PowerPC™

The IBM27-82660 PowerPC to PCI Bridge User's Manual

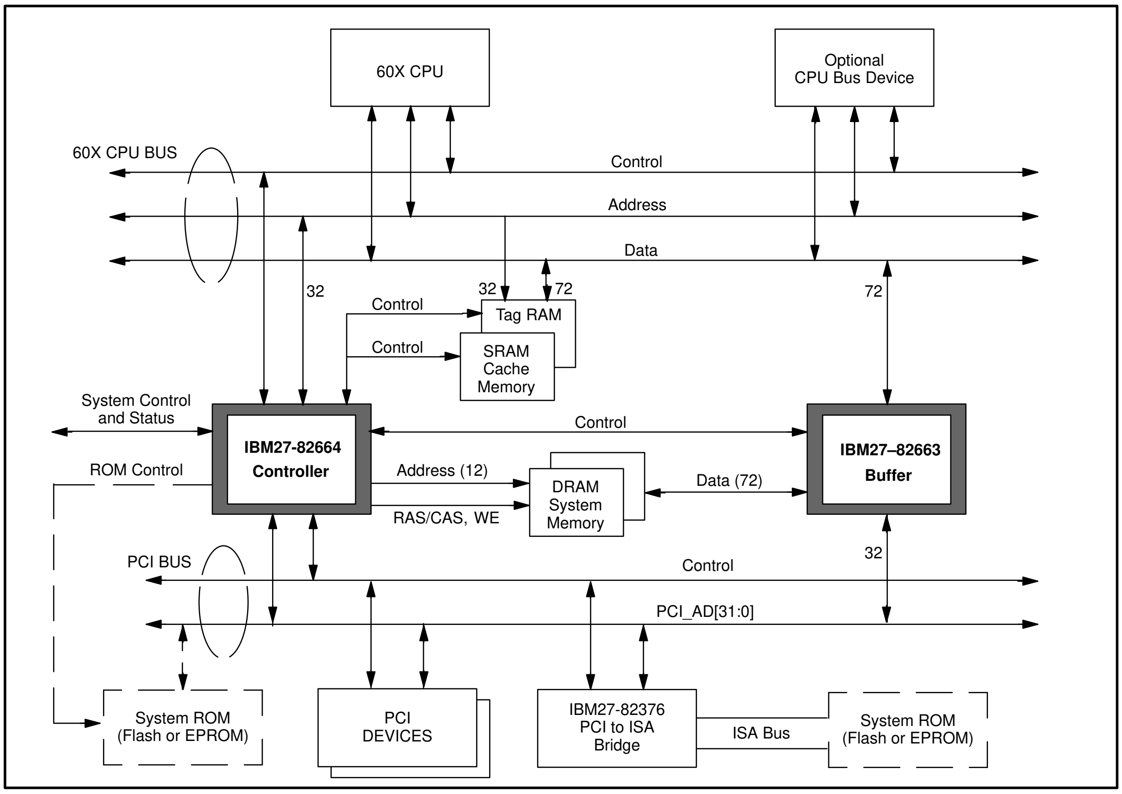
MPR660UMU–01

PowerPC to PCI Bridge, Memory Controller, L2 Cache Controller, ROM Controller, and System Resource Manager

IBM

The IBM27-82660 PowerPC™ to PCI Bridge

Memory, L2, ROM, and System Controller



IBM 660 Bridge Chip Set in a Typical System Configuration

Typical Pipelined CPU to Memory Performance at 66 MHz

Responding Device	Read	Write
L2 (9ns Synchronous SRAM)	–2–1–1–1	Snarf
L2 (15ns Asynchronous SRAM)	–3–2–2–2	Snarf
Page DRAM (70ns) Pipelined	–4–4–4–4	–5–4–4–4
EDO DRAM (70ns) Pipelined	–3–3–3–3	–4–3–3–3

Typical PCI to Memory Performance at 66 MHz CPU Clock and 33MHz PCI Clock

Read	5-1-1-1 -1-1-1-1 6-1-1-1 -1-1-1-1 6-1-1-1 -1-1-1-1 ... 6-1-1-1 -1-1-1-1
Write	5-1-1-1 -1-1-1-1 3-1-1-1 -1-1-1-1 3-1-1-1 -1-1-1-1 ... 3-1-1-1 -1-1-1-1

The IBM27-82660 PCI Bridge chip set (the 660 Bridge) provides an interface between the PowerPC 60X bus and the PCI Bus. The PCI bus standard defines an environment for high-speed local bus operations to service high-performance devices like graphics, LAN, and SCSI controllers.

The 660 Bridge chip consists of the IBM27-82663 (the 663 Buffer) and the IBM27-82664 (the 664 Controller). The 660 Bridge supports the PowerPC 601™, PowerPC 603™, and PowerPC 604™ microprocessors. The 660 Bridge supports both the L1 memory cache of the 60X CPU and an integrated write-through L2 cache controller.

- **Supports PowerPC 601, 603, and 604**
  - **Big-Endian and Little-Endian Modes**
  - **Up to 66MHz CPU Bus Clock**
  - **Enables Dual Endian-Mode Operation**
  - **Supports Address Bus Pipelining**
  - **Dual Bus Structure – CPU Data Bus Not Used For PCI to Memory Transfers**
  - **Low-Power Mode Supported**
  - **No–DRTRY# Mode Supported for 603 and 604 For Faster L2 Operation**
  - **One or Two CPU Bus Master Support**
  - **Reservations and Multi-Processor Synchronization Supported**
- **System Memory (DRAM) Controller**
  - **72-Bit Data Path for System Memory**
  - **Directly Supports Eight 8M, 32M, 64M, and 128M 8-Byte DRAM DIMMs**
  - **Directly Supports 4M, 8M, 16M, 32M, and 64M 4-Byte DRAM SIMMs**
  - **Supports up to 1G of System Memory**
  - **Supports Hyper Page Mode (EDO) DRAM For Higher Performance**
  - **Memory Configuration Registers**
  - **Programmable Memory Timings**
  - **Programmable Refresh Timer–Counter With Low Power Mode**
  - **System Memory ECC or Parity Check Generation and Checking (or No Check)**
  - **Supports 60X CPU and PCI Burst Reads and Writes of System Memory**
- **Direct Support for 8-Bit ROM**
  - **ROM can be Direct-Connect (on PCI Bus) or Remote (Attched to the I/O Bus Bridge)**
  - **Provides 1-byte to 8-byte Conversion and Pseudo-burst Mode on Reads**
  - **Flash Write and Write Lockout Supported**
- **Integrated L2 Cache Controller**
  - **Write-Through, Look-Aside, Direct Mapped**
  - **Burst or Asynchronous SRAM**
  - **Cache Size Determined Externally**
  - **Caches System Memory from 0 to 1G Supplies Data on CPU Read Hits Snarfs Data on CPU Misses**
  - **Snoops PCI Bus to Memory Transactions**
  - **L2 Data Parity Check on Read (If Enabled)**
  - **Internal L2 May Be Disabled and External L2 May Be Implemented**
- **PCI Bus Bridge**
  - **PCI Local Bus Specification Revision 2.0 Compliant**
  - **Up to 33MHz PCI Local Bus Clock Rate**
  - **Support for 1:1, 2:1, and 3:1 CPU to PCI clock rates**
  - **Enables Industry-Standard PCI Devices**
  - **Supports IBM27-82376 and Other PCI to ISA Bridges**
  - **Support for PCI Locking of System Memory Cache Line**
  - **ISA Master to System Memory Support**
  - **Supports Both Type 0 and Type 1 PCI Configuration Transactions**
  - **PCI Compliant I/Os**
- **Low Cost 208-Pin and 240-Pin Plastic Wire-Bond Flatpack Packaging**
- **Extensive Programmability**
- **Error and Exception Handling**
- **3.3V or 3.6V Power Supply**
  - **5V-Compatible I/Os**
- **PowerPC Reference Platform Version 1.1 Compliant**

Table of Contents

Appendixes

Figures

Tables

About This Book

Audience:
This book is designed for engineers and system designers who are interested in implementing PowerPC systems with a PCI bus. The material requires an understanding of computer systems at the hardware level.

Document Organization:

Section 1 —	An overview of the 660 Bridge with detailed lists of the features and functions of the 660 Bridge chip set
Section 2 —	663 Buffer and 664 Controller pin description tables arranged in functional groups with a separate table for all the interconnections between the two chips
Section 3 —	Description of the operations on the 60X CPU bus
Section 4 —	A summary of the bridge control registers
Section 5 —	The operation of the DRAM system memory controller
Section 6 —	Description of operations on the PCI bus
Section 7 —	The L2 cache controller
Section 8 —	The operation of the system ROM
Section 9 —	Error handling and reporting methods
Section 10 —	Additional material covering various bridge operations
Appendix A —	Timing diagrams and tables
Appendix B —	Example implementation schematics
Appendix C —	663 Buffer details of operation
Appendix D —	Electrical and mechanical characteristics
Appendix E —	
Appendix F —	Pin lists in alphabetic and numeric order.

Reference Material:
- *PowerPC 604 User's Manual*, IBM document MPR604UMU-01
- *PowerPC 603 User's Manual*, IBM document MPR603UMU-01
- *PowerPC 601 User's Manual*, IBM document MPR601UMU-02
- *PCI Local Bus Specification*, Revision 2.1, available from the PCI SIG
- *32MB SIMM Engineering Specification,* IBM document number MMDL02DSU-00
- *8MB SIMM Engineering Specification,* IBM document number MMDL01DSU-00
- *PowerPC Reference Platform Specification,* Version 1.0, IBM document MPRPRPPKG-01
- *The Power PC Architecture,* second edition, Morgan Kaufmann Publishers (800) 745–7323, IBM document MPRPPCARC–02
- *PowerPC 601 System Architecture*, Tom Shanley, Mindshare Press (800) 420-2677.

Document Conventions:
Kilobytes, megabytes, and gigabytes are indicated by a single capital letter after the numeric value. For example, 4K means 4 kilobytes, 8M means 8 megabytes, and 4G means 4 gigabytes.

In this document a word is 32 bits, a double-word is 64 bits, and a half-word is 16 bits.

The term SIMM is often used to mean DRAM module.

Fractional time values are identified with the terms ms, us, and ns, which represent milliseconds, microseconds, and nanoseconds respectively.

Hexadecimal values are identified (where not clear from context) with a lower-case letter h at the end of the value. For example, 001Fh means a hex value of 1F. Binary values are identified (where not clear from context) with a lower-case letter b at the end of the value. For example, 0101b means a 4-bit binary value of 0101.

In identifying ranges of values *from* and *to* are used whenever possible. The range statement from 0 to 2M means from and including zero up to (but not including) two megabytes. The hexadecimal value for the range from 0 to 64K is: 0000h to FFFFh.

The terms *asserted* and *negated* are used extensively. The term *asserted* indicates that a signal is active (logically true), regardless of whether that level is represented by a high or low voltage. The term *negated* means that a signal is not asserted. The # symbol at the end of a signal name indicates that the active state of the signal occurs with a low voltage level.

Signal ranges are always shown with the most significant number first (SIGNAL[MSb:LSb]). Signals ranges that have the first number greater than the second number (PCI_AD[31:0]) are shown in little-endian nomenclature. Those shown with the second number greater than the first (TT[0:3] for example), are shown using big-endian nomenclature. Signal range names used without an explicit range indication refer to the entire set of signals (PCI_AD means PCI_AD[31:0]).

Revision bars, such as the thick vertical line to the left of this paragraph, indicate that a change has been made to the so marked text since the previous revision (MPR660UMU-00) of this document (now MPR660UMU–01). Revision bars have been added wherever the text has changed, with no attempt to determine if the change is substantive.

Acronyms and Abbreviations:
The term 60X CPU refers to the PowerPC 601, 603, and 604 families of microprocessors.

The acronym TBD is used in areas of the document where the material is *to be determined*.

The term I/O Bridge or I/O Bus Bridge refers to a PCI master that serves to connect the PCI bus to a PC-standard bus like the ISA, EISA, or MicroChannel buses.

The term write-back means the same as copy-back in reference to a mode of cache operation.

Contributors:
This document was written and edited by Ben Drerup and Dale Elson, with substantial contributions by Robert Stevens, Sean Curry, and the Kauai-Lanai design and simulation team.

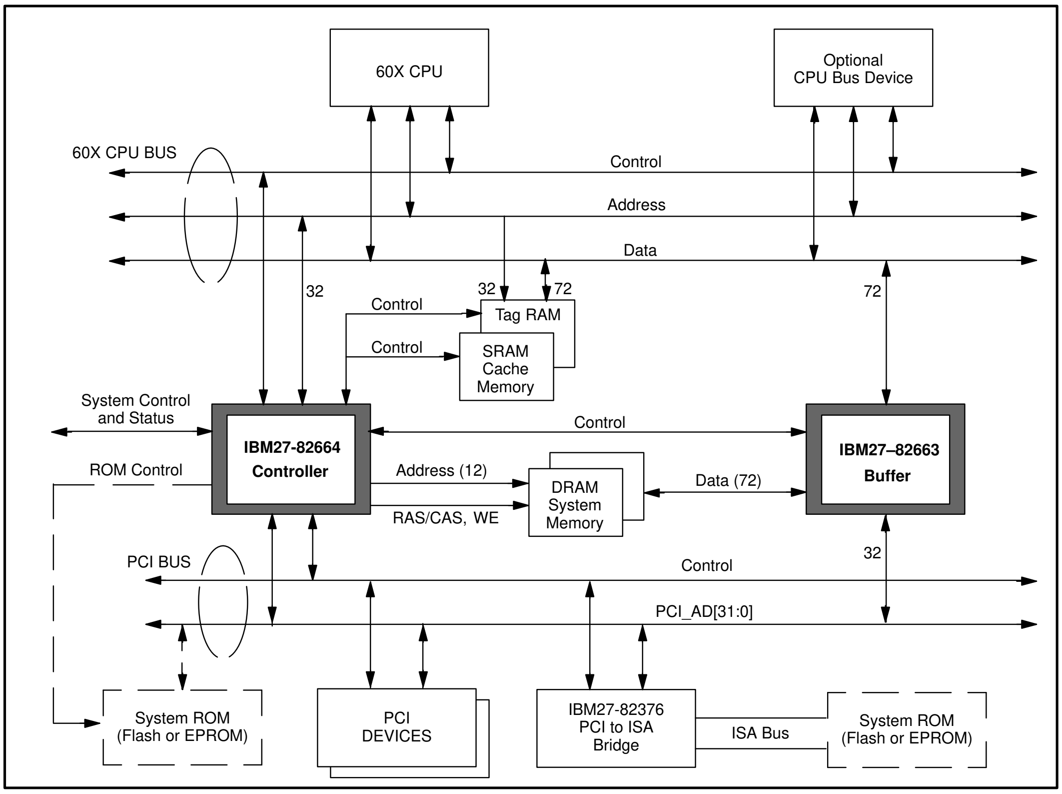


Figure 1-1. IBM 660 Bridge Chip Set in a Typical System Configuration

Section 1
660 Bridge Architectural Overview

The IBM27-82660 PCI Bridge chip set (the 660 Bridge) provides an interface that can connect a PowerPC 60X CPU to high-performance peripheral component interconnect (PCI) devices such as graphics, LAN, and SCSI controllers. The 660 Bridge chip set provides the necessary control and communications logic to connect a PowerPC 60X CPU to PCI-compliant devices through the PCI bus.

The 660 Bridge chip set is comprised of the IBM27-82663 data buffer (the 663 Buffer) and the IBM27-82664 PCI and memory controller (the 664 Controller). The 660 Bridge supports the PowerPC 601, PowerPC 603, and PowerPC 604 microprocessor chips. Within this document, these three microprocessor chips are referred to generically as the 60X CPU. The 660 Bridge memory controller supports both system memory (DRAM) and level 2 (L2) cache memory. The L2 cache controller supports write-through, look-aside, direct-mapped protocols for either burst or asynchronous SRAM.

Figure 1-1 shows the interconnection of the 660 Bridge and the associated devices that make up a functional system. The 660 Bridge connects to the 60X CPU bus and the PCI bus. The 660 Bridge manages all communication between the 60X CPU and PCI devices. The 660 Bridge also manages system memory (DRAM), L2 cache, and ROM operations.

1

1.1 The 663 Buffer Chip

The 663 Buffer chip provides the data bus path between the 60X CPU data bus, the memory data bus, and the PCI bus. This chip also contains logic for error detection and correction, data path crossovers, and endian mode swapping. The 663 Buffer contains latches on each data bus path to improve bandwidth.

1.2 The 664 Controller Chip

The 664 Controller chip contains the address bus and control logic that interfaces a PowerPC 60X processor with a PCI bus. It also contains a level 2 cache controller, a DRAM memory controller, and control logic for system central resources as required by a PowerPC reference platform compliant system.

1.3 Summary of 660 Bridge Features

This section summarizes the features of the 660 Bridge—including microprocessor support, the memory controller, the PowerPC CPU bus, the PCI expansion bus, the L2 cache controller, the system ROM controller, the bridge control register set, and the interrupt and exception handler.

- Complies with the PowerPC Reference Platform specification, version 1.0
- Complies with the PCI Revision 2.0 specification.

1.3.1 Packaging and Technology

- Designed in IBM YASU 3.3 to 3.6V CMOS4LP logic which allows I/Os that are compatible with 3.3V and 5.0V logic
- 208-pin (664 Controller) and 240-pin (663 Buffer) low-cost plastic wire-bond flatpacks
- Operates from 3.0V to 3.78V, allowing either 3.3V or 3.6V power sources.

1.3.2 Microprocessor Support

- Supports PowerPC 601, 603, and 604 microprocessors
- Supports 60X CPU bus speeds up to 66Mhz. Internally, the processor can run at faster frequencies
- Directly supports two 60X CPU bus masters
- Supports Little-endian and big-endian operating modes
  - Implements a byte swapper for little-endian mode
  - Support for low-order address translation (unmunging) in little-endian mode
- 64 bit wide 60X CPU data bus and 32 bit wide CPU address bus
- Utilizes address bus pipelining
- Dual bus structure between 60X CPU bus and PCI bus
- Data path latches enhance effective bandwidth
- Supports all clock modes for 60X CPUs (internal CPU clock to CPU bus)
- The MCP# signal on the 603 and 604 is supported for error reporting
- For reduced read latency, the no-DRTRY# mode of the 604 is supported
- The 603 no-DRTRY# mode is also supported except when the 603 is in 1:1 mode
- The 601 is always in DRTRY# mode
- Error reporting is by means of MCP# on the 603 and 604
- Error reporting is by means of TEA# and INT_CPU# on the 601
- The 604 store multiple 8-byte transfer is supported to the PCI bus
- Low-power mode is implemented through external clock stopping logic.

