIPTION
" -------------------------
" There are two main branches to this state machine, depending on the data
"    from the Serial Identification data in the PROM.
" When the PROM data is HIGH, then the state machine actively drives a
```

```
"    0x55h pattern and then a 0xAAh pattern on the bus.  Then the state
"    machine increments the isolation read counter and the isolation
"    sequence continues.
" When the PROM data is LOW, then the state machine monitors the data bus.
"    If this card detects another card driving the bus with 0x55h followed
"    by 0xAAh, then this card drops out of the current isolation round (but
"    it will compete in the next round).  If no other card is detected
"    driving the bus, then the read counter is incremented and the sequence
"    contiues.
"
" Sucessful isolation is indicated by the ISOLATED signal going high.
"

     State_Diagram sbit

     ASYNC_RESET      s0:   restart;

" Wait for the card to be in ISOLATION mode.
" ---------------------------------------
     state s0:       if (!iso_mode) then s0
                     else s1 with
                        next_bit := HIGH;
                        endwith;

" Wait for data to be available from PROM.
" --------------------------------------
     state s1:       if (!bit_avail) then s1 with
                        next_bit := HIGH;
                     else s2;

" Wait for the system to read the PROM data.  Jump to the appropriate loop
"    in the state machine depending on whether the PROM data is HIGH or LOW.
"    If HIGH, then jump to state 's3' while driving 0x55h on the bus.
"    Otherwise, jump to state 's6'.
" ------------------------------------------------------------------------
     state s2:       if (!rd_port) then s2
                     else if (rd_port & prom_data) then s3 with
                        drv_db := HIGH;
                        db_out := ^b01;  " drive the bus with 0x55h
                        endwith;
                     else if (rd_port & !prom_data) then s6


" ============================
" SERIAL IDENTIFIER DATA IS HIGH
" ============================
" WAIT for read cycle to end and continue to drive the bus with 0x55h.
" ------------------------------------------------------------------
     state s3:       if (rd_port) then s3 with
                        drv_db := HIGH;
                        db_out := ^b01;  " drive bus with 0x55h
                        endwith;
                     else s4;

" WAIT for next read cycle but now drive the bus with 0xAAh.
" ---------------------------------------------------------
     state s4:       if (!rd_port) then s4
                     else s5 with
                        drv_db := HIGH;
                        db_out := ^b10;
                        endwith;

" Keep driving the bus with 0xAAh until the end of the read cycle.  At the
" end of the cycle, increment the read counter and proceed to state 's11'.
" -----------------------------------------------------------------------
     state s5:       if (rd_port) then s5 with
                        drv_db := HIGH;
                        db_out := ^b10;
                        endwith;
                     else s11 with
                        rd_cnt_inc := HIGH;
                        endwith;
```

```
" ============================
" SERIAL IDENTIFIER DATA IS LOW
" ============================
" Check if another card is driving the bus with 0x55h data.  If there is a
"    card driving 0x55h, then goto state 's7' and begin the check for 0xAAh.
"    If no other card is driving 0x55h, then goto state 's9' to complete
"    this round.
" ------------------------------------------------------------------------
     state s6:       if (rd_port) then s6
                     else if (!rd_port & db_reg0 & !db_reg1) then s7
                     else s9;

" WAIT for next read cycle.
" ------------------------
     state s7:       if (!rd_port) then s7
                     else s8;

" Check if another card is driving the bus with 0xAAh data.  If there is a
"    card driving 0xAAh, then drop out of the this isolation round and go
"    back to state 's0'.
" If no other card is driving 0xAAh, then goto state 's11' to complete
"    this round.
" ------------------------------------------------------------------------
     state s8:       if (rd_port) then s8
                     else if (!rd_port & !db_reg0 & db_reg1) then s0 with
                        clr_rd_cnt := HIGH;
                        clr_iso_mode := HIGH;
                        endwith;
                     else s11 with
                        rd_cnt_inc := HIGH;
                        endwith;

" WAIT for next read cycle.  The data bus is not being driven with 0x55h.
" ----------------------------------------------------------------------
     state s9:       if (!rd_port) then s9
                     else s10;

" WAIT for read cycle to end, then get next bit from PROM.
" --------------------------------------------------
     state s10:      if (rd_port) then s10
                     else s11 with
                        rd_cnt_inc := HIGH;
                        endwith;