1

1.3.3 Memory Controller

- Supports memory operations for the PowerPC Architecture™
- Data bus path 72 bits wide—64 data bits and eight bits of optional ECC or parity data
- Supports Eight SIMM sockets, with empty SIMM sockets allowed at any position
- Eight RAS# outputs, eight CAS# outputs, and two write-enable outputs
- Supports industry-standard 8-byte (168-pin) SIMMs of 8M, 16M, 32M, 64M, and 128M that can be individually installed for a minimum of 8M and a maximum of 1G
- Supports industry-standard 4-byte (72-pin) SIMMs of 4M, 8M, 16M, 32M, and 64M that must be installed in pairs for a minimum of 8M and a maximum of 512M
- Programmable memory timings enable optimization of DRAM timings for a large variety of 60X CPU bus frequencies, DRAM speeds, and system topologies
- Mixed use of different size SIMMs, including mixed 4-byte and 8-byte SIMMs
- Support for parity, ECC, or neither
  - Generates ECC or even parity, eight bits for eight bytes, on all memory writes
  - Checks ECC or parity eight-bytes wide on all memory reads
  - Detects and corrects all single-bit errors in ECC mode
  - Detects all two-bit errors in ECC mode
- Page-mode access—fast page-mode is supported
- Support for extended-data-out (EDO) DRAM (hyper-page mode) for higher bandwidth memory performance
- Provides row-address and column-address multiplexing for SIMMs requiring:
  - 10 row by 10 column
  - 11 row by 10 column
  - 12 row by 10 column and 11 row by 11 column (supported simultaneously)
  - 12 row by 11 column
  - 12 row by 12 column
- Non-interleaved memory access operation
- Programmable DRAM refresh timer with low-power mode
- Memory refresh address counter
- Burst-mode memory address generation logic
  - 32-byte CPU bursts to and from memory
  - Variable length PCI burst to and from memory
- Little-endian and big-endian modes
- Supports ISA master to DRAM access

1

Memory Controller (Continued)

If 70ns DRAM is used in a system with the 60X CPU bus at 66MHz (CPU clock period 15ns), the minimum access times when the CPU accesses system memory are as follows:

- Pipelined burst read: 4-4-4-4 CPU clocks—16 CPU clocks for 32 bytes of data
- Pipelined burst write: 5-4-4-4 CPU clocks—17 CPU clocks for 32 bytes of data

If 70ns EDO DRAM is used in a system with the 60X CPU bus at 66MHz (CPU clock period 15ns), the minimum access times when the CPU accesses system memory are as follows:

- Pipelined burst Read: 3-3-3-3 CPU clocks—12 CPU clocks for 32 bytes of data
- Pipelined burst Write: 4-3-3-3 CPU clocks—13 CPU clocks for 32 bytes of data

Other minimum memory timings are as follows:

- Pipelined L2 cache read at 66MHz with synchronous SRAM—2-1-1-1
- Pipelined L2 cache read at 66MHz with asynchronous SRAM—3-2-2-2
  - PCI to memory read at 66MHx CPU and 33MHz PCI
    5-1-1-1 -1-1-1-1 6-1-1-1 -1-1-1-1 6-1-1-1 -1-1-1-1 ... 6-1-1-1 -1-1-1-1
  - PCI to memory write at 66MHx CPU and 33MHz PCI
    5-1-1-1 -1-1-1-1 3-1-1-1 -1-1-1-1 3-1-1-1 -1-1-1-1 ... 3-1-1-1 -1-1-1-1.

1.3.4 PCI Expansion Bus

- PCI bus frequency up to 33 MHz—PCI bus runs synchronously to the 60X CPU bus at one-third, one-half, or the same frequency
- 32-bit multiplexed PCI address and data path
- Support for I/O bus bridge (ISA, EISA, MicroChannel™)
- Supports IBM27-82376 PCI/ISA bridge chip
- PCI to DRAM access—with L1 and L2 cache snooping
- Support for memory mapping of 60X address space into PCI transactions
  - PCI I/O reads and writes
  - PCI memory reads and writes
  - PCI configuration reads and writes
  - PCI interrupt acknowledge reads
- Supports ISA bus master access to system memory with ISA bridge on the PCI bus
- Supports contiguous ISA I/O and non-contiguous ISA I/O mappings (non-contiguous I/O so operating systems can memory-protect 32-byte blocks of ISA I/O space)
- Uses external PCI arbiter (usually from the I/O bridge chip)
- Support for PCI resource locking of system memory
- Supports type 0 and type 1 PCI configuration cycles.

1.3.5 L2 Cache Controller

- Controls external burst-mode or asynchronous SRAMs for cached data
- Controls external tag RAMs for tag information
- Look-aside, direct-mapped, and write-through protocols
- Supports all cache sizes
- Externally determined Cache size
- Responds to snoop cycles for PCI reads and writes of system memory
- Option to check parity of data stored in L2 on read cycles
- L2 cache controller can be disabled if external L2 cache is used.

1.3.6 System ROM Controller

- Supports ROM attached to PCI_AD bus or I/O bus such as the IBM27-82376 ISA-attached ROM (I/O bridge must support ROM addresses FFF0 0000h to FFFF FFFFh)
- Supports 8-bit ROM, flash ROM, or EPROM
  - Provides 8-bit to 64-bit conversion on reads
  - 21-bit address support for up to 2M addressable ROM
- Flash ROM write cycles generated for in-system flash ROM writes
- Flash ROM write lock-out support
- Single-beat (one-byte to eight-byte) read cycle
- Single-beat (one-byte) write cycle
- Pseudo burst-mode (32-byte) read cycle.

1

1.3.7 The Bridge Control Register Set

- Implemented PCI register model
- Includes 650 Bridge-compatible registers
- Configuration through register 0CF8h and 0CFCh configuration method
- All chip set options (except two) are configured in the register set.
  - The 603 in 1:1 mode is a strapping option.
  - Direct-attach or remote ROM is a strapping option.

1.3.8 Interrupt and Exception Handler

- Non-maskable error reporting by means of TEA# or MCP#
- Supports error address and control capture registers
- Reports the following types of errors:
  - Memory parity or ECC error
  - CPU illegal transfer
  - PCI bus parity error
  - CPU data bus parity error
  - PCI cycle abort
  - L2 cache parity error
  - Write to locked flash ROM
  - Memory access out of range
- Drives CPU data lines to all one-bits on out-of-range memory reads
- PCI configuration read cycles return all one-bits when no device responds

1.4 Improvements Over 650 Bridge

- Includes L2 cache controller that supports synchronous and asynchronous SRAMs
- System performance improvements:
  - Programmable memory controller optimizes to memory speed and topology
  - Support for high-bandwidth memory technology (extended-data-out DRAM)
  - Up to 1G addressable DRAM
  - Utilizes 60X CPU bus address pipelining
  - Dual bus structure between 60X CPU bus and PCI bus
  - Data path latches provide bandwidth improvements
- ECC memory support
- Supports IBM27-82376 PCI/ISA bridge chip
- ROM can be attached to an ISA bus controller or to the PCI bus
- Improved clocking modes:
  - Supports 603 in 1-to-1 mode (60X internal clock vs. 60X CPU bus)
  - Support for PCI at one-third of 60X CPU bus frequency added
- The 604 store multiple 8-byte transfer is supported to the PCI bus
- Low-cost packaging—a 208-pin and a 240-pin plastic quad flatpack
- Uses an external PCI bus arbiter
- Provides basic multi-processor support
- Supports type 1 PCI configuration cycles in addition to type 0
- Implements Low-power mode
- Implements Expanded error control and reporting
- Provides a DRAM refresh timer
- Implements PCI-compatible configuration register set.

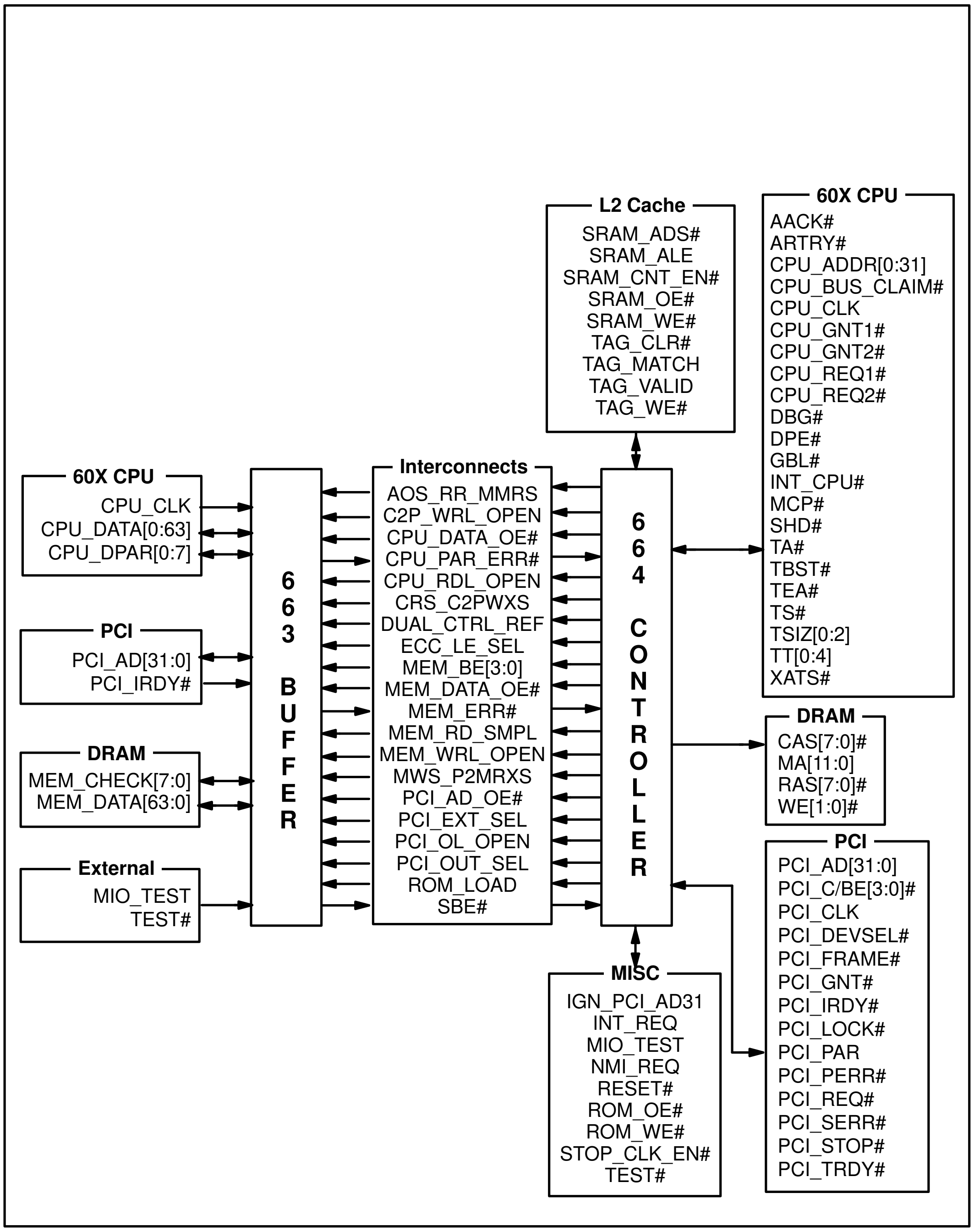


Figure 2–1. 660 Bridge Pin Connections

2

Section 2
660 Bridge Pin Descriptions

This section describes the connectivity of the 660 Bridge. See Figure 2–1.

The terms *asserted* and *active* indicate that a signal is logically true, regardless of the voltage level. The terms *negated*, *inactive*, and *deasserted* mean that a signal is logically false.

The # symbol at the end of a signal name listed in Table 2-1 indicates that the active or asserted state of the signal occurs with a low voltage level. Otherwise the signal is active at a high voltage level.

Pins which are marked s/t/s are sustained tri-state. This is an active low tri-state signal owned and driven by one agent at a time. The agent that drives the s/t/s pin active low must actively drive it high before letting it float. On the PowerPC bus, this is known as the restore function. On the PCI bus, the agent must drive the signal high for one clock and then tristate it. A new agent cannot drive the pin any sooner than one clock after the previous owner tri-states it. An external pull-up is required to maintain the inactive state.

Appendix E contains numeric and alphabetic lists of the pins in the 663 Buffer and 664 Controller. These lists include pin numbers.

Table 2-1. 660 Bridge Signal Descriptions

Signal Name	663	664	Description
CPU Bus Interface			
AACK#	—	I/O 109	CPU address acknowledge. AACK# is asserted on a CPU bus clock to signal the end of the current address tenure. AACK# is an input to the 664 Controller when a CPU bus target claims the current transaction by means of CPU_BUS_CLAIM#.
ARTRY#	—	I/O 110	Address retry. ARTRY# is asserted by a CPU bus device to signal that the current address tenure needs to be rerun at a later time. ARTRY# is asserted in the following circumstances: 1) An external device like another 60X CPU has an L1 cache hit. 2) The 660 Bridge is retried for one of the following reasons: a) A CPU to PCI cycle with PCI retry b) A CPU to memory cycle where memory is locked by the PCI c) A CPU to PCI cycle pending when a PCI master gets access to system memory

2

Table 2-1. 660 Bridge Signal Descriptions(Continued)

Signal Name	663	664	Description
CPU Bus Interface			
CPU_ADDR[0:31]	—	I/O	CPU address bus. Represents the physical address of the current transaction. The CPU_ADDR lines hold a valid address from the bus cycle in which TS# is asserted through the bus clock in which AACK# is asserted. CPU_ADDR is an input to the 664 Controller on any transactions initiated by a CPU bus master. The 664 Controller or the CPU bus target responds with AACK# when the address is no longer required, ending the address tenure. CPU_ADDR is an output from the 664 Controller on system memory transactions initiated by a PCI bus master device. The 664 Controller initiates an address tenure to snoop the address requested by the PCI bus master.
	see App E		
CPU_BUS_CLAIM#	—	I 132	CPU bus claim. This signal is asserted by a CPU bus target to inhibit the 664 Controller from driving the AACK#, TA#, TEA#, and CPU data bus lines. This signal is sampled by the 664 Controller on the second CPU_CLK after TS# is sampled active. CPU bus targets can only map to system memory space (0 to 2G) and only to memory space that is not cached by the L2. The L2 caches as much of the space from 0 to 1G as is populated by DRAM. So, if 8M is installed starting at 0, the CPU_BUS_CLAIM# can be asserted from 8M to 1G. If the internal L2 is disabled, then the entire 0 to 1M memory space can be claimed by CPU_BUS_CLAIM#.
CPU_CLK	I 144	I 121	CPU bus clock. The 660 Bridge supports up to a 66Mhz CPU bus clock frequency. The CPU_CLK frequency must be an integer multiple (1x, 2x, or 3x) of PCI_CLK.
CPU_DATA[0:63]	I/O	—	The 64-bit 60X CPU data bus. CPU_DATA[0] is the most-significant-bit. CPU_DATA[0:31] connect to the 60X CPU signals DH[0:31]. CPU_DATA[32:63] connect to the 60X CPU signals DL[0:31].
	see App E		
CPU_DPAR[0:7]	I/O	—	60X CPU data parity bus. The most-significant-bit is CPU_DPAR[0], the least-significant-bit is CPU_DPAR[7].
	see App E		
CPU_GNT1#	—	O 134	CPU bus grant. CPU_GNT1# is the grant line for the CPU_REQ1# request line. The 664 Controller CPU bus arbiter parks the bus on the device connected to CPU_GNT1# when no masters are requesting the CPU bus and no PCI bus master is requesting access to system memory.
CPU_GNT2#	—	O 135	CPU bus grant. CPU_GNT2# is the grant line for CPU_REQ2#.
CPU_REQ1#	—	I 127	CPU bus request. CPU_REQ1# is the request line for the primary CPU on the CPU bus.
CPU_REQ2#	—	I 128	CPU bus request. CPU_REQ2# is the request line for a secondary CPU bus master, like a secondary CPU or other device.
DBG#	—	O	CPU data bus grant. The 660 Bridge asserts DBG# (while ARTRY# is inactive) to signal that the requesting CPU may take ownership of the CPU data bus. This signal must be connected to the CPU only in the case of a 604 using fast L2/data streaming mode. Else this signal may be either connected to the CPU or a no–connect at the 660 Bridge and pulled low at the CPU.

Table 2-1. 660 Bridge Signal Descriptions(Continued)

Signal Name	663	664	Description
CPU Bus Interface			
DPE#	—	I 133	Data parity error from CPU. The processor checks data parity and reports any data beat with bad parity two CPU bus clocks after the TA# for that data beat. CPU bus parity is one odd parity bit for every byte—eight data parity bits total.
GBL#	—	O t/s 120	Global. This signal is asserted during snoop operations to indicate to the CPU that it must snoop.
INT_CPU#	—	O 139	CPU interrupt. The 664 Controller asserts INT_CPU# to signal the processor to run an interrupt cycle. The software is expected to run a read PCI interrupt acknowledge transaction in response to INT_CPU# being asserted. The 660 Bridge asserts INT_CPU# for various error conditions as well as for real interrupt requests. When an error is the cause of INT_CPU#, the 660 Bridge terminates the request for a PCI interrupt acknowledge transaction with TEA#—the PCI transaction is not run in this situation.
MCP#	—	O o/d 138	Machine check pin. When a 603 or 604 CPU is used with the 664 Controller, this pin is driven to notify the CPU of a system error from a source not affiliated with the current transaction. The 601 does not implement the MCP# pin, therefore the 664 Controller drives an INT_CPU# to the CPU, and terminates the resulting PCI interrupt acknowledge request with TEA#. The 664 Controller asserts MCP# in the event of a catastrophic or unrecoverable system error. This signal is asserted for two CPU bus cycles.
SHD#	—	O t/s 141	Shared. The function of this pin is to restore the SHD# net to a high state after it has been asserted (the same as ARTRY# restore). The restore is enabled by means of bit 4 of the memory controller timing programming register (8000 0821h).
TA#	—	I/O 111	CPU bus transfer acknowledge. TA# signals that a data transfer has occurred. For every CPU clock that TA# is asserted, a data beat completes. For a single-beat cycle, TA# is only one clock. For a four–beat burst, the data tenure is complete on the fourth cycle in which TA# is asserted.
TBST#	—	I/O 144	Transfer burst. TBST# indicates a burst transfer of four 64-bit double-words on the 60X CPU bus. Burst transfers by the 60X CPU are only allowed to system memory, not to the PCI bus. The 660 Bridge does support PCI burst reads and writes of system memory by PCI devices. The 664 Controller drives TBST# inactive during PCI to memory snoop cycles.
TEA#	—	O t/s 137	CPU bus transfer error acknowledge. Assertion of TEA# causes a machine check exception in the CPU. Assertion of TEA# terminates the current data tenure. The 664 Controller asserts TEA# in the event of a catastrophic or unrecoverable system error. TEA# can be masked by setting the mask TEA# bit in the bridge control registers. If XATS# is asserted for a PIO cycle, the 664 Controller asserts TEA# regardless of the condition of MASK_TEA#.

Table 2-1. 660 Bridge Signal Descriptions(Continued)

Signal Name	663	664	Description
CPU Bus Interface			
TS#	—	I/O 143	CPU bus transfer start. TS# is asserted low for one CPU bus clock to signal a valid address on the CPU_ADDR lines to start a transaction. TS# is an input to the 664 Controller when a CPU bus master had a qualified address bus grant on the bus on the preceding bus clock. TS# is an output of the 664 Controller to initiate a snoop cycle on behalf of a PCI bus master accessing system memory.
TSIZ[0:2]	—	I/O 145 146 147	CPU bus transfer size—number of bytes. The TSIZ lines are valid with the CPU_ADDR lines. TSIZ[0:2] is ignored when TBST# is asserted for 32-byte bursts.
TT[0:4]	—	I/O	Transfer type. Indicates the type of transaction currently in progress. The TT lines are valid with the CPU_ADDR lines.
	See App E		
XATS#	—	I 129	Extended address transfer start. When asserted this signal indicates that the 60X CPU is performing I/O controller interface (PIO) operations. If the T-bit in a 60X CPU segment register is set, it indicates that addresses in the range of that segment register are I/O controller interface accesses. If XATS# is asserted, the 664 Controller generates a TEA# error to the 60X CPU (regardless of the setting of MASK_TEA#).
PCI Bus Interface			
PCI_AD[31:0]	I/O	I/O	PCI address/data bus. Address and data are multiplexed on the same pins. A PCI bus transaction consists of an address tenure followed by one or more data tenures. The address tenure is defined as one PCI bus clock in duration, and coincident with the clock in which PCI_FRAME# is first asserted. After the first clock, the PCI_AD pins carry data. If no bus transaction is in progress, then the current PCI bus master must drive the PCI_AD pins. The 660 Bridge only samples or drives the PCI_AD lines during the address phase of a PCI transaction. PCI_AD is an input to the 660 Bridge during PCI transactions initiated by other devices. PCI_AD is an output of the 664 Controller during PCI transactions when the 660 Bridge is the initiator. PCI_AD is an input to the 663 Buffer during read data phases, and an output during write data phases.
	See App. E		

Table 2-1. 660 Bridge Signal Descriptions(Continued)

Signal Name	663	664	Description
PCI Bus Interface			
PCI_C/BE[3:0]#	—	I/O t/s 3 4 5 6	C (bus command) and BE (byte enable) multiplexed lines. During a PCI address phase this is a bus command. During a PCI data phase PCI_C/BE[3:0]# are byte enables—one bit for each of the four bytes on the PCI bus. During a PCI data tenure, PCI_C/BE[3]# applies to PCI_AD[31:24]—data byte three. If no bus transaction is in progress, then the current PCI bus master must drive the PCI_C/BE[3:0]# pins. PCI_C/BE[3:0]# is input to the 664 Controller during transactions initiated by a PCI bus master. The 664 Controller performs a decode based on the command and the address to determine if it is the target of the transaction. If the decode is successful, the 664 Controller reads or writes system memory based on the byte enables asserted. PCI_C/BE[3:0]# is output by the 664 Controller during transactions initiated by a CPU bus master to the PCI bus.
PCI_CLK	—	I 123	PCI bus clock. The PCI bus clock is required to be within PCI bus specifications. The frequency of the CPU bus clock must be either 1x, 2x, or 3x the PCI bus clock frequency. The 660 Bridge determines the CPU to PCI bus frequency ratio at the rising edge of RESET#.
PCI_DEVIL#	—	I/O s/t/s 204	PCI device select. PCI_DEVSEL# is driven by the target of the current PCI transaction. PCI_DEVSEL# is asserted by a PCI target that has decoded an address and control bit encoding and claims the transaction. When another PCI bus master initiates a PCI memory transaction, the 664 Controller drives PCI_DEVSEL# as the target of the transaction whenever the address is from 0 to 2G and within the range of physical memory (the installed DRAM space).
PCI_FRAME#	—	I/O s/t/s 200	PCI frame. The current PCI bus master drives PCI_FRAME#. PCI_FRAME# signals the beginning of the address tenure on the PCI bus. PCI_FRAME# can remain asserted throughout the data tenure and is deasserted to signal the final data tenure of the transaction.
PCI_GNT#	—	O 54	PCI bus grant. The PCI bus arbiter asserts PCI_GNT# in response to the 664 Controller asserting PCI_REQ# to request the PCI bus.
PCI_IRDY#	I 177	I/O s/t/s 201	PCI initiator ready. PCI_IRDY# is driven by the current PCI bus master. Assertion of PCI_IRDY# indicates that the PCI initiator is ready to complete this data phase. Wait states are inserted on any data tenure until both PCI_IRDY# and PCI_TRDY# are asserted.
PCI_LOCK#	—	I 53	PCI lock. This signal is used to allow PCI masters to establish a resource lock of one cache line of system memory. The 660 Bridge is never a locking master.
PCI_PAR	—	I/O t/s 7	PCI parity. Even parity across PCI_AD[31:0] and the PCI_C/BE[3:0]# lines. PCI_PAR is valid one PCI bus clock after either an address or data tenure. The PCI device that drove the PCI_AD lines is responsible for driving the PCI_PAR line on the next PCI bus clock. The 664 Controller checks and drives PCI parity.
PCI_PERR#	—	I/O s/t/s 10	PCI parity error. PCI_PERR# is asserted in response to a parity error detected by the PCI device receiving the PCI_AD, PCI_C/BE, and PCI_PAR lines. PCI_PERR# can be asserted on the second clock after the PCI_AD transfer occurs—one cycle after valid PCI_PAR is driven.

2

Table 2-1. 660 Bridge Signal Descriptions(Continued)

Signal Name	663	664	Description
PCI Bus Interface			
PCI_REQ#	—	O 58	PCI bus request. The 664 Controller asserts PCI_REQ# to the PCI bus arbiter to request the PCI bus for a transaction initiated by a CPU bus device.
PCI_SERR#	—	O s/o/d 71	PCI system error. PCI_SERR# is asserted for one PCI clock when a catastrophic failure is detected when the 660 Bridge is a PCI target. PCI_SERR# is not monitored by the 660 Bridge.
PCI_STOP#	—	I/O s/t/s 203	PCI stop. The target of the current PCI transaction can assert PCI_STOP# to indicate that the PCI target wants to end the current transaction. Data transfer can still take place if the target also asserts PCI_TRDY#, but that is the final data tenure.
PCI_TRDY#	—	I/O s/t/s 202	PCI target ready. The target of the current PCI transaction drives PCI_TRDY#. Assertion of PCI_TRDY# indicates that the PCI target is ready. Data transfer occurs when both PCI_TRDY# and PCI_IRDY# are asserted. Wait states are inserted on any data tenure until both PCI_IRDY# and PCI_TRDY# are asserted.
DRAM Interface			
CAS[7:0]#	—	O	Column address selects.
MA[11:0]	—	O	Memory address.
MEM_CHECK[7:0]	I/O See App E	— 	Memory ECC/parity bus. MEM_CHECK[7] is the most significant bit. ECC or even parity is generated and written on memory write cycles. ECC or even parity across eight bytes is checked on all memory read cycles.
MEM_DATA[63:0]	I/O See App E	— 	Memory data bus. MEM_DATA[63] is the most significant bit.
RAS[7:0]#	— See App E	O 	Row address selects.
WE[1:0]#	—	O 175 176	DRAM write enables. WE[1]# and WE[0]# are the same and are used to avoid the need for external buffers.
SRAM Interface			
SRAM_ADS#/ ADDR0	—	O 124	SRAM address strobe/address 0. Enables latching of new address for the SRAMs when burst SRAMs are used. Least significant address bit when asynchronous SRAMs are used.
SRAM_ALE	—	O 119	SRAM address latch enable. Enables latching of address for the SRAMs to support address pipelining when asynchronous SRAMs are used. This signal is always high when burst SRAMs are used.
SRAM_CNT_EN#/ ADDR1	—	O 125	SRAM count enable/address 1. Enables incrementing of burst address if burst SRAMs are used. Next to least significant address bit when asynchronous SRAMs are used.
SRAM_OE#	—	O 117	SRAM output enable.
SRAM_WE#	—	O 118	SRAM write enable.

2

Table 2-1. 660 Bridge Signal Descriptions(Continued)

Signal Name	663	664	Description
TagRAM Interface			
TAG_CLR#	—	O 116	Tag RAM clear. When asserted, all tags are forced to the invalid state.
TAG_MATCH	—	I 142	Tag RAM match indication (cache hit). This signal is asserted for a cache hit.
TAG_VALID	—	O 115	Tag RAM valid bit. When asserted, the current block is marked valid.
TAG_WE#	—	O 114	Tag RAM write enable.
Miscellaneous			
IGN_PCI_AD31	—	I 57	Ignore PCI_AD[31]. This signal is asserted when the Intel™ SIO is the PCI master. IGN_PCI_AD31 is used to allow ISA bus masters to access system memory in a manner compatible with the 650 Bridge when the SIO is used as the ISA bridge. The 664 Controller expects the memory address to appear in the range of 0 to 16M during the address tenure when IGN_PCI_AD31 is asserted and then maps the access to system memory at 0 to 16M.
INT_REQ	—	I 55	Interrupt request. This signal from the interrupt controller is synchronized with the CPU bus clock and passed through to the CPU as an interrupt on INT_CPU#. This signal is also used as the driver inhibit signal when the chip is in test mode.
MIO_TEST	I	I 154	Chip level test. Deassert low for normal operation.
NMI_REQ	—	I 56	Non–maskable interrupt request. When detected active (normally from the ISA bridge), an error is reported to the CPU. This signal is also used as the receiver inhibit signal when the chip is in test mode.
RESET#	—	I 156	Power–On–Reset. When this pin is low, all latches in the 664 Controller enter a pre–defined state. Clocking mode is re-sampled, and ROM write lockout is cleared.
ROM_OE#	—	O 47	ROM output enable. ROM_OE# enables direct-attached ROM. The signal is always high for remote ROM.
ROM_WE#	—	O 60	ROM write enable. Write enable for flash ROM for direct-attach ROM. The signal is always high for remote ROM.
STOP_CLK_EN#	—	I 151	Prepares the 660 Bridge for stopping of the CPU_CLK.
TEST#	I	I 155	Test mode. Pull to logic high during normal operation.

Table 2-1. 660 Bridge Signal Descriptions(Continued)

Signal Name	663	664	Description
Miscellaneous			
AOS_RR_MMRS	I 176	O 69	All Ones Select/ROM Remote/Mask MEM_RD_SMPL. This signal is used to force the data bus to 64 one-bits when the CPU reads memory of PCI space that is unoccupied. To force all ones, this signal must be asserted on the CPU_CLK that the CPU read latch samples data. When ROM_LOAD is asserted, this signal is used to determine the location of the ROM. When ROM_REMOTE is deasserted it indicates that the ROM is locally on the PCI bus. In this case ROM data always arrives on PCI_AD[31:24]. When ROM_REMOTE is asserted the ROM is assumed to be on a tertiary bus (such as the ISA bus). In this case ROM data arrives like all other one-byte PCI targets—first byte on PCI_AD[7:0], second byte on PCI_AD[15:8], etc. When the PCI is burst reading memory, MASK_MEM_RD_SMPL is asserted after the first MEM_RD_SMPL. It then stays asserted until the PCI–to–MEM read latch is empty. Note: On PCI burst reads of memory, the PCI–to–MEM read latch keeps getting refilled as long as data from the memory is available before the PCI uses it up.
C2P_WRL_OPEN	I 150	O 61	CPU–to–PCI Write Latch Open. When asserted, the CPU–to–PCI write latch accepts new data on each CPU_CLK. When deasserted the CPU–to–PCI write latch holds its current contents. This signal is asserted when data is to be sampled from the CPU. It is held deasserted until the data is written to the PCI.
CPU_DATA_OE#	I 138	O 197	CPU Data Output Enable. When asserted, the 663 Buffer drives the CPU_DATA bus on the next CPU_CLK. The 663 Buffer automatically drives the OE for CPU_DATA with the same value as it had on the previous CPU_CLK when an ECC correction occurs.
CPU_PAR_ERR#	O 178	I 192	CPU Data Bus Parity Error. When asserted, this signal indicates a parity error on the CPU data bus during a write cycle. This signal is only valid one CPU_CLK after the CPU data bus is valid (one clock delay).
CPU_RDL_OPEN	I 141	O 50	CPU Read Latch Open. When asserted the CPU read latch accepts new data on each CPU_CLK. When deasserted the CPU read latch holds its current contents. This signal is asserted when data is to be sampled from memory or the PCI. When sampling data from memory, this signal is also active on the following CPU_CLK to allow ECC corrections to occur if necessary. If no ECC corrections occur, the same data is provided by the MEM read ECC correction logic.

Table 2-1. 660 Bridge Signal Descriptions(Continued)

Signal Name	663	664	Description
Miscellaneous			
CRS_C2PWXS	I 147	O 65	CPU Read Select/CPU–to–PCI Write Crossover Select. When the CPU read latch is sampling data, this signal controls the CPU read multiplexer. When asserted, the memory data bus is routed to the CPU read latch. When deasserted, PCI data is routed to the CPU read latch. This signal must be valid on the CPU_CLK that the CPU read latch samples data. When the CPU–to–PCI write latch is sampling data, this signal controls the CPU–to–PCI write crossover. When asserted, the most-significant 32 bits are driven to the CPU–to–PCI write latch. When deasserted, the least-significant 32 bits are driven to the CPU–to–PCI write latch. This signal must be valid on the CPU_CLK that the CPU–to–PCI write latch samples data.
DUAL_CTRL_REF	I 175	O 205	Control Signal Multiplexer Select. For 663 Buffer inputs that have two functions (MEM_BEs and ECC_LE_SEL) this signal indicates which function is currently active. This signal is generated by dividing CPU_CLK by two.
ECC_LE_SEL	I 145	O 2	ECC Select/Little–Endian Select. This signal indicates use of ECC or byte parity when DUAL_CTRL_REF is high and indicates use of little-endian or big-endian mode when DUAL_CTRL_REF is low. Asserting ECC_LE_SEL enables the generation and checking of ECC to memory. Deassertion enables generation and checking of byte parity. Asserting ECC_LE_SEL indicates little–endian mode. Deassertion indicates big–endian mode.
MEM_BE[3:0]	I 157 156 155 154	O 1 208 207 206	Memory Byte Enables. The eight BEs for read–modifiy–write memory cycles are multiplexed on MEM_BE[3:0]. The first four BEs are asserted on MEM_BE[3:0] when DUAL_CTRL_REF is high and the second four are asserted on MEM_BE[3:0] when DUAL_CTRL_REF is low. When combined, they form MEM_RMW_BE[7:0]#. An active byte enable indicates valid write data from the CPU or PCI. The MEM_RMW_BE[7:0]# signals are sampled on the same clock that read–data is sampled by the MEM write latch, therefore MEM_BE[3:0] must be valid on the first and second clock before read data is sampled. When not running a read–mod–write cycle MEM_BE[3:0] should indicate all data lanes are enabled—MEM_RMW_BE[7:0]# all asserted low.
MEM_DATA_OE#	I 137	O 196	Memory Data Output Enable. When asserted, the 663 Buffer drives the MEM_DATA bus on the next CPU_CLK.
MEM_ERR#	O 179	I 194	Memory Error. When asserted, indicates an uncorrectable multi-bit or parity error has occurred during a memory read. This signal is only valid on the third CPU_CLK after MEM_RD_SMPL is asserted.
MEM_RD_SMPL	I 139	O 49	Memory Read Sample. This signal is asserted on the CPU_CLK that data from the memory is sampled during a CPU or PCI read. MEM_RD_SMPL is used by the ECC logic to determine when to sample ECC results. This signal is also used by the PCI read extension latch and the PCI–to–MEM read latch to load new data. See PCI_OL_OPEN for more details.

Table 2-1. 660 Bridge Signal Descriptions(Continued)

Signal Name	663	664	Description
Miscellaneous			
MEM_WRL_OPEN	I 146	O 51	Memory Write Latch Open. When asserted the MEM write latch accepts new data on each CPU_CLK. When deasserted the MEM write latch holds its current contents. This signal is asserted when data is to be sampled from the CPU or the PCI. It is held deasserted until the data is written to memory. When this signal is asserted on read–modify–write cycles, data is sampled from the CPU and memory or the PCI and memory. This signal is also active on the following CPU_CLK to allow ECC corrections to occur if necessary. If no ECC corrections occur, the same data is provided by the MEM read ECC correction logic.
MWS_P2MRXS	I 148	O 66	Memory write select/PCI–to–memory read crossover select. When the memory write latch is sampling data this signal controls the memory write multiplexer. When MWS_P2MRXS is asserted, the CPU data bus is routed to the memory write latch. When deasserted, the PCI data is routed to the memory write latch. This signal must be valid on the CPU_CLK that the memory write latch samples data. When the PCI–to–MEM read latch is sampling data this signal controls the PCI–to–MEM read crossover. When asserted, the most-significant 32 bits are driven to the PCI–to–MEM read latch. When deasserted, the least-significant 32 bits are driven to the PCI–to–MEM read latch. This signal must be valid on the CPU_CLK that the PCI–to–MEM read latch samples data.
PCI_AD_OE#	I 136	O 195	PCI Data Output Enable. When asserted, the 663 Buffer drives the PCI_AD bus. Note: This is an asynchronous input to the buffer chip—it is not clocked.
PCI_EXT_SEL	I 149	O 67	PCI Read Extension Select/PCI Write Extension Select. When the PCI is reading from memory this signal controls the PCI read extension multiplexer. When asserted, data from the PCI read extension latch is routed to the PCI–to–MEM read latch. When deasserted, data from the memory is routed to the PCI–to–MEM read latch. This signal must be valid when the PCI–to–MEM read latch samples data. When the PCI is writing to memory this signal controls the PCI write extension multiplexer. When asserted, data from the PCI write extension latch is routed to the MEM write latch. When deasserted, PCI data is routed to the MEM write latch. This signal must be valid when the MEM write latch samples data.

Table 2-1. 660 Bridge Signal Descriptions(Continued)

Signal Name	663	664	Description
Miscellaneous			
PCI_OL_OPEN	I151	O 64	PCI Other Latches Open. This signal controls the latch enables for the following three latches—the PCI–to–MEM read latch, the PCI read extension latch, and the PCI write extension latch. During PCI reads to memory, the PCI read extension latch and the PCI–to–MEM read latch accept new data when (PCI_OL_OPEN is asserted and when PCI_IRDY# is asserted) or (when MEM_RD_SMPL is asserted) (one CPU_CLK after MEM_RD_SMPL when ECC is enabled). During the period when enabling the latches is dependent on PCI_IRDY#, this signal must only be active on CPU_CLKs when PCI_CLK rises to guarantee sampling PCI_IRDY# only on rising PCI_CLK. Essentially, PCI_OL_OPEN on PCI reads to memory if !PCI_TRDY# && !PCI_CLK. This allows the 663 Buffer to advance to the next data as soon as the current data is accepted by the PCI master. This signal is also asserted during PCI writes to memory when data is sampled from the PCI. It is held deasserted until the data is passed to the MEM write latch.
PCI_OUT_SEL	I 153	O 68	PCI Output Select. When asserted the memory data is routed to the PCI output bus. When deasserted, CPU data is routed to the PCI output bus. This must be valid the entire time that data is driven to the PCI bus. This signal is asynchronous—not referenced to clock.
ROM_LOAD	I 140	O 70	ROM Load. This signal is used to load data from a ROM one byte at a time until eight bytes are received, then pass the eight bytes to the CPU. The ROM data buffer receives the one-byte transfers and assembles them into the eight-byte result. When ROM_LOAD is deasserted for more than one CPU_CLK the ROM data buffer is reset. When ROM_LOAD is then asserted, it receives data into its first byte. Each time ROM_LOAD is deasserted the first byte is passed to the second, the second is passed to the third, etc. ROM_LOAD should not be deasserted for more than one CPU_CLK while assembling the eight bytes to prevent resetting the buffer. Also, the ROM data buffer is routed toward the CPU data read latch when ROM_LOAD is asserted. It is expected that the CPU data read latch will grab eight bytes of ROM Data on the eighth ROM_LOAD.
SBE#	O 180	I 193	Single-Bit Error. When asserted, indicates a correctable single-bit error has occurred on the memory data bus. This signal is valid only on the CPU_CLK following the assertion of MEM_RD_SMPL. If the memory is not in ECC mode, this signal is undefined.

2

Section 3
The 60X CPU Bus

The 60X CPU bus connects the processor(s) with the 660 Bridge. The L2 cache is also attached to the 60X CPU bus. The 660 Bridge supports CPU bus frequencies up to 66MHz.

3.1 60X CPU Bus Masters

Typical system configurations using the 660 Bridge have two masters on the 60X CPU bus—the CPU and the 660 Bridge. The 660 Bridge is the master on the 60X CPU bus when it runs snoop cycles in response to PCI master access to system memory. The CPU is the master at all other times.

The 660 Bridge can support a second external master on the 60X CPU bus, which is usually an external L2 cache or a second CPU. To support a second external master, the 660 Bridge includes the following:

- A second set of bus request and bus grant signals.
- Sampling of ARTRY# on the third CPU_CLK (fourth clock for the 603 in 1:1 mode) following the falling edge of TS# for all accesses to system memory. This allows cache coherency protocols to function properly. Note that the L2 cache controller may already be asserting TA# on this cycle.