" =================
" CHECK IF ISOLATED
" =================
" Check to see if all 72 serial identifier reads have occured.  If so, then
"    this card has been isolated.  Otherwise, continue the isolation process.
" ------------------------------------------------------------------------
     state s11:      if (rd_cnt_cmp) then s11 with
                        isolated := HIGH;
                        endwith;
                     else s12 with
                        next_bit := HIGH;
                        endwith;

" This state synchronizes the EEPROM controller's data_avail signal to this
"    this state machine.
" ------------------------------------------------------------------------
     state s12:      if (bit_avail) then s12 with
                        next_bit := HIGH;
                        endwith;
                     else s1 with
                        next_bit := HIGH;
                        endwith;

end isolate
```

Figure 17—ABEL source file for Plug and Play main state machine (PPMAIN.ABL).

```
" Copyright (c) 1994 Xilinx, Inc.

module ppmain
title 'Plug & Play main control state machine';

" CLOCK
     clk             pin;

" INPUTS
     reset           pin;
     init_detected   pin;
     wake_cmd        pin;
     iso_lose        pin;
     iso_win         pin;
     db_zero         pin;
     csn_match       pin;
     csn_wr          pin;

" OUTPUTS
     restart         pin istype 'reg_D';

     en_sleep        pin;
     en_isolation    pin;
     en_wait_init    pin;
     en_config       pin;
     en_wait_csn     pin;

" STATE DEFINITIONS
     states                  STATE_REGISTER istype 'reg_D';

     wait_init,
     sleep,
     iso_mode,
     wait_csn,
     config     state;

" EQUATES
     HIGH = 1;
     LOW  = 0;

" Set power-up initial state to 'wait_init'
Xilinx Property 'InitialState wait_init';

EQUATIONS
     states.clk      = clk;
     restart.clk     = clk;

     en_wait_init    = wait_init;
     en_sleep        = sleep;
     en_isolation    = iso_mode;
     en_wait_csn     = wait_csn;
     en_config       = config;

     State_Diagram   states
     ASYNC_RESET     wait_init:      reset;

" STATE:  WAIT FOR INITIATION KEY (wait_init)
" ----------------------------------------
" Plug and Play cards wake up in a queiscent state and wait for an initiation
"   key before becoming active.  Once the initiated, the cards transition to
"   the SLEEP state.

     state wait_init:     if (!init_detected) then wait_init
             else sleep;
```

```
" STATE: SLEEP (sleep)
" --------------------
" A card wakes up with its Card Select Number (CSN) set to zero.  Once a card
"   is uniquely isolated, it is assigned a non-zero CSN identifier.
" Cards that have not been isolated will enter ISOLATION when the software
"   sends a WAKE[CSN] command with CSN=0.
" A card that has been isolated (i.e. CSN > 0) will enter the CONFIGURATION
"   state when the software sends a WAKE[CSN] command where CSN matches the
"   cards unique CSN.
" All other cards will remain in SLEEP.

     state sleep:         if (wake_cmd & db_zero & csn_match) then iso_mode with
            restart := HIGH;
            endwith;
        else if (wake_cmd & !db_zero & csn_match) then config with
            restart := HIGH;
            endwith;
        else sleep;


" STATE:  ISOLATION
" -----------------
" Unisolated cards compete to win the isolation process using their serial
"   identification data.
" If a card loses during the current isolation round, it then goes back to
"   the SLEEP state.
" If a card wins, it goes to WAIT FOR CSN.
" If no winner is yet assigned, the card competes in the next isolation round.

     state iso_mode:      if (iso_lose) then sleep with
            restart := HIGH;
            endwith;
        else if (iso_win) then wait_csn
        else iso_mode;


" STATE:  WAIT FOR CSN
" --------------------
" A uniquely isolated card waits for the software to assign the card its
"   unique Card Selection Number.
" After the CSN is written, the card enters the CONFIGURATION state.

     state wait_csn:      if (csn_wr) then config
        else wait_csn;


" STATE:  CONFIGURATION
" ---------------------
" A card remains in the CONFIGURATION state until the software wishes to
"   wake another card (i.e. WAKE[CSN] with CSN <> card's CSN).  When another
"   is awakened, this card will go back into the SLEEP state because only
"   one card can be in the CONFIGURATION state.