Each bus master must provide both of the following functions:

- Support at least one level of address pipelining, so that address tenure(n) can end and address tenure(n+1) can begin before data tenure n ends. Note that this requirement must be met even if there is only a single bus master. The 60X CPUs meet this requirement.
- Support the data bus busy protocol, driving DBB# while it is the data bus master, and monitoring DBB# to correctly determine the end of a data tenure mastered by another CPU bus agent. The 60X CPUs meet this requirement except in no–DRTRY# mode; therefore, when a second CPU bus master is installed, any 60X CPUs on the bus must have DRTRY# enabled.

Note: This is the extent to which the 660 Bridge supports multiple processors. This may be sufficient for many special purpose multi-processor systems, but general-purpose symmetric-multi-processor systems require additional support such as multi-interrupt controllers and error handlers. 660 Bridge compatibility with external logic to support symmetric-multi-processor systems has not been investigated.

3.1.1 External L2 as a CPU Bus Master

An external L2 cache that is look-aside with write-back capability can act as the second CPU bus master. To attach an external L2 controlled by the L2 invalidate register (8000 0814h), the system control register (8000 081Ch), and the I/O map-type register (8000 0850h), use the following setup:

1. Disable the internal L2 by setting bit 1 of the cache status register (index B1) to zero.
2. Enable external register support mode by setting bit 5 of the bridge chip set options 3 register (index D4) to one.

3.1.2 60X CPU Bus Targets

The 660 Bridge supports target devices attached to the CPU bus. By default, the bridge acts as the target for all CPU bus cycles; however, a mechanism has been provided for other CPU bus agents to claim a cycle and respond as the target of that cycle.

CPU bus targets can only claim memory cycles initiated by a CPU bus master other than the 660 Bridge. For example, they cannot claim snoop cycles or PIO cycles.

CPU bus targets can only claim cycles from the 0 to 2G system memory address space. These cycles must not be cached by the internal L2 cache controller. But the internal L2 caches all memory space which is occupied by DRAM. For example, if the L2 is enabled and 1G of memory is installed, then only the space fron 1G to 2G can be claimed. However, if the internal L2 is disabled, the entire address space from 0 to 2G can be claimed regardless of the installed memory.

A CPU bus target claims a cycle by asserting CPU_BUS_CLAIM# on the CPU_CLK after it samples TS# active. The 660 Bridge samples this signal two clocks after it samples TS# active.

If a CPU bus target claims a cycle, it is responsible for terminating both the address and data tenures associated with the cycle, including driving AACK#, TA#, and TEA#. The claiming CPU bus agent must drive AACK# even if the bus operation is retried (ARTRY# asserted). CPU bus targets may not assert DRTRY#. The CPU bus target is also responsible for detecting and reporting any errors that occur during the claimed cycle.

CPU bus targets must not drive AACK#, TA#, TEA#, or CPU_DATA until the listed number of clocks following the assertion of TS#:

1. First clock, if the internal L2 is disabled (second clock for 603 in 1:1 mode).
2. Third clock, if the internal L2 is enabled (fourth clock for 603 in 1:1 mode). This is necessary because the internal L2 drives these signals until it determines that the cycle is not a cache hit.
3. In compliance with the 60X bus spec, TA#, TEA#, and CPU_DATA also must not be driven until one clock after the previous data bus tenure has ended. These signals must not be driven until the data bus tenure for the claimed cycle has begun.

The CPU bus target must restore and tristate AACK#, TA#, and TEA# following their assertion at the end of the associated bus tenure.

The CPU bus target must have some means (e.g., DBG#) for determining when it may begin a data tenure.

The CPU bus target must not assert AACK# until the previous data bus tenure has ended. This is necessary to meet the 660 Bridge requirement of one level of pipelining.

3.1.3 60X CPU Bus Parity

The 660 Bridge can generate parity and check for parity errors on the 60X CPU data bus (CPU_DATA[0:63]). Whenever a 60X CPU bus master performs a write cycle, the 660 Bridge checks the parity. Whenever a 60X CPU bus master performs a read cycle, the 660 Bridge drives the data parity bits. If desired, the data SRAMs for the L2 cache can store and read the parity data to provide an extra measure of error checking.

The 660 Bridge does not generate or check CPU data bus parity on cycles claimed by other CPU bus targets.

3.2 CPU Bus Arbitration

The 660 Bridge arbiter coordinates the activities of the three CPU bus agents, CPU1, CPU2, and the snoop engine. The snoop engine of the bridge is a conceptual set of logic inside the Bridge which broadcasts snoop cycles to the CPU bus in response to PCI to memory transactions (see section 6.4).

Each CPU bus agent has an address bus request line and an address bus grant line. CPU1 is the default for the CPU. CPU may be an additional CPU, an external (to the Bridge) L2, or other well behaved CPU bus agent. The request and grant lines of the snoop engine are internal to the Bridge.

The 660 Bridge supports pipelined split-bus transactions, which feature an address tenure and a subsequent data tenure (see the *PowerPC 604 User's Manual*, especially section 8.2). The initial business of the Bridge arbiter is with the address tenure, which may or may not be followed by a data tenure.

3.2.1 Arbiter Rules

The following rules describe the operation of the arbiter:

1. There are three CPU bus masters that can generate CPU address bus requests: CPU1, CPU2, and the snoop engine. The priority of the agents is
   - CPU1 (the highest priority)
   - CPU2
   - Snoop engine (the lowest priority).

   If more than one bus request is sampled active at the same time, the address bus is granted to the highest priority requesting BM.
2. When no bus master is requesting the CPU address bus, the Bridge parks the bus on the CPU that most recently owned it. This behavior supports multiprocessing applications.
3. The Bridge supports CPU bus masters which conform to the bus arbitration protocol described in Chapter 8 of the *PowerPC 604 User's Manual*. In general, the CPU bus master will deassert BR# for one CPU clock cycle following the assertion of TS#.
4. The Bridge arbiter recognizes an active CPU address bus request from the snoop engine from the time that the Bridge decodes a PCI bus transaction to system memory

(FRAME# initially sampled active), until the termination of the PCI bus transaction. The bus request is considered to be negated during PCI bus idle (turn-around) cycles. Note that this bus request always goes inactive between consecutive PCI transactions.

5. The system arbiter (external to and independently of the Bridge) selects the current PCI bus master, who is in general allowed to initiate PCI transactions independently of the state of the CPU bus. Thus a PCI bus master can begin a PCI to memory transaction while a CPU has control of the CPU address bus. In this case:

5.1 The current CPU bus master is allowed to complete the current transfer:

5.1.1 The address tenure completes, followed by two CPU clocks during which the CPU address bus is rearbited. Then the snoop is broadcast onto the CPU address bus.

5.1.2 The data tenure of the CPU transfer completes normally.

5.2 The Bridge asserts DEVSEL# as usual. It is not delayed by the current CPU address or data bus activity.

5.3 At the end of the CPU to memory data tenure, there is an idle CPU clock, and then the memory controller begins the PCI to memory transaction.

5.3.1 On reads, TRDY# is delayed until the first data phase is valid on the PCI bus. In general, pipelined PCI read bursts require 2 fewer PCI clocks to deliver the first beat than are required for bursts that start from a CPU bus idle (non–pipelined bursts).

5.3.2 On writes, TRDY# is asserted 1 or 2 PCI clocks after the completion of the data phase of the CPU to memory transfer. The posted write buffers in the Bridge will accept the first four 4–byte data phases as 1/2–1–1–1. Subsequent data phases are accepted one at a time as the buffer writes each 4–byte to memory. Write times are affected by various factors, as discussed in section 6.4.

5.4 If the snoop cycle (see step 5.1.1) is retried by either CPU, the Bridge retries the PCI bus master and grants the CPU bus to the CPU bus agent that retried the cycle (assuming that it is requesting the bus). If both CPUs retry the cycle, the bus is granted first to CPU1.

6. Fairness is enforced by the Bridge arbiter, which will not re–grant the CPU bus to a previous owner before granting it to a new requester. This assures that each of the three agents gets a fair chance to own the bus. No agent is held off of the bus for more cycles than it takes for each of the other two agents to run one transaction (worst case).

6.1 An apparent exception to this (which does not violate fairness) is that if CPU1, CPU2, or the snoop engine is retried during a cycle, the arbiter will next grant the CPU bus to the bus agent that retried the cycle (assuming that it is requesting the bus). If both of the other agents retry the cycle, the CPU bus is granted first to the one with the highest priority, and then to the other agent. Then the bus is returned to the original owner. The worst case hold-off in this case is three CPU bus transactions.

7. When a CPU begins a CPU to PCI transaction (single-beat only):

7.1 If the Bridge has the PCI_GNT# (and the PCI bus is idle), the Bridge initiates the PCI transaction.

7.2 If the Bridge does not have the PCI_GNT#, then:

7.2.1 If there is currently a PCI to system memory transaction in progress, the Bridge will retry the CPU.

7.2.2 If there is not a PCI to memory transaction in progress (there is a PCI to PCI transaction in progress or the PCI bus is idle and not parked on the Bridge), then the Bridge arbitrates for the PCI bus and begins the transaction.

7.2.2.1 On CPU writes, the first transaction is posted even before the PCI bus is granted to the bridge. Completion of any second transaction depends on completion of the first transaction.

7.2.2.2 On CPU reads, the CPU transfer will not complete until data is returned from the PCI bus.

8. When a CPU begins a CPU to memory transfer, there is no collision with a PCI broadcast snoop cycle, because the CPU bus is granted to the snoop engine throughout the PCI to memory transaction.

3.3 Broadcast Snoop Details

PCI to memory transactions begin with the Bridge decoding and recognizing the transaction, arbiting for the CPU bus, and in the case of writes, completing a broadcast snoop cycle to the CPU bus. The memory-controller-active part of a PCI to memory transaction is delayed by the time it takes to arbitrate for the CPU address bus on writes (waiting for the snoop cycle to complete), if the transaction starts from a CPU bus idle condition. Pipelined PCI to memory transactions (which start while a previous CPU bus transaction is still running), do not suffer this initial delay, but they do have to wait for the completion of the data tenure of the previous transfer in order to begin. Pipelined transactions begin as soon as the memory controller is available: they do not have to wait for the snoop to complete, because it has already completed.

Another CPU address tenure is allowed to begin several CPU clocks before the end of the PCI transaction, as soon as the arbiter knows that it will not be broadcasting another snoop immediately (e.g., on the last beat of a PCI burst).

The following discussion includes three figures (Figure 3-1, Figure 3-2, and Figure 3-3) that show PCI to memory transactions and the associated broadcast snoop operations. These figures show the following:

- CPU and PCI clocks that are for reference only
- CPU address bus tenures from TS# to AACK# (no retries considered)
- CPU data bus tenures (these are approximate only – their duration is not to scale nor to clock)
- PCI bus tenures (same caveats).

3.3.1 Snoop Cycles From CPU Bus Idle

Figure 3-1 shows a CPU to memory transfer, followed by at least one CPU bus idle cycle, followed by a PCI bus master transaction to memory, and followed by another CPU bus address tenure. Since the PCI transaction begins during a CPU bus idle, as soon as the Bridge recognizes the PCI transaction, it grants the CPU bus to itself (the snoop engine), which requires a maximum of 1 PCI clock, and then it broadcasts a snoop cycle on the CPU bus. This snoop cycle is very similar to the address tenures of non-pipelined CPU to memory transfers.

3

If the PCI transaction is a read, the memory controller begins the memory read immediately (if there is a CPU bus retry, the memory controller can stop a read without corrupting the memory). The snoop operation requires 3 CPU clocks to complete, which is less time than is required for the memory controller to access the first 8 bytes of DRAM, so no time is lost to the PCI transaction because of the snoop. From the assertion of FRAME#, it will take 5 PCI clocks until the bridge places the first 4–byte on the PCI bus and the PCI bus master can sample TRDY# active. Subsequent data phases of the read are serviced at basically -1-1-1, so the first 4 beats take 5-1-1-1 to complete. For a PCI bus oriented view of the transaction, see Section 6.4.

If the PCI transaction is a write, the memory controller is delayed 2 PCI clocks (3 CPU clocks) for the snoop to complete before beginning the memory access; however, some overhead functions are occurring during this time, so the net effective snoop delay is usually 1 PCI clock. Following successful completion of the snoop, the PCI to memory write data phases are posted, and once the 1 to 2 PCI clock snoop delay is over, the subsequent 3 PCI data beats are accepted at -1-1-1. Thus the write pattern is 5-1-1-1.

Also note that in both the read and write cases, another CPU address tenure is allowed to begin several CPU clocks before the end of the PCI transaction (as soon as the arbiter knows that it will not be broadcasting another snoop immediately—e.g., on the last beat of a PCI burst).

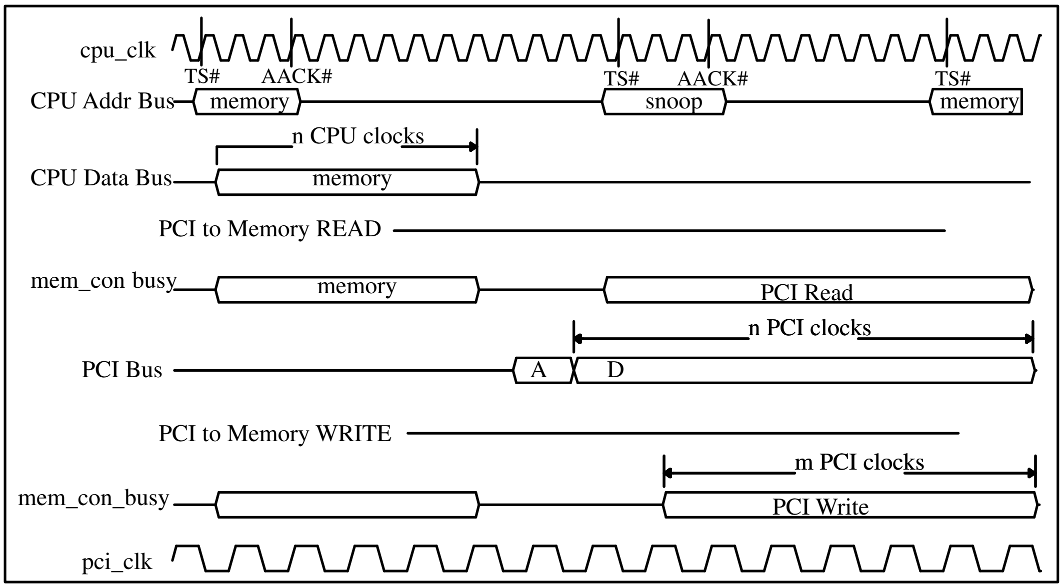


Figure 3-1. PCI Bus Snoops From CPU Bus Idle

3.3.2 Pipelined Snoop Cycle Following a CPU Bus Transfer

Figure 3-2 shows a PCI to memory transaction pipelined after a CPU to memory transfer. In this case, the Bridge recognizes the initiation of a PCI to memory transaction during the address tenure of a CPU to memory transfer. At the close of the CPU to memory address tenure, up to 2 CPU clocks of CPU address bus are idle for arbitration and internal overhead.

Then the bridge broadcasts the PCI snoop on the CPU address bus. The snoop cycle is completed before the end of the CPU to memory transfer, so it imposes no delay on either the CPU transfer or the PCI transaction.

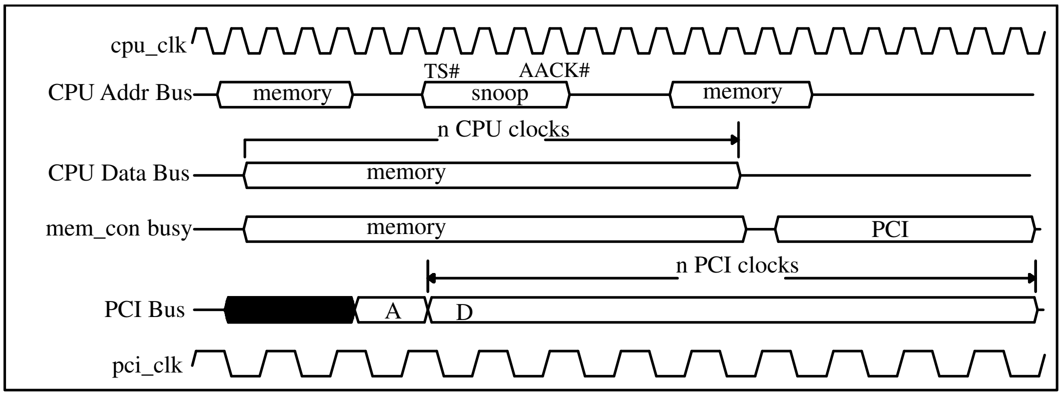


Figure 3-2. Pipelined PCI Bus Snoop Following a CPU Bus Transfer

3.3.3 Pipelined Snoop Cycle Following a Pipelined CPU Bus Transfer

Figure 3-3 shows an initial CPU to memory transfer, followed by a pipelined CPU to memory transfer, and followed by a PCI to memory transfer. The pipelined PCI transaction executes as it did in Figure 3-2. Note that snoop cycles continue to have little to no effect on CPU bus performance during pipelined transactions. In this case, the Bridge recognizes a PCI to memory transaction initiation sometime during the address phase of the second CPU to memory transfer. Note that the Bridge will not complete the CPU address tenure for transfer 2 until the data tenure for tenure 2 begins (1 clock after the end of the data tenure of transfer 1). The third transaction (the pipelined PCI snoop) has no effect on the timing of the two CPU transfers.

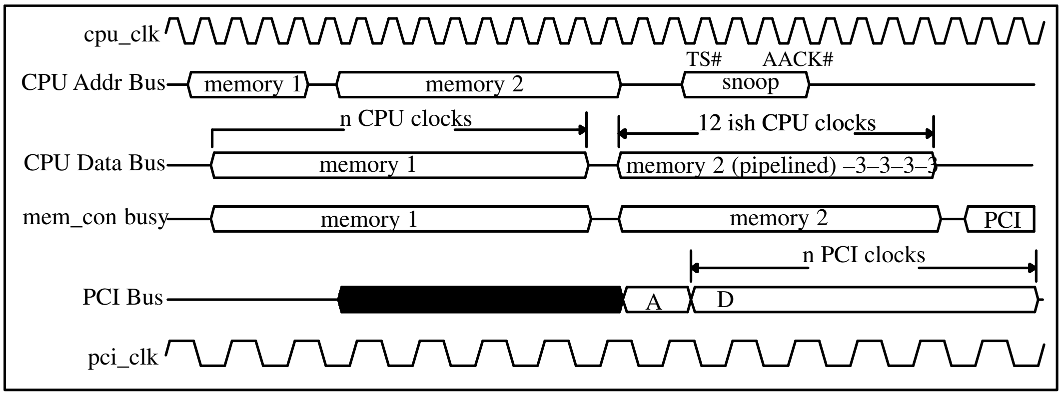


Figure 3-3. Pipelined PCI Bus Snoop Following a Pipelined CPU Bus Transfer

3.3.4 PCI Bus Snoop On Block Boundary During a PCI Burst Read

Figure 3-4 shows a PCI snoop that is incurred by a PCI burst crossing over a cache block 32-byte boundary during a PCI to memory read. See section 6.4.2.2 for a discussion of timing variable **M**.