     state config:        if (wake_cmd & !csn_match) then sleep
        else config;

end ppmain;
```

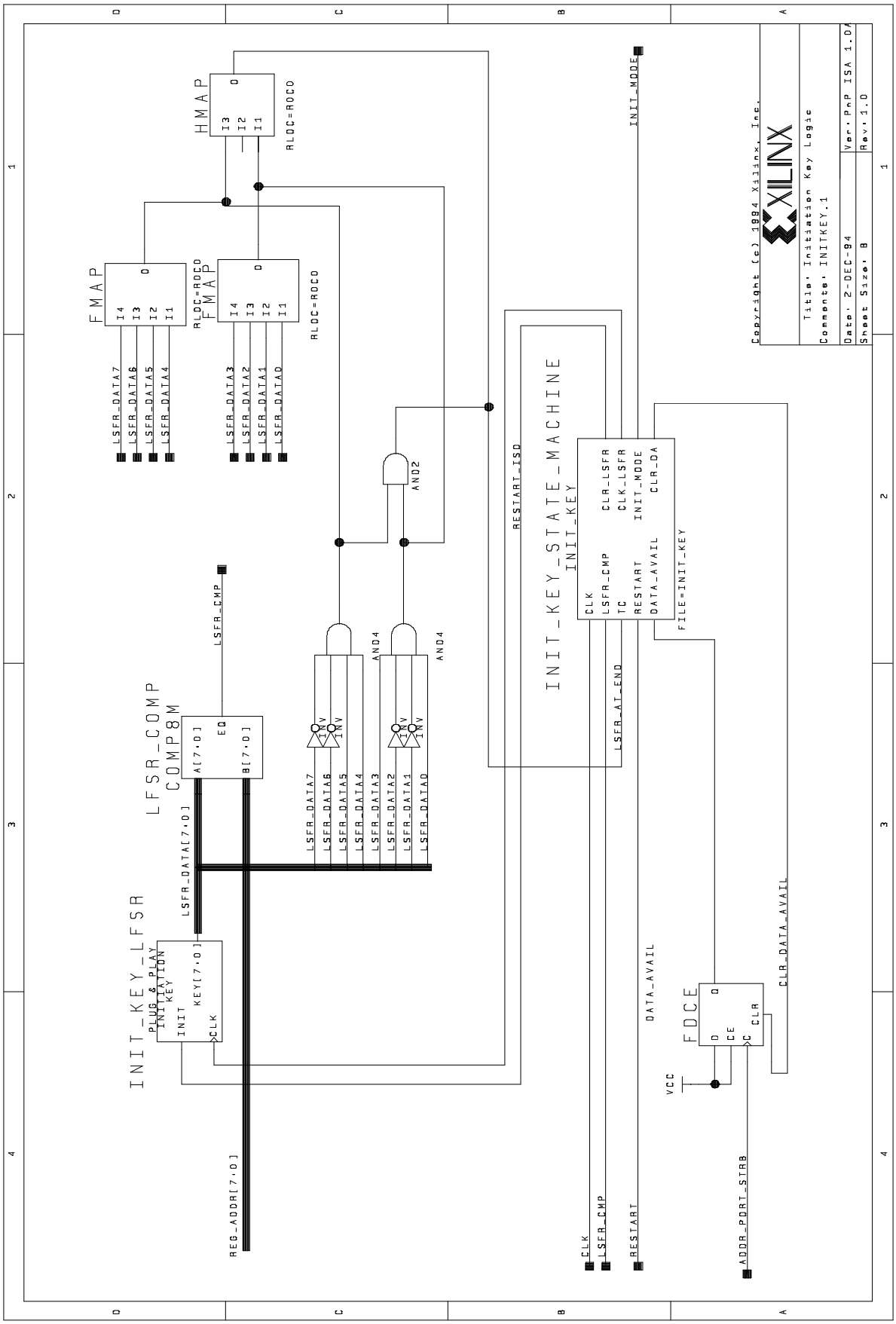


Figure 18—Initiation Key Schematic (INITKEY.1)