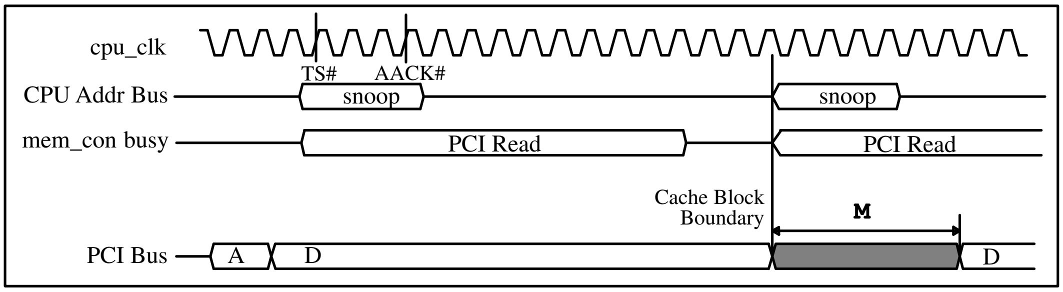


Figure 3-4. PCI Bus Snoop On Block Boundary During a PCI Burst Read

3.3.5 PCI Bus Snoop On Block Boundary During a PCI Burst Read

Figure 3-5 shows a PCI snoop incurred by a PCI burst crossing over a cache block 32-byte boundary during a PCI to memory write. See section 6.4.2.3 for a discussion of timing variable **X**.

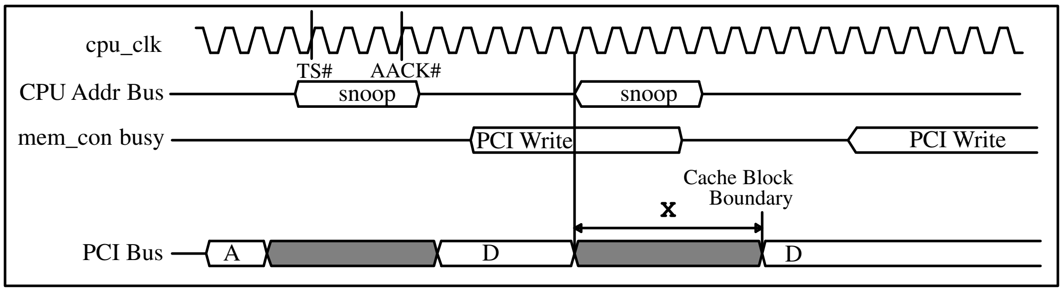


Figure 3-5. PCI Bus Snoop On Block Boundary During a PCI Burst Write

3.4 CPU Bus Address Mapping

The 660 Bridge determines the target of a CPU bus master transaction based on the CPU bus address range as shown in Table 3-1. The acronym BCR means bridge control register.

Table 3-1. 660 Bridge Address Mapping of CPU Bus Transactions

CPU Bus Address	Other Conditions	Target Transaction	Target Bus Address	Notes
0 to 2G 0000 0000h to 7FFF FFFFh		System Memory	0 to 2G 0000 0000h to 7FFF FFFFh	(1)(2)
2G to 2G + 8M 8000 0000h to 807F FFFFh	Contiguous Mode	PCI I/O Transaction, BCR Transaction, or PCI Configuration (Type 1) Transaction	0 to 8M 0000 0000h to 007F FFFFh	(3)
	Non-Contiguous Mode		0 to 64K 0000 0000h to 0000 FFFFh	(4)
2G + 8M to 2G + 16M 8080 0000h to 80FF FFFFh		PCI Configuration (Type 0) Transaction	PCI Configuration Space 0080 0000h to 00FF FFFFh	
2G + 16M to 3G – 8M 8100 0000h to BF7F FFFFh		PCI I/O Transaction	16M to 1G – 8M 0100 0000h to 3F7F FFFFh	
3G – 8M to 3G BF80 0000h to BFFF FFFFh		BCR Transactions and PCI Interrupt Ack. Transactions	1G – 8M to 1G 3F80 0000h – 3FFF FFFFh	(3)(6)
3G to 4G – 2M C000 0000h to FF7F FFFFh		PCI Memory Transaction	0 to 1G – 2M 0000 0000h to 3F7F FFFFh	
4G – 2M to 4G FFE0 0000h to FFFF FFFFh	Direct Attach ROM Read, Write, or Write Lockout	BCR Transaction	0 to 2M 0000 0000h to 007F FFFFh (ROM Address Space)	(2)
	Remote ROM	PCI Memory Transaction to I/O Bus Bridge	1G – 2M to 1G 3FE0 0000h to 3FFF FFFFh	(2)

Notes for Table 3-1:

1) System memory can be cached. Addresses from 2G to 4G are not cacheable.
2) Memory does not occupy the entire address space.
3) Registers do not occupy the entire address space.
4) Each 4K page in the 8M CPU bus address range maps to 32 bytes in PCI I/O space.
5) Registers and memory do not occupy the entire address space. Accesses to unoccupied addresses result in all one-bits on reads and no-ops on writes.
6) A memory read of BFFF FFF0h generates an interrupt acknowledge transaction on the PCI bus.

3.4.1 Address Mapping for Non-Contiguous I/O

Figure 3-6 shows the address mapping that the 660 Bridge performs in non-contiguous mode. The I/O map type register (address 8000 0850h) and the bridge chip set options 1 register (index BAh) control the selection of contiguous and non-contiguous I/O. In non-contiguous mode, the 8M address space of the 60X bus is compressed into 64K of PCI address space, and the 60X CPU cannot create PCI I/O addresses from 64K to 8M.

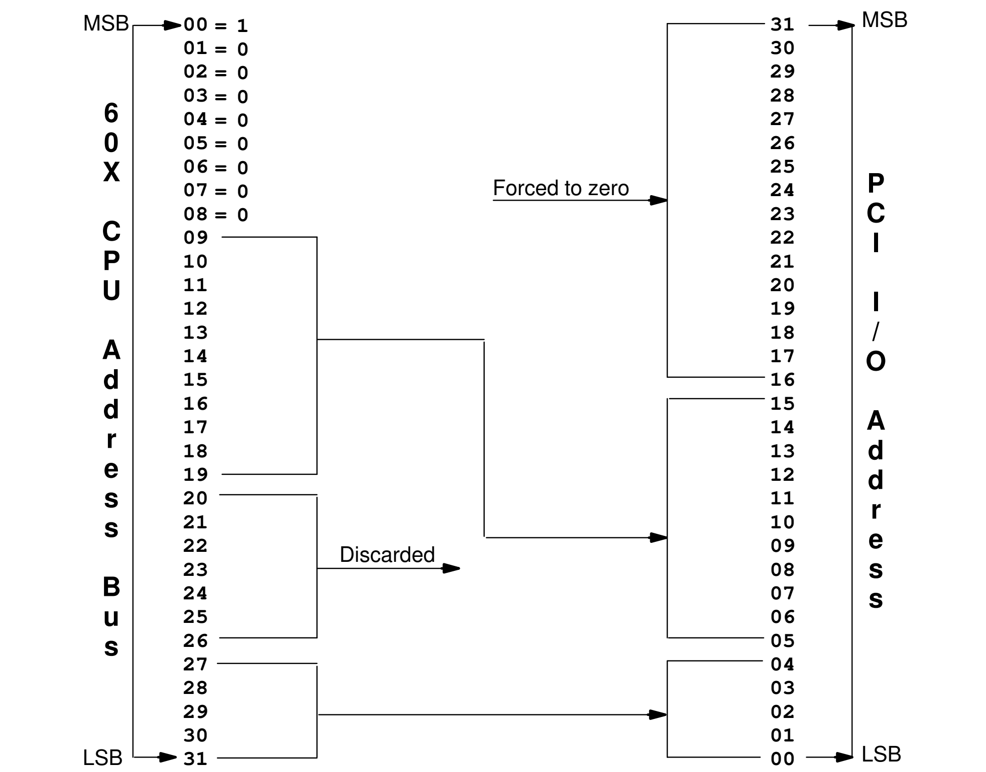
If little-endian mode is selected, CPU_ADDR[29:31] are unmunged before they reach PCI_AD[2:0]. In little-endian mode, CPU_ADDR[29:31] are munged by the 60X CPU before the address is driven to the 60X CPU bus. The 660 Bridge unmunges the three low-order address bits as part of the process of handling little-endian memory formatting. See the *PowerPC 601 User's Manual*.

In non-contiguous I/O mode, the 660 Bridge partitions the address space so that each 4K page is remapped into a 32-byte section of the 0 to 64K ISA port address space, so that 60X CPU protection attributes can be assigned to any of the 4K pages. This provides a flexible mechanism to lock the I/O address space from change by user-state code. This partitioning spreads the ISA I/O address locations over 8M of 60X CPU address space.

In non-contiguous mode, the first 32 bytes of a 4K page are mapped to a 32-byte space in the PCI address space. The remainder of the addresses in the 4K page are mirrors into the the same 32-byte PCI space. Each of the 32 contiguous port addresses in each 4K page has the same protection attributes in the 60X CPU.

For example, 60X CPU addresses 8000 0000h to 8000 001Fh are converted to PCI I/O port 0000h through 001Fh. PCI I/O port 0020h starts in the next 4K page at 60X CPU address 8000 1000h.

3



The three least-significant address bits are unmunged during the transformation if little-endian mode is selected.

Figure 3-6. Non-Contiguous PCI I/O Address Transformation

3.4.2 Address Mapping for Contiguous I/O

In contiguous I/O mode (CONTIG_IO asserted), 60X CPU addresses from 2G to 2G + 8M generate a PCI I/O cycle on the PCI bus with PCI_AD[29:00] unchanged.

3.4.3 PCI Final Address Formation

The 660 Bridge maps 60X CPU bus addresses from 2G to 4G as PCI transactions, error address register reads, or ROM reads and writes. The 660 Bridge manipulates 60X bus addresses from 2G to 4G to generate PCI addresses as follows:

- PCI_AD[31:30] are set to zero.
- PCI_AD[2:0] are unmunged if little-endian mode is selected.
- After unmunging, PCI_AD[1:0] are set to 00b except for PCI I/O cycles.

3.5 Transfer Type Decoding

Table 3-2 shows the 660 Bridge decoding of CPU bus transfer types. Based on TT[0:3], the 660 Bridge responds to CPU bus master cycles by generating a read transaction, a write transaction, or an address-only response. (The 660 Bridge ignores TT[4] when it evaluates the transfer type.)

The bridge decodes the target of the transaction based on the address range of the transfer as shown in Table 3-1. The transfer type decoding shown in Table 3-2 combines with the target decoding to produce the following:

- System memory reads and writes
- PCI I/O reads and writes
- PCI configuration reads and writes
- PCI interrupt acknowledge reads
- PCI memory reads and writes
- System ROM reads and writes
- Various bridge control register (BCR) reads and writes.

Within Table 3-2, SBR means single-beat read, and SBW means single-beat write.

Table 3-2. TT[0:3] (Transfer Type) Decoding by 660 Bridge

TT[0:3]	60X Operation	60X Bus Transaction	660 Bridge Operation For CPU to Memory Transfers	660 Bridge Operation For CPU to PCI Transactions
0000	Clean block or lwarx	Address only	Asserts AACK#. No other response. No PCI transaction.	
0001	Write with flush	SBW or burst	Memory write operation.	PCI write transaction.
0010	Flush block or stwcx	Address only	Asserts AACK#. No other response. No PCI transaction.	
0011	Write with kill	SBW or burst	Memory write operation. L2 invalidates addressed block.	PCI write transaction.
0100	sync or tlbsync	Address only	Asserts AACK#. No other response. No PCI transaction.	
0101	Read or read with no intent to cache	SBR or burst	Memory read operation.	PCI read transaction.
0110	Kill block or icbi	Address only	Asserts AACK#. L2 invalidates addressed block.	Asserts AACK#. No other response.
0111	Read with intent to modify	Burst	Memory read operation.	PCI read transaction.
1000	eieio	Address only	Asserts AACK#. No other response. No PCI transaction.	
1001	Write with flush atomic, stwcx	SBW	Memory write operation.	PCI write transaction.
1010	ecowx	SBW	Asserts AACK# and TA# if the transaction is not claimed by another 60X bus device. No PCI transaction. No other response.	
1011	Reserved		Asserts AACK#. No other response. No PCI transaction.	
1100	TLB invalidate	Address only	Asserts AACK#. No other response. No PCI transaction.	
1101	Read atomic, lwarx	SBR or burst	Memory read operation.	PCI read transaction.
1110	External control in, eciwx	Address only	660 asserts all ones on the CPU data bus. Asserts AACK#, and TA# if the transaction is not claimed by another 60X bus device. No PCI transaction. No other response.	
1111	Read with intent to modify atomic, stwcx	Burst	Memory read operation.	PCI read transaction.

Transfer types in Table 3-2 that have the same response are handled identically by the bridge. For example, if the address is the same, the bridge generates the same memory read transaction for transfer types 0101, 0111, 1101, and 1111.

The 660 Bridge does not generate PCI or system memory transactions in response to address only transfers. The bridge does drive all-ones onto the CPU bus and signals TA# during an eciwx if no other CPU bus agent claims the transfer.

3.6 CPU Bus Operation Details

The following sections discuss details about the operation of the CPU bus under various conditions.

3.6.1 CPU to System Memory

The system memory address space is from 0 to 2G. Physical memory does not occupy the entire address space. When the 60X CPU reads an unpopulated location, the Bridge returns all-ones and completes the transfer normally. When the CPU writes to an unpopulated location, the Bridge signals normal transfer completion to the CPU but does not write the data to memory. The memory select error bit in the error status 1 register (bit 5 in index C1h) is set in both cases.

All CPU to memory writes are posted. The 660 Bridge supports all CPU to memory transfer sizes that do not cross an 8-byte boundary.

3.6.2 CPU to PCI I/O Transactions

The 660 Bridge supports all CPU to PCI transfer sizes that do not cross a 4-byte boundary, and, to support the 604 store multiple instruction, the bridge supports 8-byte CPU to PCI writes that are aligned on an 8-byte boundary.

In compliance with the PCI specification, the 660 Bridge master aborts all I/O transactions that are not claimed by a PCI agent.

3.6.3 CPU to PCI Interrupt Acknowledge Transaction

Reading the interrupt acknowledge address (BFFF FFF0h) causes the bridge to arbitrate for the PCI bus and then to execute a standard PCI interrupt acknowledge transaction. The system interrupt controller claims the transaction and supplies the 1-byte interrupt vector. There is no physical interrupt vector BCR in the bridge. Other PCI bus masters can initiate interrupt acknowledge transactions.

3.6.4 CPU to ROM

The system ROM address space is from 4G – 2M to 4G. If the size of the installed ROM is less than 2M, it is mirrored throughout the ROM space. When the CPU writes to any ROM location while the ROM is locked out, the bridge signals normal transfer completion to the CPU but does not write the data to the ROM. The CPU bus write to the locked flash bit in the error status 2 register (bit 0 in index C5h) is set.

3.6.5 CPU to BCR Transfers

The bridge control registers (BCRs) do not occupy the entire address space assigned to them in the memory map. If the transaction is not claimed by the Bridge, other PCI agents can claim it. If the transaction is not claimed (including subtractively), the bridge master aborts the transaction.

Software should not configure any PCI agents to have I/O locations that overlap the BCR locations.

Section 4
Bridge Control Registers

The Bridge Control Registers (BCRs) allow the designer a high degree of control over system operations. CPU bus masters access BCRs to configure the operation of the 660 Bridge, to generate PCI configuration transactions, and to access various system components. Some BCR accesses are to real registers, either internal or external to the 660 Bridge. Some BCR accesses do not address real registers per se, but cause the Bridge to initiate various other sequences, such as PCI configuration transactions or ROM writes.

BCRs may be grouped as direct-access BCRs (which are accessed using a single CPU transfer) and indexed BCRs (which are accessed through the PCI/BCR configuration address and PCI/BCR configuration data registers using two CPU transfers).

Accesses to the 660 Bridge BCR set allow the CPU to:

- Configure, monitor, and control the 660 bridge,
- Perform PCI type 0 and type 1 configuration transactions,
- Perform PCI I/O transactions to external logic,
- Execute PCI interrupt acknowledge transactions, and
- Write to and lockout the ROM.

All BCR information is presented in little-endian notation. All BCR addresses are given for contiguous mode.

CPU writes to the register set are never posted or pipelined. This ensures that an operation changed by the write takes effect before any other CPU bus operation is executed.

4.1 Overview

4.1.1 Direct-Access Bridge Control Registers

A CPU bus master reads or writes a direct-access BCR by initiating a memory read or write to an address listed in Table 4-1. In response, the 660 Bridge requests the PCI bus from the PCI arbiter. Upon being granted the PCI bus (or if the bus is already parked on the CPU), the Bridge initiates one of the following transactions:

- A PCI BCR transaction to access a 650 Bridge compatible BCR
- A PCI BCR transaction to access an indexed BCR
- A PCI I/O transaction to access an external logic register
- A type 0 PCI configuration transaction to a PCI agent
- A PCI interrupt acknowledge transaction
- A direct-connect flash ROM write or ROM write lockout transaction.

The 650 compatible direct access BCRs are provided to ease software migration from the IBM27–82650 Bridge chipset to the 660 Bridge chipset. These BCRs are especially useful during system and software development.

The 660 Bridge is a much more programmable device than the 650 Bridge, but using the 650 compatible direct access BCRs tends to limit the programmability of the 660 Bridge to that of the 650 Bridge. Thus whenever possible, the designer should use the 660 Bridge (indexed) BCR set rather than the 650 compatible direct access BCR set.

4.1.2 Indexed Bridge Control Register Access

The indexed BCRs shown in Table 4-2 are accessed by means of the PCI/BCR configuration address BCR (address 8000 0CF8h, section 4.3.1.1) and the PCI/BCR configuration data BCR (address 8000 0CFCh, section 4.3.1.2). This method of access is described by the *PCI Local Bus Specification (Rev. 2.0),* and is therein defined as *Configuration Mechanism #1*. Note that these two registers are direct access BCRs.

The sequence of events that the Bridge generates to access the BCRs is called a PCI BCR transaction. This transaction is very similar to a PCI configuration transaction.

4.1.3 Indexed Bridge Control Registers

The 660 Bridge features a large indexed BCR set, which allows extensive programmability of the Bridge and the system. Indexed BCR operations allow CPU bus masters to perform the following operations:

- Access the 660 Bridge internal indexed BCRs
- Initiate PCI type 0 configuration transactions
- Initiate PCI type 1 configuration transactions

The target registers of PCI agents that claim the PCI configuration transactions appear to the CPU to be virtual indexed BCRs.

4.2 Direct-access BCRs

A CPU bus master reads or writes a direct-access BCR by initiating a memory read or write to an address listed in Table 4-1. In response, the 660 Bridge requests the PCI bus from the PCI arbiter. Upon being granted the PCI bus (or if the bus is already parked on the CPU), the bridge initiates one of the following transactions:

- A PCI BCR transaction to access a 650 Bridge compatible BCR
- A PCI BCR transaction to access an indexed BCR
- A PCI I/O transaction to access an external logic register
- A type 0 PCI configuration transaction to a PCI agent
- A PCI interrupt acknowledge transaction
- A direct-connect flash ROM write or ROM write lockout transaction

Table 4-1 is also useful as an index of transaction types and register descriptions.

4

Table 4-1. Direct-access BCRs

Bridge Control Register	CPU Bus Address	R/W	Bytes	Resultant Bus Transaction (See Section)	See Section
Special Port 92 Mirror	8000 0092	R/W	1	PCI I/O to 0000 0092	4.2.2.1
L2 Invalidate	8000 0814	W	1	PCI BCR (4.2.1.1) or PCI I/O to 0000 0814	4.2.2.2
System Control 81C	8000 081C	R/W	1	PCI BCR (4.2.1.1) or PCI I/O to 0000 081C	4.2.2.3
Memory Controller Misc	8000 0821	R/W	1	PCI BCR (4.2.1.1)	4.2.2.4
Memory Parity Error Status	8000 0840	R	1	PCI BCR (4.2.1.1)	4.2.2.5
L2 Error Status	8000 0842	R	1	PCI BCR (4.2.1.1)	4.2.2.6
L2 Parity Error Read and Clear	8000 0843	R	1	PCI BCR (4.2.1.1)	4.2.2.7
Unsupported Transfer Type Error	8000 0844	R	1	PCI BCR (4.2.1.1)	4.2.2.8
I/O Map Type	8000 0850	R/W	1	PCI BCR (4.2.1.1) or PCI I/O to 0000 0850	4.2.2.9
PCI/BCR Configuration Address	8000 0CF8	R/W	4	PCI BCR (4.2.1.1)	4.3.1.1
PCI/BCR Configuration Data	8000 0CFC	R/W	4	PCI BCR (4.2.1.1) or PCI Configuration (6.3.4.1)	4.3.1.2
PCI Type 0 Configuration Addresses IBM27–82650 Compatible	8080 08xx 8080 10xx 8080 20xx 8080 40xx 8080 80xx 8081 00xx 8082 00xx 8084 00xx	R/W	4	PCI Configuration (6.3.4.1)	6.3.4
System Error Address	BFFF EFF0	R	4	PCI BCR (4.2.1.1)	4.2.2.13
Interrupt Acknowledge	BFFF FFF0	R	1	PCI Interrupt Acknowledge	6.3.5

Table 4-1. Direct-access BCRs

Bridge Control Register	CPU Bus Address	R/W	Bytes	Resultant Bus Transaction (See Section)	See Section
ROM Write	FFFF FFF0	W	4	PCI BCR (ROM)	8
ROM Lockout	FFFF FFF1	W	1	PCI BCR (ROM)	8

4.2.1 Bus Transactions

The following sections discuss in detail some of the bus transactions shown in Table 4-1.

4

4.2.1.1 PCI BCR Transactions

A PCI BCR transaction is the result of a CPU transfer to a CPU address location shown in Table 4-1. Figure 4-1 shows the information flows in the system and Figure 4-2 shows the basic timing. During writes, the data flows from the CPU to the 663 Buffer over the CPU data lines and from the buffer to the controller over the PCI_AD lines. During reads, the data flows from the 664 Controller to the buffer over the PCI_AD lines and from the buffer to the CPU over the CPU data bus. This allows data to be passed on the PCI_AD lines from the 663 Buffer chip, which attaches to the CPU data bus, to the 664 Controller chip, which contains the physical registers.

Only the CPU is capable of causing PCI BCR transactions. Attempts by PCI bus masters to access the BCRs are not claimed by the bridge.

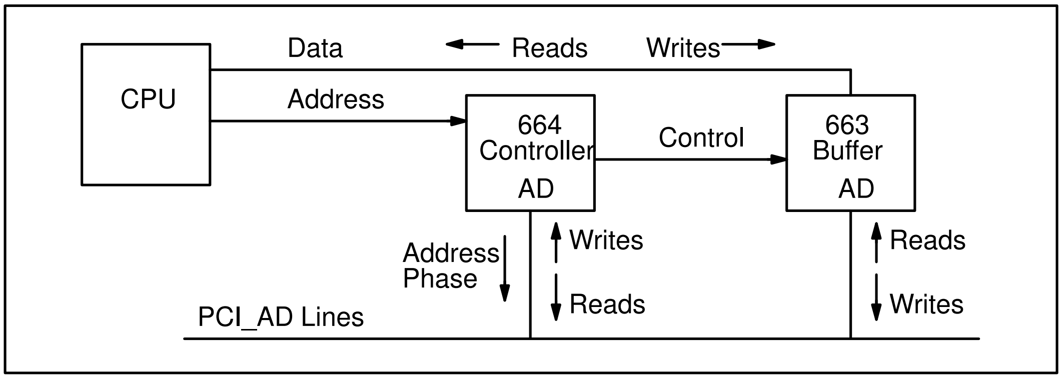


Figure 4-1. BCR Configuration Information Flow

The CPU initiates the PCI BCR transaction by beginning a memory transfer with TS#. The controller decodes the transfer as a BCR access. On the next rising edge of PCI_CLK, the controller asserts PCI_REQ#. Some number of PCI_CLK cycles later, the PCI arbiter grants the PCI bus to the CPU by asserting PCI_GNT#. On the next PCI_CLK (clock 1), the bridge asserts PCI_FRAME# to start the PCI transaction.

During the address phase, the controller drives Bh onto the PCI_C/BE# lines (during reads and writes) to signal a configuration transaction, and drives all zeros onto the PCI_AD lines for the address phase. Since the address is 0000 0000h, none of the IDSEL#s are activated.

During the data phase, the controller drives 0000b onto the PCI_C/BE# lines, and asserts PCI_IRDY#.

During read transactions, the controller drives the data from the selected register onto the PCI_AD lines during the data phase. The data travels over the PCI_AD lines to the buffer and then over the CPU data lines to the CPU.

TBD

Figure 4-2. BCR Configuration Basic Timing Diagram

During write transactions, the controller disables its drivers, and after a turn-around cycle, the buffer drives the data from the CPU bus onto the PCI_AD lines. The controller then latches the required data into the BCR.

At the end of the data phase, the controller signals TA# and AACK# to the CPU to terminate the CPU bus transfer. The controller deasserts PCI_IRDY# and PCI_FRAME#, ending the PCI transaction and allowing another PCI bus master to begin a transaction. Note that PCI_DEVSEL#, PCI_STOP#, and all IDSEL#s remain high during the entire transaction. Other PCI agents monitoring the bus see a configuration transaction which is master aborted.

The 660 Bridge deasserts PCI_REQ# on PCI_CLK two. The arbiter can deassert PCI_GNT# to the bridge on or after PCI_CLK one, because PCI_FRAME# remains asserted throughout the transaction, preventing a bus idle state.

4.2.2 Direct-Access BCR Listing

This section contains detailed listings of the direct-access BCR set. All addresses provided are in the contiguous address mode. The non-contiguous address is different. All registers are described in little-endian fashion with the highest numbered bit (usually D7) as the most-significant bit and D0 as the least-significant bit. In big-endian mode, the bit ordering is the same, with the left-most bit still as the most-significant bit, but typically referred to as D0. Byte addresses are specified so that (for a single byte transaction) the address is the same regardless of the endian mode.

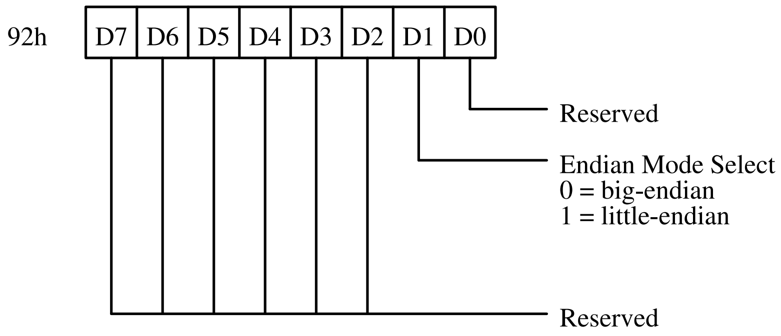
4.2.2.1 Special Port 92 Mirror BCR

Direct Access 8000 0092	Read/Write	Reset to 00

CPU reads of this register always cause the bridge to arbitrate for the PCI bus and then to execute a single-byte PCI I/O read of location 8000 0092 (the data returned is sourced by a PCI agent). CPU writes to this register are latched into the bridge BCR and also forwarded to the PCI bus as a single-byte PCI I/O write to location 8000 0092. This allows the 660 Bridge to implement the function associated with this register without running wires from an external register into the 664 Controller and also allows systems to support other functions in bits 7:2 and 0 of special port 92. This transaction is usually subtractively decoded by the I/O bus bridge for access to an external logic register.

The port 92 register is a PCI target device external to the Bridge. It either actively or subtractively decodes and responds to a PCI I/O read or write of PCI address 8000_0092 with DEVSEL# and TRDY#, according to correct PCI protocol. Regardless of whether the Bridge is in internal or external register mode, reads and writes of this register are forwarded to the PCI bus. Reads always return the data which is sourced by the PCI device (the contents of the internal BCR are used during the operation of the Bridge). The Bridge always drives the data lines during writes, and the data is also latched into the internal BCR. If the PCI target does not respond to the transaction (during either a read or a write and in both internal and external modes) an error is generated and correct operation does not occur.

CPU writes to this register are not posted or pipelined to ensure that the results take effect before the CPU begins any other bus operations. Other PCI bus masters are able to access the external register, but during writes the data is not latched into the 660 Bridge.



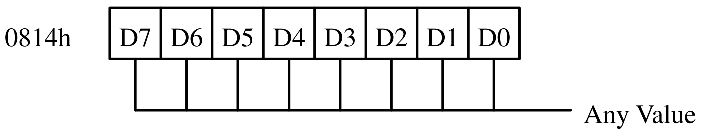
Bit 1 Endian mode select: If the value of this bit is changed, the endian mode is switched immediately following completion of the CPU write to the register.

4.2.2.2 L2 Invalidate BCR

Direct Access 8000 0814	Write Only	Reset: Undefined

This 8-bit, write-only register causes all contents of the internal L2 cache to be invalidated. The internal L2 cache does not have to be disabled during this operation. Reads to this register are undefined and do not cause an L2 invalidate.

This register can be put into external register support mode so that writes to this register are latched into this register and are also forwarded to the PCI. In this mode, reads to this register are always forwarded to the PCI, which allows functions to be added to the reserved bit locations of this register.



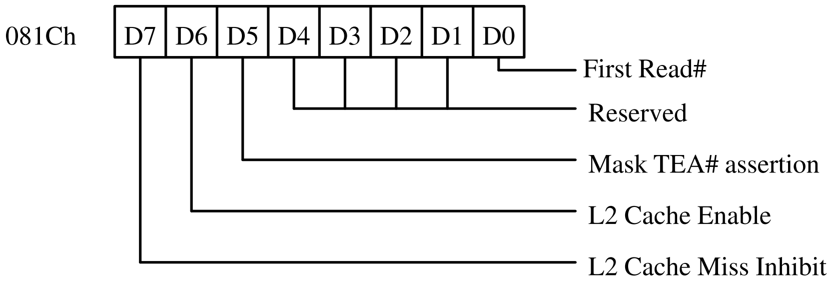
Bits 7:0 Writing any value to this register causes the L2 invalidate operation.

4.2.2.3 System Control 81C BCR

Direct Access 8000 081C	Read/Write	Reset to 00

This BCR controls the L2 cache, the TEA# mask, and other functions.

This 8-bit, read/write BCR can be put into external register mode so that CPU writes to this BCR are latched into the bridge BCR and are also forwarded to the PCI bus as a PCI I/O write to location 0000 081C (the bridge arbitrates for the PCI bus and then executes a single byte, single data phase PCI I/O write). In this mode, reads to this BCR are always forwarded to the PCI as I/O reads to location 0000 081C, which allows the system to use these bits externally (the external logic must provide the proper read value of all bits including bits 7:5).

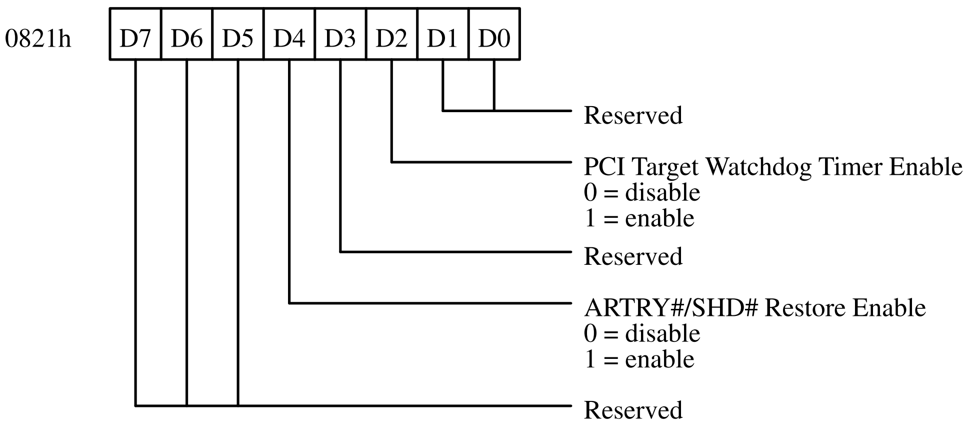


Bit 0	First_Read#. This bit is 0 the first time this BCR is read after reset. Subsequent reads of this BCR return 1 for this bit. This bit cannot be written by software. Writes to this BCR do not affect the state of this bit (including writes before the initial read).
Bits 4:1	Reserved. May be used externally in external register support mode.
Bit 5	TEA# assertion enabled. When enabled, the 664 will assert TEA# when an error condition is detected. Some error conditions can cause a bus hang (for example, XATS# assertion) so the 664 Controller still terminates such cycles with TEA#, even if this bit is set to 0. This bit can also be accessed via the bridge chipset options 1 BCR (index BAh, see section 4.3.37). This bit may be used externally in external register support mode. 0 : TEA# assertion disabled. 1 : TEA# assertion enabled.
Bit 6	L2 Cache Enable: When disabled, the L2 cache does not respond to any cycles (and does not maintain coherence or update the tags). Disabling the L2 cache does not cause its contents to be invalidated. This bit only enables the internal L2 if the internal L2 controller is enabled by bit 1 of the cache status register (index B1h, section 4.3.30). This bit may be used in external register mode to enable/disable an external L2. 0 = L2 cache disabled or not present. 1 = L2 cache enabled.
Bit 7	L2 Cache Miss Inhibit: This prevents L2 cache misses from updating the L2 cache. This allows the L2 cache to retain its contents during memory accesses and remain coherent (snooping and tag updates continue to occur). 0 = Prevent updates 1 = Normal operation (allow updates)

4.2.2.4 Memory Controller Miscellaneous BCR

Direct Access 8000 0821	Read/Write	Reset to 14h

This 8-bit, read/write register controls miscellaneous functions. The control of memory controller timings from this register is not supported. Instead, this is done by means of the 80h to A7h registers in the 660 Bridge primary register space.



4

Bit 2 PCI Target Watchdog Timer Enable: This bit controls the enable of the PCI target watchdog timer. When enabled, this timer is cleared and begins counting when a PCI target activates PCI_DEVSEL#. The timer stops when the PCI cycle terminates. If the PCI cycle has not terminated when the timer count reaches 2000 PCI clocks, the cycle is master aborted.

Note: The PCI specification places no limitation on the response time of PCI targets; therefore, this timer needs to be disabled if slow PCI devices are used. Disabling this timer allows the system to lock up if a PCI target hangs after asserting PCI_DEVSEL#.

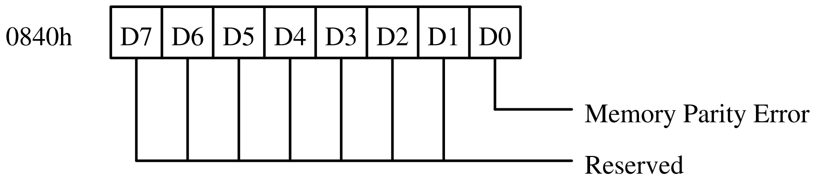
Bit 4 ARTRY#/SHD# Restore Enable: This bit controls the enable of the ARTRY# and SHD# restore logic in the 664 Controller. When enabled, the 664 Controller restores ARTRY# and SHD# regardless of what device drove the signal active. This mode can only be used if ARTRY# and SHD# restore is disabled on all other CPU bus devices.

When bit 4 is disabled, the 664 Controller only restores ARTRY# and SHD# if it drove it active.

4.2.2.5 Memory Parity Error Status BCR

Direct Access 8000 0840	Read Only	Reset to 01h

This 8-bit, read-only register indicates the status of memory parity or multi-bit ECC errors. This bit is deactivated by reading the system error address register (port BFFF EFF0h). Bit 0 may also be accessed via the error status 1 register (index C1h, section 4.3.40).

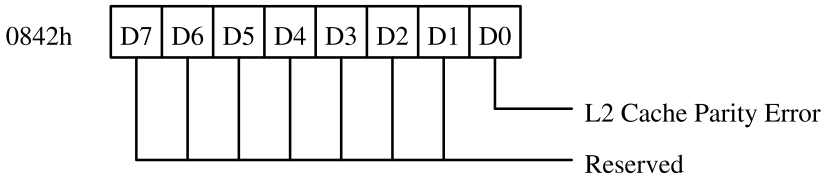


Bit 0 Memory Parity Error:
0 = Error Detected
1 = No Error Detected.

4.2.2.6 L2 Error Status BCR

Direct Access 8000 0842	Read Only	Reset to 01h

This 8-bit, read-only register indicates if a parity error has been detected during a CPU read from the L2 cache. This bit is deactivated by reading the L2 cache parity error read and clear register (port 8000 0843h). This bit may also be accessed via the error status 2 register (index C5h, section 4.3.44).

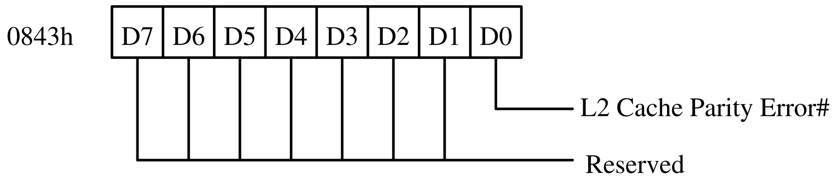


Bit 0 L2 Cache Parity Error:
0 = Error Detected
1 = No Error Detected.

4.2.2.7 L2 Parity Error Read and Clear BCR

Direct Access 8000 0843	Read Only	Reset to 01h

This 8-bit, read-only register indicates if a parity error has been detect during a CPU read from the L2 cache and clears the error if it is active. This bit may also be accessed via the error status 2 register (index C5h, section 4.3.44), but that access will not automatically clear the bit.

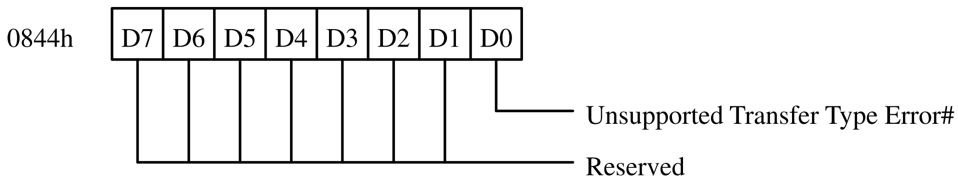


Bit 0 L2 Cache Parity Error:
0 = Error Detected
1 = No Error Detected.

4.2.2.8 Unsupported Transfer Type Error BCR

Direct Access 8000 0844	Read Only	Reset to 01h

This is an 8-bit, read-only register that indicates the status of unsupported transfer type errors or XATS# asserted from the CPU. This bit is cleared (to 1) by reading the system error address register (port BFFF EFF0h). Transfer type error status is also available via the error status 2 register (index C1h, section 4.3.40).



Bit 0 Unsupported Transfer Type Error:
0 = Error Detected
1 = No Error Detected.

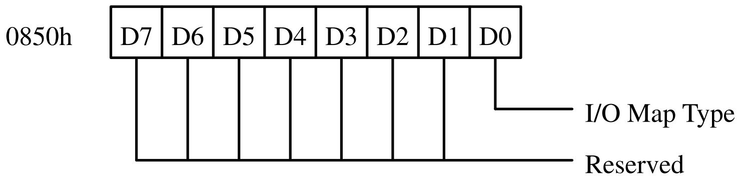
4.2.2.9 I/O Map Type BCR

Direct Access 8000 0850	Read/Write	Reset to 01h

This 8-bit, read/write register determines if the I/O mapping is contiguous or non–contiguous.

This register can be put into external register support mode so that CPU writes to this register are latched into the bridge BCR and are also forwarded to the PCI bus as a PCI I/O write to location 0000 0850h (the bridge arbitrates for the PCI bus and then executes a single byte, single data phase PCI I/O write). In this mode, reads to this register are always forwarded to the PCI (in the same manner), as I/O reads to location 0000 0850h, which allows system designers to externally add function to the reserved bit locations of this register. However, if this mode is used, the external logic must provide the proper read value of all bits including bit 0.