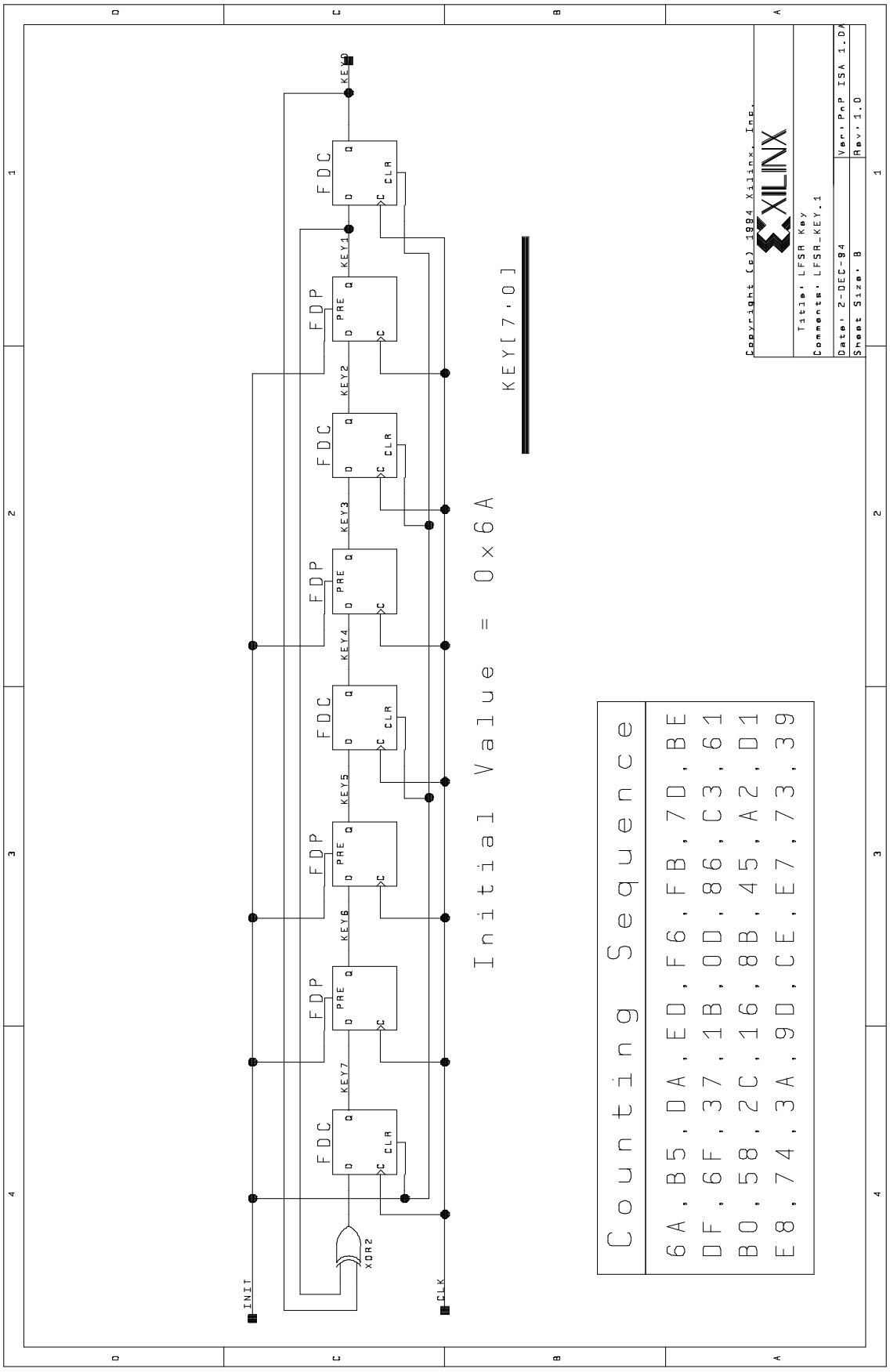


Figure 19—Initiation Key LFSR Schematic (LFSR_KEY.1)

Figure 20—ABEL source file for Initiation Key state machine (INIT_KEY.ABL)