CPU writes to this register are not posted or pipelined to ensure that the results take effect before the CPU begins any other bus operations. This transaction is typically subtractively decoded by the I/O bus bridge for access to an external logic register. Other PCI bus masters are able to access the external registers, but during writes the data will not be latched into the 660 Bridge.



Bit 0 I/O Map Type: This bit can also be written by means of the bridge chip set options 1 register (indexed BAh, section 4.3.37)
0 = Non–Contiguous
1 = Contiguous.

4.2.2.10 PCI/BCR Configuration Address BCR

Direct Access 8000 0CF8	Read/Write	Reset to 0000 0000

This BCR is one of a pair of BCRs that are used to access the indexed BCRs and PCI configuration space, as discussed in section 4.3.1. This BCR is described in section 4.3.1.1.

4.2.2.11 PCI/BCR Configuration Data BCR

Direct Access 8000 0CFC to CFF	Read/Write	Reset: Undefined

This BCR is one of a pair of BCRs that are used to access the indexed BCRs and PCI configuration space, as discussed in section 4.3.1. This BCR is described in section 4.3.1.2.

4.2.2.12 PCI Type 0 Configuration Addresses

Direct Access 8080 08xx to 8084 00xx	Read/Write	Reset: Undefined

Accesses to these 650 Bridge compatible addresses cause the Bridge to run type 0 PCI configuration transactions. This is described in section 6.3.4.

4.2.2.13 System Error Address BCR

Direct Access BFFF EFF0	Read Only	Reset: Undefined

This 32-bit, read-only register indicates the address at which a parity or multi-bit ECC error or an illegal transfer error occurred. This register must be accessed by means of a 4-byte transfer.

Reading this register deasserts the memory parity error indicator (which can be read by means of the memory parity error status register port 8000 0840h and primary register C1h) and the illegal transfer error indicator (which can be read by means of the illegal transfer error register port 8000 0844 and primary register C1h).

4.2.2.14 Interrupt Acknowledge BCR

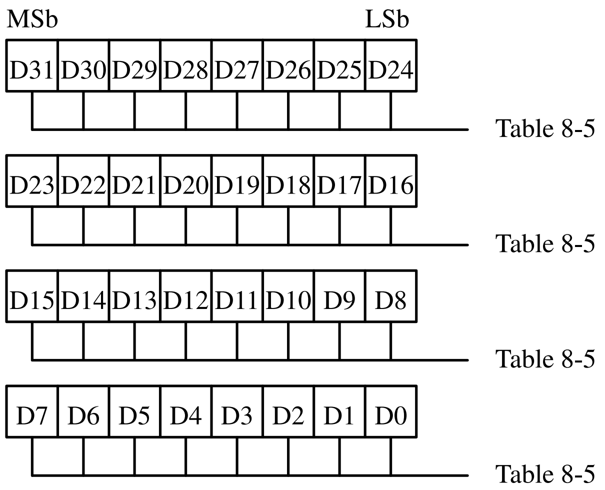
Direct Access BFFF FFF0	Read Only	Reset: Undefined

CPU accesses mapped to this virtual BCR cause the 660 Bridge to execute a PCI interrupt acknowledge transaction. The interrupt vector, which is supplied by the responding PCI agent, is forwarded to the CPU bus as the data read from the BCR. See section 6.3.5.

4.2.2.15 ROM Write Bridge Control BCR

Direct Access FFFF FFF0h	Write Only	Reset NA

This 32-bit, write-only register is used to program the ROM in direct-attach ROM systems (see section 8.2.2). This register must be written by means of a 4-byte transfer. Bits are shown with little-endian labels. See section 8 for details of ROM operations.

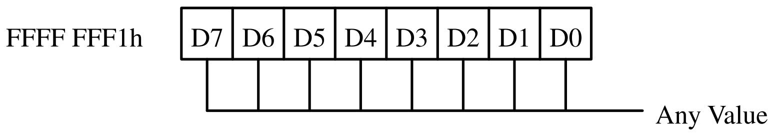


BCR Byte	Content in Little-Endian System	Content in Big-Endian System
MSB	ROM Data	ROM Address low byte
	ROM Address high byte	ROM Address mid byte
	ROM Address mid byte	ROM Address high byte
LSB	ROM Address low byte	ROM Data

4.2.2.16 ROM Lockout BCR (Direct-Attach ROM)

Direct Access FFFF FFF1h	Write Only	Reset NA

After it has been written once, this 8-bit, write-only register prevents ROM writes. This register is only used in direct-attach ROM systems. See section 8 for details of ROM operations.



Bits 7:0 Writing any value to the register prevents all future writes to a ROM that is connected directly to the 660 Bridge through the PCI_AD lines.

4.3 Indexed BCRs

The 660 Bridge features a large indexed BCR set, which allows extensive programmability of the Bridge and the system. Indexed BCR operations allow CPU bus masters to perform the following operations:

- Access the 660 Bridge internal indexed BCRs
- Initiate PCI type 0 configuration transactions
- Initiate PCI type 1 configuration transactions

The target registers of PCI agents that claim the PCI configuration transactions appear to the CPU to be virtual indexed BCRs.

4.3.1 Indexed BCR Access

The indexed BCRs shown in Table 4-2 are accessed by means of pairs of direct-access BCR accesses. The method of accessing them follows that described by the *PCI Local Bus Specification (Revision 2.0),* defined as Configuration Mechanism #1.

The first access of the pair, a direct-access BCR access (see Section 4.2.1.1) to the PCI/BCR configuration address BCR (8000 0CF8h), defines the destination of the transaction.

The second access of the pair, to the PCI/BCR configuration data BCR (8000 0CFCh), reads or writes the data and initiates the bus sequence. There is no physical PCI/BCR configuration data BCR. When the CPU accesses this BCR address, the bridge arbitrates for the PCI bus. After being granted the bus (or if the bus is parked on the CPU) the bridge executes either a PCI BCR transaction to an indexed BCR or a PCI configuration transaction to a PCI agent, depending on the contents of the bus and device fields of the PCI/BCR configuration address BCR.

Data written to the data BCR is forwarded to the actual target device or BCR over the PCI_AD lines during the data phase of the transaction. During reads, data supplied by the target device or BCR as sampled from the PCI_AD lines during the data phase of the transaction is returned to the CPU.

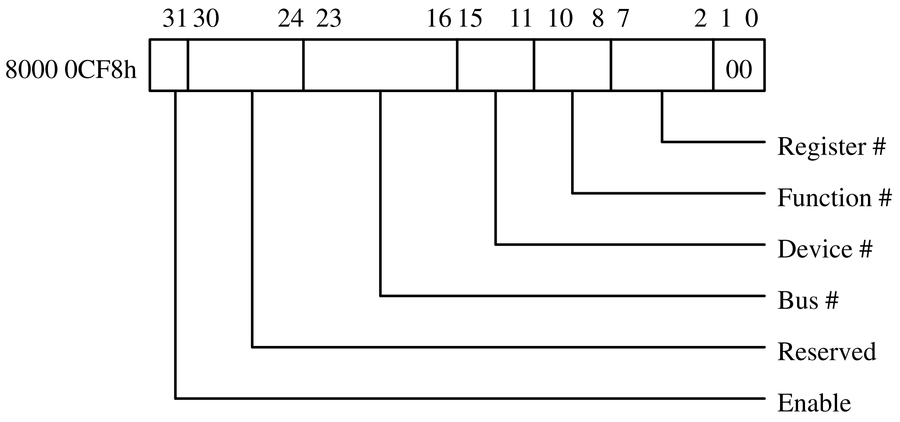
The 660 Bridge indexed BCRs should only be accessed individually during a PCI BCR transaction. For example, the 660 Bridge PCI command BCR (index 04h to 05h) can be accessed as a 2-byte object at index 04h, but the 660 Bridge revision ID BCR (1-byte) and the 660 Bridge PCI standard programming interface (1-byte) BCR should not be read together as a 2-byte object.

PCI configuration transactions can also be generated using these two BCRs and the appropriate configuration address information. Note that the 660 Bridge places no transfer size or alignment restrictions on PCI configuration transactions, other that they do not cross a 4-byte boundary.

4.3.1.1 PCI/BCR Configuration Address BCR

Direct Access 8000 0CF8	Read/Write	Reset to 0000 0000

This 32-bit read/write register is used as a pointer to PCI configuration registers during PCI configuration transactions and as a pointer to the selected indexed BCR during indexed BCR configuration transactions. The CPU must perform 4-byte transfers when accessing this BCR. Transfers of less than 4 bytes are mapped to PCI I/O space instead of accessing this BCR.



Bits 1:0 Reserved, always 00b for type 0 PCI configuration transactions and 01b for type 1 PCI configuration transactions.

Bits 7:2 Register Number: During PCI configuration transactions, these bits form the upper six bits of the PCI register number, which specifies which of the 256 possible registers within the PCI configuration space of a selected device is addressed. The least significant two bits are determined by the two least significant bits of the address of the accessed PCI/BCR configuration data BCR:

PCI/BCR Configuration Data BCR		Bits 3 and 2 of PCI/BCR Configuration Address BCR							
Address	2 lsb of index	00		01		10		11	
8000 0CFC	00	0000	B0	0100	B4	1000	B8	1100	BC
8000 0CFD	01	0001	B1	0101	B5	1001	B9	1101	BD
8000 0CFE	10	0010	B2	0110	B6	1010	BA	1110	BE
8000 0CFF	11	0011	B3	0111	B7	1011	BB	1111	BF

For example, if the upper four bits of the register are 1011b, bits 3:2 are 10b, and the data BCR access is to 8000 0CFEh, then indexed register BA is accessed. During indexed BCR accesses, the BCR index is also determined in this manner.

Bits 10:8 Function Number: During PCI configuration transactions, these bits specify which of eight functions within the PCI configuration space of a selected device is to be addressed. These bits are not used (don't cares) during indexed BCR accesses.

Bits 15:11 Device Number: During PCI configuration transactions, these bits specify which of 21 devices (or slots) is selected. As shown in the table, only one of the PCI_AD[31:11] bits is asserted for each valid value of bits 15:11. These bits are not used during indexed BCR accesses.

Bits 15:11	Asserted PCI_AD[]	Bits 15:11	Asserted PCI_AD[]	Bits 15:11	Asserted PCI_AD[]	Bits 15:11	Asserted PCI_AD[]
00000	none	00110	[16]	01100	[22]	10010	[28]
00001	[11]	00111	[17]	01101	[23]	10011	[29]
00010	[12]	01000	[18]	01110	[24]	10100	[30]
00011	[13]	01001	[19]	01111	[25]	10101	[31]
00100	[14]	01010	[20]	10000	[26]	other	none
00101	[15]	01011	[21]	10001	[27]		

Bits 23:16 Bus Number: During PCI configuration transactions, these bits specify which of 256 buses is selected. The PCI bus attached to the 660 Bridge is bus number 0.

If bus = 0 and device = 0, the bridge executes a BCR access to one of the indexed BCRs when the PCI/BCR configuration data BCR is accessed.

If bus = 0 and device > 0, the bridge masters a standard PCI type 0 configuration transaction when the PCI/BCR configuration data BCR is accessed.

If bus > 0, the bridge masters a standard PCI type 1 configuration transaction when the PCI/BCR configuration data BCR is accessed.

Bit 31 Enable: This bit must be a 1 to enable accesses to either the the PCI configuration registers or the indexed BCRs.

4.3.1.2 PCI/BCR Configuration Data BCR

Direct Access 8000 0CFC	Read/Write	Reset: Undefined

31 0

8000 0CFCh

This 32-bit read/write virtual register is used to access the register pointed to by the PCI/BCR configuration address BCR. When the CPU accesses this BCR, the bridge arbitrates for the PCI bus. After being granted the bus (or if the bus is parked on the CPU) the bridge executes either a BCR access or a PCI configuration transaction, depending on the contents of the bus and device fields of the PCI/BCR Configuration Address BCR.

During write transactions, the data written to this BCR is driven onto the PCI_AD lines during the data phase. During read transactions, the data that the CPU receives from this BCR is the data sampled during the data phase of the bus transaction. There is no physical PCI/BCR configuration data BCR.

4.3.2 Indexed BCR Summary

Table 4-2 contains a summary listing of the indexed BCRs. Accesses to these registers are described in Section 4.2.1.1.

Table 4-2. Indexed BCR Listing

Bridge Control Register	Index	R/W	Bytes	See
PCI Vendor ID	Index 00 – 01	R	2	4.3.3
PCI Device ID	Index 02 – 03	R	2	4.3.4
PCI Command	Index 04 – 05	R/W	2	4.3.5
PCI Device Status	Index 06 – 07	R/W	2	4.3.6
Revision ID	Index 08	R	1	4.3.7
PCI Standard Programming Interface	Index 09	R	1	4.3.8
PCI Subclass Code	Index 0A	R	1	4.3.9
PCI Class Code	Index 0B	R	1	4.3.10
PCI Cache Line Size	Index 0C	R	1	4.3.11
PCI Latency Timer	Index 0D	R	1	4.3.12
PCI Header Type	Index 0E	R	1	4.3.13
PCI Built-in Self-Test (BIST) Control	Index 0F	R	1	4.3.14
PCI Interrupt Line	Index 3C	R	1	4.3.15
PCI Interrupt Pin	Index 3D	R	1	4.3.16
PCI MIN_GNT	Index 3E	R	1	4.3.17
PCI MAX_LAT	Index 3F	R	1	4.3.18
PCI Bus Number	Index 40	R	1	4.3.19
PCI Subordinate Bus Number	Index 41	R	1	4.3.20
PCI Disconnect Counter	Index 42	R/W	1	4.3.21
PCI Special Cycle Address BCR	Index 44 –45	R	2	4.3.22
Memory Bank 0 Starting Address	Index 80	R/W	1	4.3.23
Memory Bank 1 Starting Address	Index 81	R/W	1	4.3.23
Memory Bank 2 Starting Address	Index 82	R/W	1	4.3.23
Memory Bank 3 Starting Address	Index 83	R/W	1	4.3.23
Memory Bank 4 Starting Address	Index 84	R/W	1	4.3.23
Memory Bank 5 Starting Address	Index 85	R/W	1	4.3.23
Memory Bank 6 Starting Address	Index 86	R/W	1	4.3.23
Memory Bank 7 Starting Address	Index 87	R/W	1	4.3.23
Memory Bank 0 Extended Starting Address	Index 88	R/W	1	4.3.23.1
Memory Bank 1 Ext Starting Address	Index 89	R/W	1	4.3.23.1

Table 4-2. Indexed BCR Listing (Continued)

Bridge Control Register	Index	R/W	Bytes	See
Memory Bank 2 Ext Starting Address	Index 8A	R/W	1	4.3.23.1
Memory Bank 3 Ext Starting Address	Index 8B	R/W	1	4.3.23.1
Memory Bank 4 Ext Starting Address	Index 8C	R/W	1	4.3.23.1
Memory Bank 5 Ext Starting Address	Index 8D	R/W	1	4.3.23.1
Memory Bank 6 Ext Starting Address	Index 8E	R/W	1	4.3.23.1
Memory Bank 7 Ext Starting Address	Index 8F	R/W	1	4.3.23.1
Memory Bank 0 Ending Address	Index 90	R/W	1	4.3.24
Memory Bank 1 Ending Address	Index 91	R/W	1	4.3.24
Memory Bank 2 Ending Address	Index 92	R/W	1	4.3.24
Memory Bank 3 Ending Address	Index 93	R/W	1	4.3.24
Memory Bank 4 Ending Address	Index 94	R/W	1	4.3.24
Memory Bank 5 Ending Address	Index 95	R/W	1	4.3.24
Memory Bank 6 Ending Address	Index 96	R/W	1	4.3.24
Memory Bank 7 Ending Address	Index 97	R/W	1	4.3.24
Memory Bank 0 Extended Ending Address	Index 98	R/W	1	4.3.25
Memory Bank 1 Ext Ending Address	Index 99	R/W	1	4.3.25
Memory Bank 2 Ext Ending Address	Index 9A	R/W	1	4.3.25
Memory Bank 3 Ext Ending Address	Index 9B	R/W	1	4.3.25
Memory Bank 4 Ext Ending Address	Index 9C	R/W	1	4.3.25
Memory Bank 5 Ext Ending Address	Index 9D	R/W	1	4.3.25
Memory Bank 6 Ext Ending Address	Index 9E	R/W	1	4.3.25
Memory Bank 7 Ext Ending Address	Index 9F	R/W	1	4.3.25
Memory Bank Enable	Index A0	R/W	1	4.3.26
Memory Timing 1	Index A1	R/W	1	4.3.27
Memory Timing 2	Index A2	R/W	1	4.3.28
Memory Bank 0 & 1 Addressing Mode	Index A4	R/W	1	4.3.29
Memory Bank 2 & 3 Addressing Mode	Index A5	R/W	1	4.3.29
Memory Bank 4 & 5 Addressing Mode	Index A6	R/W	1	4.3.29
Memory Bank 6 & 7 Addressing Mode	Index A7	R/W	1	4.3.29
Cache Status	Index B1	R/W	1	4.3.30
Refresh Cycle Definition	Index B4	R	1	4.3.31
Refresh Timer B5 (Not used – see Indexed BCR D0)	Index B5	R	1	4.3.32
RAS Watchdog Timer	Index B6	R/W	1	4.3.33

4

Table 4-2. Indexed BCR Listing (Continued)

Bridge Control Register	Index	R/W	Bytes	See
PCI Bus Timer (Not used)	Index B7	R	1	4.3.34
Single-Bit Error Counter	Index B8	R/W	1	4.3.35
Single-Bit Error Trigger Level	Index B9	R/W	1	4.3.36
Bridge Options 1	Index BA	R/W	1	4.3.37
Bridge Options 2	Index BB	R/W	1	4.3.38
Error Enable 1	Index C0	R/W	1	4.3.39
Error Status 1	Index C1	R/W	1	4.3.40
Error Simulation 1	Index C2	R/W	1	4.3.41
CPU Bus Error Status	Index C3	R	1	4.3.42
Error Enable 2	Index C4	R/W	1	4.3.43
Error Status 2	Index C5	R/W	1	4.3.44
Error Simulation 2	Index C6	R/W	1	4.3.45
PCI Bus Error Status	Index C7	R/W	1	4.3.46
CPU/PCI Error Address	Index C8 – CB	R/W	4	4.3.47
Single-Bit ECC Error Address	Indx CC – CF	R/W	4	4.3.48
Refresh Timer Divisor	Index D0 – D1	R/W	2	4.3.49
Suspend Refresh Timer	Index D2 – D3	R/W	2	4.3.50
Bridge Chip Set Options 3	Index D4	R/W	1	4.3.51

4.3.3 PCI Vendor ID Register

Index 00 to 01h	Read Only	Reset to 1014h

The vendor ID register is a 16-bit, read-only BCR used to identify the manufacturer of the 660 Bridge. Reading this register always returns 1014h (index 00h = 14h, index 01h = 10h). This is the vendor ID assigned for all IBM–produced PCI devices.