```
" Copyright (c) 1994 Xilinx, Inc.

module init_key
title 'Plug & Play initiation key state machine'

" clock
  clk            pin;

" inputs
  lsfr_cmp       pin;   "Compare between LFSR data and data bus
  tc             pin;   "Terminal count from LFSR
  restart        pin;   "Restart initiation process
  data_avail     pin;   "Data is available

" outputs
  clr_da          pin istype 'reg_D'; "Clear data available
  clr_lsfr        pin istype 'reg_D'; "Put LFSR in initial state
  clk_lsfr        pin istype 'reg_D'; "Increment the LFSR
  init_mode       pin istype 'reg_D'; "Indicates INITATION complete

" State diagram declaration and assignment
  sbit            STATE_REGISTER istype 'reg_D';
  s0, s1, s2, s3  STATE;

" The power-on initial state is 's0'.
  xilinx property 'Initialstate s0';

EQUATIONS


" STATE MACHINE DEFINITION
  sbit.clk      = clk;
  clr_da.clk    = clk;
  clr_lsfr.clk  = clk;
  clk_lsfr.clk  = clk;
  init_mode.clk = clk;

State_diagram sbit

  ASYNC_RESET     s0:     restart;

" Wait for data to become available and the proceed to state 's1'.
" ------------------------------------------------------------
  State s0:       if (!data_avail) then s0
       else    s1;

" If the initiation key data sent by the Plug and Play software matches the
"    LFSR data, then continue.  Otherwise, clear the LFSR and start over.
" ------------------------------------------------------------------------
  State s1:         if (lsfr_cmp) then s2
                    else s0 with
                      clr_lsfr := 1;
                      clr_da := 1;
                      endwith;

" If the LFSR has reached its terminal count, then continue to state 's3'.
"    Otherwise, go back to state 's0' and wait for net byte of initiation
"    key data.
" ------------------------------------------------------------------------
  State s2:         if (tc) then s3 with
                      init_mode := 1;
                      clr_da := 1;
                      endwith;
                    else s0 with
                      clk_lsfr := 1;
                      clr_da := 1;
                      endwith;
```

```
" Once the entire initation key has been read and verified, indicate that
"    the card is now in INITIATION mode.
" -------------------------------------------------------------------------
  State s3:     if (!data_avail) then s3 with
                   init_mode := 1;
                   clr_lsfr := 1;
                   endwith;
                else s0;

end init_key
```

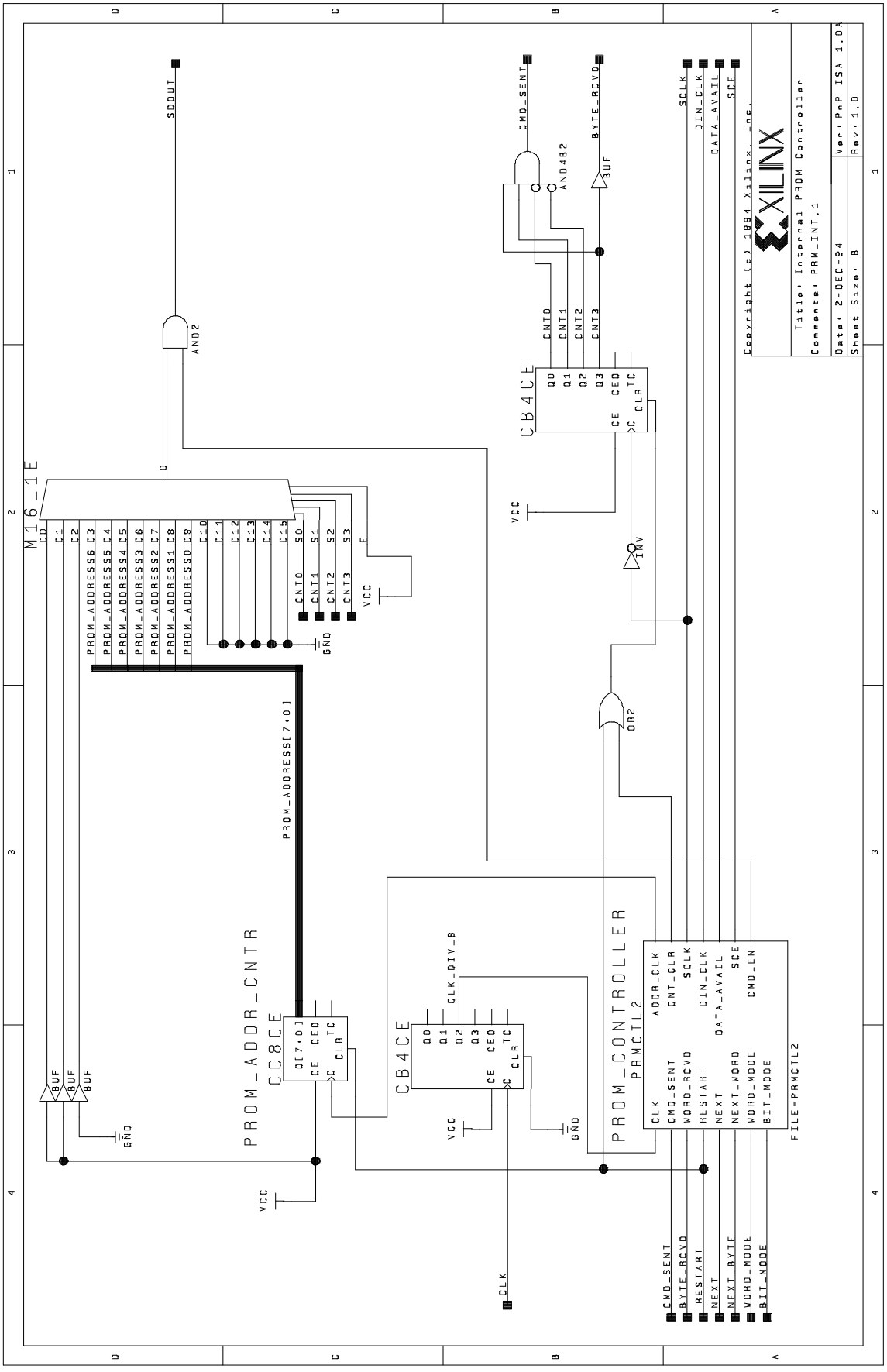


Figure 21.—EEPROM control logic.

Figure 22.—ABEL source file for the EEPROM control state machine (PRMCTL2.ABL).

```
module prmctl2
title 'EEPROM read control'

" This state machine is used to control access to an industry-standard EEPROM.
" It transmits a read command to the EEPROM and then manages the receipt of the
" serial data stream.  The EEPROM data can be returned to the system in either
" a bit at a time or byte.

"clock
        clk             pin;

"inputs
        cmd_sent        pin;
        word_rcvd       pin;

        restart         pin;
        next            pin;
        next_word       pin;

        word_mode       pin;
        bit_mode        pin;

"outputs
        addr_clk        pin istype 'reg_D';
        cnt_clr         pin istype 'reg_D';
        sclk            pin istype 'reg_D';
        din_clk         pin istype 'reg_D';
        data_avail      pin istype 'reg_D';
        sce             pin istype 'reg_D';
        cmd_en          pin istype 'reg_D';

        HIGH, LOW = 1,0;

" state diagram declarations

        sbit                                    STATE_REGISTER istype 'reg_D';
        s0, s1, s2, s3, s4, s5, s6, s7, s8, s9  state;

xilinx property 'InitialState s0';

EQUATIONS
        sbit.clk        = clk;
        addr_clk.clk    = clk;
        cnt_clr.clk     = clk;
        sclk.clk        = clk;
        din_clk.clk     = clk;
        data_avail.clk  = clk;
        sce.clk         = clk;
        cmd_en.clk      = clk;
"
" STATE MACHINE FOR INTERFACING TO SERIAL EEPROM
"
" The serial EEPROM requires that a read command be sent with the word address
" imbedded. The states s1 through s4 control external logic to transfer a read
" command to the EEPROM.
"
" The states s5 through s7 are used to read back the serial data and indicate
" that a bit is available from the EEPROM.  The state machine detects when the
" entire 16 bits has been read from the EEPROM and when the next read command
" must be sent.

        State_Diagram sbit

        ASYNC_RESET     s0:     restart;

        " SEND THE READ COMMAND
        state s0:      if (next & bit_mode) then s2 with
```

```
                        sce := HIGH;
                        cmd_en := HIGH;
                        endwith;
                else if (next_word & word_mode) then s2 with
                        sce := HIGH;
                        cmd_en := HIGH;
                        endwith;
                else s0 with
                        cnt_clr := HIGH;
                        endwith;

state s2:       goto s9 with
                        sclk := HIGH;
                        sce  := HIGH;
                        cmd_en := HIGH;
                        endwith;

state s9:       goto s3 with
                        sce := HIGH;
                        cmd_en := HIGH;
                        endwith;

state s3:       if (!cmd_sent) then s2 with
                        sce := HIGH;
                        cmd_en := HIGH;
                        endwith;
                else s4 with
                        sce     := HIGH;
                        cnt_clr := HIGH;
                        addr_clk := HIGH;
                        endwith;


" READ the data back out of the EEPROM a bit at a time
state s4:      goto s5 with
                        sclk := HIGH;
                        sce  := HIGH;
                        endwith;


state s5:     if (next) then s6 with
                        din_clk := HIGH;
                        sce := HIGH;
                        endwith;
              else if (next_word) then s7 with
                        din_clk := HIGH;
                        sce := HIGH;
                        endwith;

state s6:    if (next) then s6 with
                        data_avail := HIGH;
                        sce := HIGH;
                        endwith;
             else s7 with
                        data_avail := HIGH;
                        sce := HIGH;
                        endwith;

state s7:   if (next_word & !word_rcvd & word_mode) then s4 with
                        sce := HIGH;
                        endwith;
            else if (next_word & word_rcvd & word_mode) then s7 with
                        data_avail := HIGH;
                        endwith;
            else if (!next & bit_mode) then s7 with
                        data_avail := HIGH;
                        sce := HIGH;
                        endwith;
            else if (next & !word_rcvd & bit_mode) then s4 with
                        sce := HIGH;
                        endwith;
            else s0 with
```

```
                                cnt_clr := HIGH;
                                endwith;

        state s8:   if (!next_word & word_mode) then s8 with
                                data_avail := HIGH;
                                endwith;
                    else s0 with
                                cnt_clr := HIGH;
                                endwith;

end
```

North America

Corporate Headquarters

Xilinx, Inc.
2100 Logic Drive
San Jose, CA 95124
U.S.A.

Tel: 1+(408) 559-7778
FAX: 1+(408) 559-7114
Web: **http://www.xilinx.com/**

Northern California

Xilinx, Inc.
Oakmead Pkwy.
Suite 202
Sunnyvale, CA 95051

Tel: (408) 245-9850
FAX: (408) 245-9865

Southern California

Xilinx, Inc.
15615 Alton Parkway
Suite 280
Irvine, CA 92718

Tel: (714) 727-0780
FAX: (714) 727-3128

Colorado

Xilinx, Inc.
5690 DTC Parkway
Suite 490W
Englewood, CO 80111

Tel: (303) 220-7541
FAX: (303) 220-8641

New Hampshire

Xilinx, Inc.
61 Spit Brook Road
Nashua, NH 03060

Tel: (603) 891-1096
FAX: (603) 891-0890

Pennsylvania

Xilinx, Inc.
905 Airport Rd.
Suite 200
West Chester, PA 19380

Tel: (610) 430-3300
FAX: (610) 430-0470

Texas

Xilinx, Inc.
4100 McEwen
Suite 237
Dallas, TX 75244

Tel: (214) 960-1043
FAX: (214) 960-0927

Illinois

Xilinx, Inc.
939 N. Plum Grove Road
Suite H
Schaumburg, IL 60173

Tel: (708) 605-1972
FAX: (708) 605-1976

North Carolina

Xilinx, Inc.
6080-C Six Forks Road
Raleigh, NC 27609

Tel: (919) 846-3922
FAX: (919) 846-8316

Europe

United Kingdom

Xilinx, Ltd.
Suite 1B, Cobb House
Oyster Lane, Byfleet
Surry KT14 7DU
UNITED KINGDOM

Tel: (44) 932-349401
FAX: (44) 932-349499

France

Xilinx Sarl
Espace Jouy Technology
21, rue Albert Calmette, Bt. C
78353 Jouy en Josas Cedex
FRANCE

Tel: (33) 1-34-63-01-01
FAX: (33) 1-34-63-01-09

Germany

Xilinx, GmbH
Dorfstr. 1
85609 Aschheim
München
GERMANY

Tel: (49) 89-904-5024
FAX: (49) 89-904-4748

Japan

Xilinx, K.K.
Daini-Nagaoka Bldg. 2F
2-8-5, Hatchobori Chuo-ku
Tokyo 104
JAPAN

Tel: (03) 3297-9191
FAX: (03) 3297-9189

Asia Pacific

Hong Kong

Xilinx Asia Pacific
Unit No. 2318-2309
Tower 1, Metroplaza
Hing Fong Road
Kwai Fong, N.T.
HONG KONG

Tel: (852) 2410-2717
FAX: (852) 2494-7159
E-mail: **hongkong@xilinx.com**

XILINX®

The Programmable Logic CompanySM