4.3.4 PCI Device ID Register

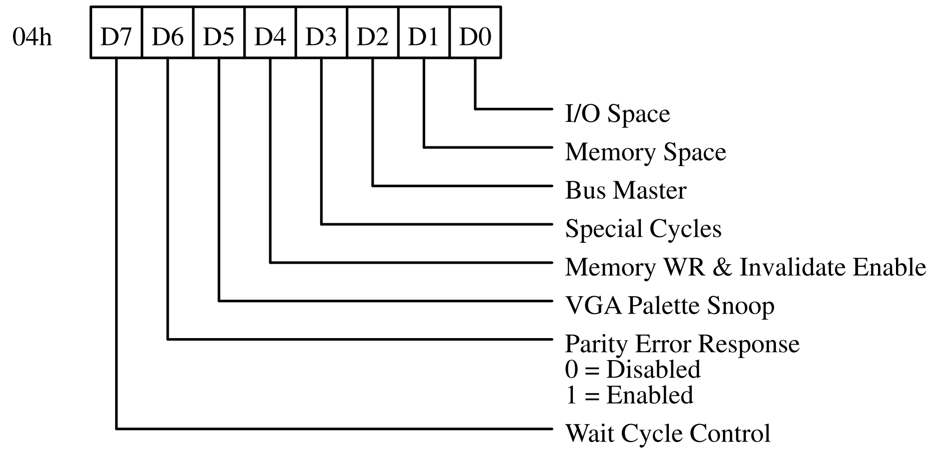
Index 02 to 03h	Read Only	Reset to 0037h

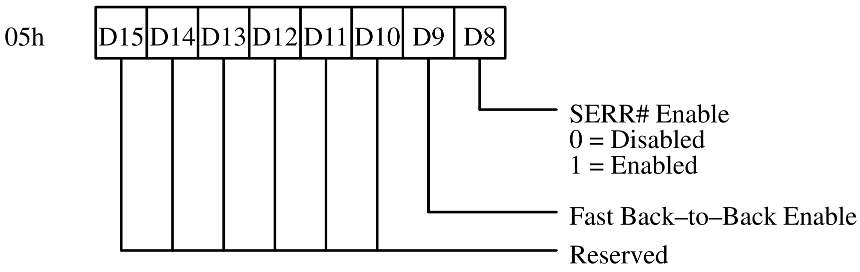
The device ID register is a 16-bit, read-only BCR used to identify the 660 Bridge. Reading this register always returns 0037h (index 02h = 37h, index 03h = 00h).

4.3.5 PCI Command Register

Index 04 to 05	Read/Write	Reset to 0006h

The PCI command register is a 16-bit, read/write BCR used to control the operation of the 664 Controller on the PCI bus.





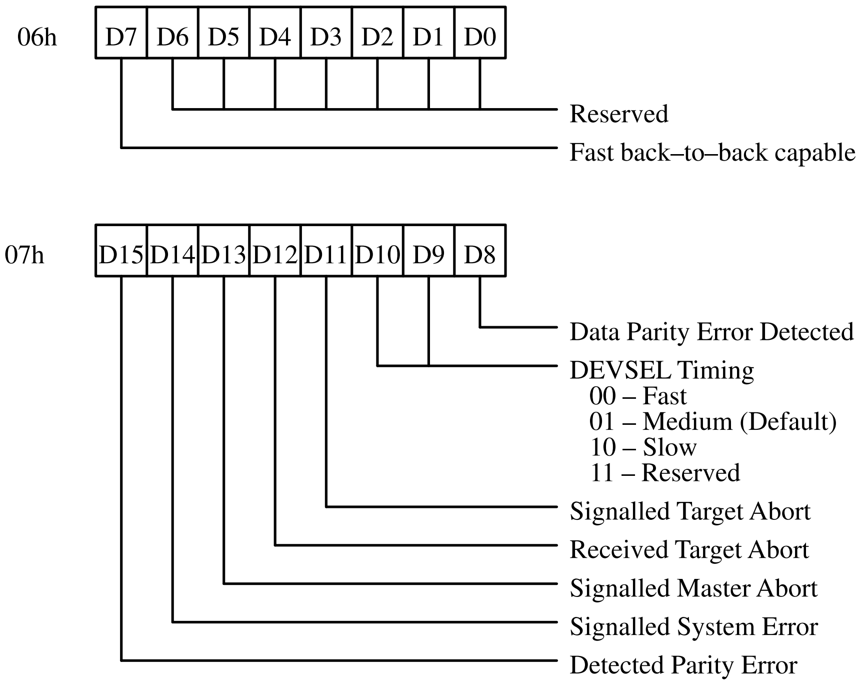
Bit 0	Controls the response of the 660 Bridge to I/O space accesses. The 660 Bridge does not respond to the I/O address space; therefore, this bit is read-only and always returns 0 when read.
Bit 1	Controls the response of the 660 Bridge to memory space accesses. The 660 Bridge always responds to the memory address space; therefore, this bit is read-only and always returns 1 when read.
Bit 2	Enables the 660 Bridge to master cycles on the PCI bus. The 660 Bridge is always enabled as a PCI bus master; therefore, this bit is read-only and returns 1 when read.

Bit 3	Enable special cycle operations. The 660 Bridge does not respond to special cycle operations; therefore, this bit is read-only and returns 0 when read.
Bit 4	Enable memory write and invalidate command support. The 660 Bridge does not support caching on the PCI bus; therefore, this bit is read-only and returns 0 when read.
Bit 5	Enable special palette snooping. The 660 Bridge is not a VGA device; therefore, this bit is read-only and returns 0 when read.
Bit 6	Parity error response. This bit is enabled for all types of PCI bus parity errors, including the following: 1) PCI data bus parity errors while PCI master 2) PCI data bus parity errors while PCI target 3) PCI address bus parity errors When parity error response is disabled (set to 0) detection of these errors is masked.
Bit 7	Address stepping wait states. The 660 Bridge does not require the use of AD stepping; therefore, this bit is read-only and returns 0 when read.
Bit 8	Enable PCI_SERR#. Enables driving PCI_SERR# when PCI address bus parity error is detected. PCI command register bit 6 must also be enabled.
Bit 9	Fast back-to-back write enable. The 660 Bridge does not support fast back-to-back write cycles to different devices on the PCI bus; therefore, this bit is read-only and returns 0 when read.
Bits 15:10	Reserved, returns 000000b when read.

4.3.6 PCI Device Status Register

Index 06 to 07h	Read/Bit reset	Reset to 0200h

This 16-bit, read/bit–reset BCR records status information for PCI-related events. Bits in this register can only be set as a result of a specific event occurring on the PCI bus, and they can only be reset by the CPU. To reset any bit, a 1 is written to the specific bit location. A 0 written to a specific bit location leaves that bit unaltered.



Bits 6:0 Reserved, returns 0000000b when read.

Bit 7 Fast back-to-back capable. This bit is read-only and is a 0 to indicate that fast back-to-back cycles are not supported when the 660 Bridge is the PCI target. For example, when a PCI master accesses system memory.

Bit 8 Data parity error detected. This bit is set if both of the following conditions are true:
1. The 660 Bridge is the master when a data bus parity error is detected during a PCI read cycle, or the 660 Bridge is the master when PCI_PERR# is sampled active during a PCI write cycle.
2. PCI command register bit 6 is set to 1.

If this bit is set, a TEA#/MCP# to the processor is generated if enabled. Writing a 1 to this bit clears this bit to 0.

Bits 10:9 PCI_DEVSEL# response timing. The 660 Bridge asserts PCI_DEVSEL# in the second clock following a PCI_FRAME# generated by a PCI bus master attempting to access memory. These bits always return 01.

Bit 11 This bit is set whenever the 660 Bridge terminates a PCI cycle for which it is the target with target abort.

Bit 12 This bit is set whenever a PCI cycle for which the 660 Bridge is the master is terminated with target abort.

Writing a 1 to this bit clears this bit to 0.

Bit 13 This bit is set whenever the 660 Bridge terminates a PCI cycle for which it is the master with master abort during a PCI memory or I/O cycle.

Writing a 1 to this bit clears this bit to 0.

Bit 14 Signalled system error. If this bit is set, the 660 Bridge asserts PCI_SERR#. This bit is set if both of the following conditions are true:
1) The PCI_SERR# enable bit is set (PCI command regsiter bit 8).
2) Certain errors are detected during a PCI transaction while the Bridge is the PCI target. Not all errors cause this bit to be set. See Section 9 for details.

Writing a 1 to this bit clears this bit to 0.

Bit 15 Detected parity error. This bit is set whenever the 660 Bridge detects a PCI bus parity error. This bit is not maskable—if an error is detected, this bit is set regardless of any control bits. The following events set this bit:
1) PCI address bus parity error detected when an external PCI master accesses system memory.
2) PCI data bus parity error detected when an external PCI master writes to system memory.
3) PCI data bus parity error detected when the 660 bridge masters a PCI read cycle.

Writing a 1 to this bit clears this bit to 0.

4.3.7 Revision ID

Index 08h	Read Only	Reset to 00h

The revision ID register is an 8-bit, read-only BCR used to hold the current incremental revision number of the 660 Bridge. For the first revision, this register returns 00h.

4

4.3.8 PCI Standard Programming Interface

Index 09h	Read Only	Reset to 00h

The PCI standard programming interface register is an 8-bit, read-only BCR used to hold programing interface information. Since the 660 Bridge does not support a programming interface, this register returns 00h.

4.3.9 PCI Subclass Code

Index 0Ah	Read Only	Reset to 00h

The PCI subclass code register is an 8-bit, read-only BCR used to hold device class information. This register returns 00h, indicating that the 660 Bridge is a host bridge.

4.3.10 PCI Class Code

Index 0Bh	Read Only	Reset to 06h

The PCI class code register is an 8-bit, read-only BCR used to hold device class information. This register returns 06h, indicating that the 660 Bridge is a bridge between the PCI bus and the 60X CPU bus.

4.3.11 PCI Cache Line Size

Index 0Ch	Read Only	Reset to 00h

The PCI cache line size register is an 8-bit, read-only BCR used to hold the size of a PCI cache line. Since the 660 Bridge does not support a PCI cache, this register returns 00h.

4.3.12 PCI Latency Timer

Index 0Dh	Read Only	Reset to 00h

The PCI latency timer register is an 8-bit, read-only BCR used to hold the value of the PCI latency timer. Because the 660 Bridge can only burst two PCI cycles as master (only with the 604 store multiple instruction), this timer is not implemented and this register returns 00h.

4.3.13 PCI Header Type

Index 0Eh	Read Only	Reset to 00h

The PCI header type register is an 8-bit, read-only BCR used to describe the header region of PCI configuration space. The 660 Bridge implements a standard PCI header and this register returns 00h.

4

4.3.14 PCI Built-in Self-Test (BIST) Control

Index 0Fh	Read Only	Reset to 00h

The PCI built–in self-test control register is an 8-bit, read-only BCR used for control and status of BIST. The 660 Bridge does not implement BIST and this register returns 00h.

4.3.15 PCI Interrupt Line

Index 3Ch	Read Only	Reset to 00h

The PCI interrupt line register is an 8-bit, read-only BCR used for PCI interrupt routing information. The 660 Bridge does not implement this feature and this register returns 00h.

4.3.16 PCI Interrupt Pin

Index 3Dh	Read Only	Reset to 00h

The PCI interrupt pin register is an 8-bit, read-only BCR used to indicate the PCI interrupt pin to be used. The 660 Bridge does not generate PCI interrupts and this register returns 00h.

4.3.17 PCI MIN_GNT

Index 3Eh	Read Only	Reset to 00h

The PCI MIN_GNT register is an 8-bit, read-only BCR used to specify the minimum setting for the PCI latency timer. The 660 Bridge does not implement the PCI latency timer and this register returns 00h.

4

4.3.18 PCI MAX_LAT

Index 3Fh	Read Only	Reset to 00h

The PCI MAX_LAT register is an 8-bit, read-only BCR used to specify the maximum setting for the PCI latency timer. The 660 Bridge does not implement the PCI latency timer and this register returns 00h.

4.3.19 PCI Bus Number

Index 40h	Read Only	Reset to 00h

The PCI bus number register is an 8-bit, read-only BCR used to identify the number of the bus controlled by the 660 Bridge. This register returns 00h indicating that the 660 Bridge controls PCI bus number 0. Peer bridges are not supported (for example, a second 60X to PCI bridge on the 60X CPU bus); therefore, this register value is read-only. All CPU accesses to CONFIG_DATA when CONFIG_ADDRESS specifies the bus number as 0 result in the 660 Bridge running a type 0 configuration cycle on its PCI bus.

4.3.20 PCI Subordinate Bus Number

Index 41h	Read Only	Reset to 00h

The PCI subordinate bus number register is an 8-bit, read-only BCR. Since the 660 Bridge does not support peer PCI bridges (only hierarchical), this register is unused. All CPU accesses to CONFIG_DATA when the CONFIG_ADDRESS specifies a bus number other than 0 cause the 660 Bridge to run a type 1 configuration cycle on its PCI bus.

4.3.21 PCI Disconnect Counter

Index 42h	Read/Write	Reset to 00

This BCR determines the maximum number of PCI clocks (from 1 to 255) that a PCI master can burst access system memory before a target disconnect is initiated. Loading the counter with 00 disables it. The counter begins to count down when FRAME# is sampled active, and is reloaded between PCI to memory accesses. If the counter reaches 0, the 660 Bridge target disconnects the PCI bus master.

4.3.22 PCI Special Cycle Address Register

Index 44 to 45h	Read Only	Reset to 0000h

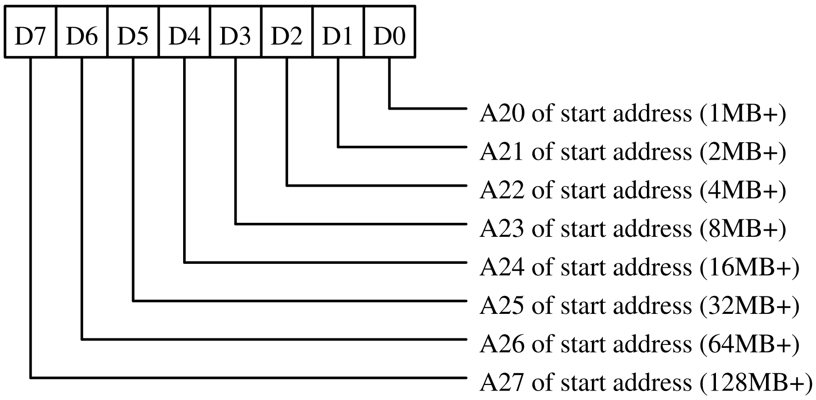
The PCI special cycle address register is a 16-bit, read-only BCR. It is not implemented and returns 0000h.

4.3.23 Memory Bank Starting Address

Index 80 to 87h	Read/Write	Reset to 00h (each BCR)

This array of eight BCRs (along with the eight extended starting address registers) contains the starting address for each memory bank. Each pair of registers maps to the corresponding RAS# decode. For example, RAS[4]# corresponds to the BCRs at index 84h and 8Ch. The eight least-significant bits of the bank starting address are contained in the starting address register, and the most-significant bits come from the corresponding extended starting address register. The starting address of the bank is entered with the least significant 20 bits truncated. These BCRs must be programmed in conjunction with the ending address and extended ending address registers.

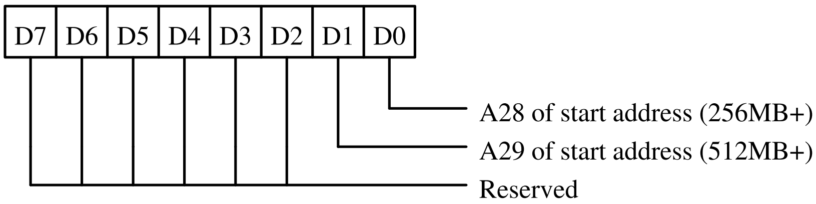
Program the banks in ascending order, such that (for n = 0 to 6) the starting address of bank n+1 is higher than the starting address of bank n. Each bank must be located in the 0 to 1G address range (see section 5.3.12).



4.3.23.1 Memory Bank Extended Starting Address

Index 88 to 8Fh	Read/Write	Reset to 00h (each BCR)

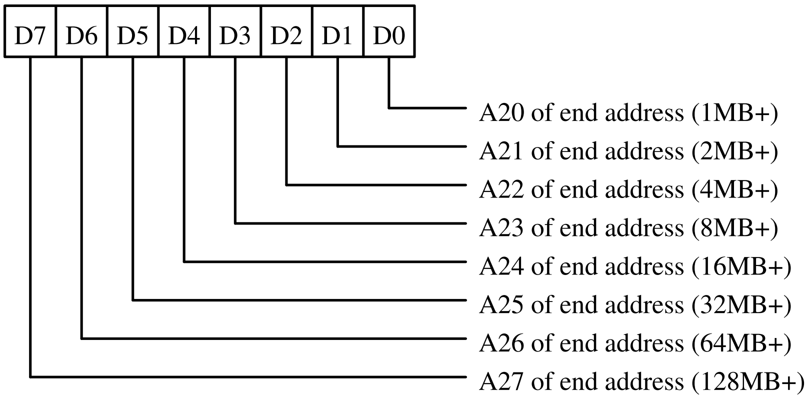
This array of eight BCRs (along with the eight starting address registers) contains the starting address for each memory bank. These BCRs contain the most-significant address bits of the starting address of the corresponding bank.



4.3.24 Memory Bank Ending Address

Index 90 to 97h	Read/Write	Reset to 00h (each BCR)

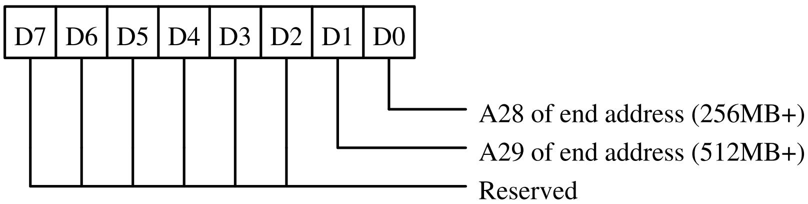
This array of eight BCRs (along with the eight extended starting address registers) contains the ending address for each memory bank. Each pair of registers maps to the corresponding RAS# decode. For example, RAS[4]# corresponds to the BCRs at index 94h and 9Ch. The eight least-significant bits of the bank ending address are contained in the ending address register, and the most-significant bits come from the corresponding extended ending address register. The ending address of the bank is entered as the address of the next highest memory location minus 1, with the least significant 20 bits truncated. Each bank must be located in the 0 to 1G address range. These BCRs must be programmed in conjunction with the ending address and extended ending address registers. See section 5.3.12.



4.3.25 Memory Bank Extended Ending Address

Index 98 to 9Fh	Read/Write	Reset to 00 (each BCR)

This array of eight 8-bit, read/write registers (along with the eight ending address registers) contains the ending address for each memory bank. These BCRs contain the most-significant address bits of the ending address of its bank.

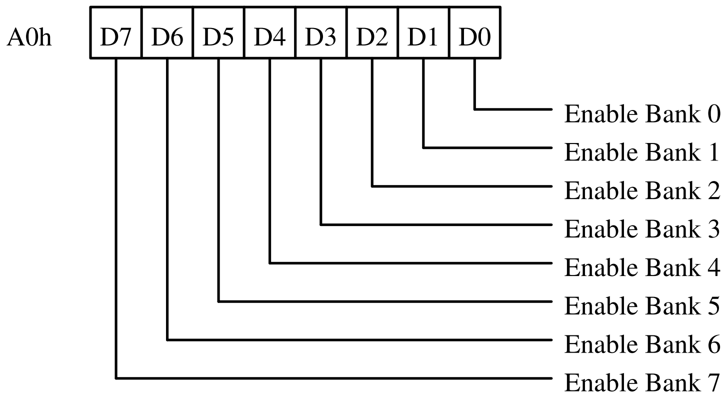


4.3.26 Memory Bank Enable

Index A0h	Read/Write	Reset to 00h

This BCR contains a control enable for each bank of memory. Each bank of memory must be enabled for proper refreshing. For each bit, a 0 disables that bank of memory and a 1 enables it.

This register must be programmed in conjunction with the starting address and ending address registers. If a bank is disabled by this register, the corresponding starting and ending address register entries become don't cares.



4.3.27 Memory Timing Register 1

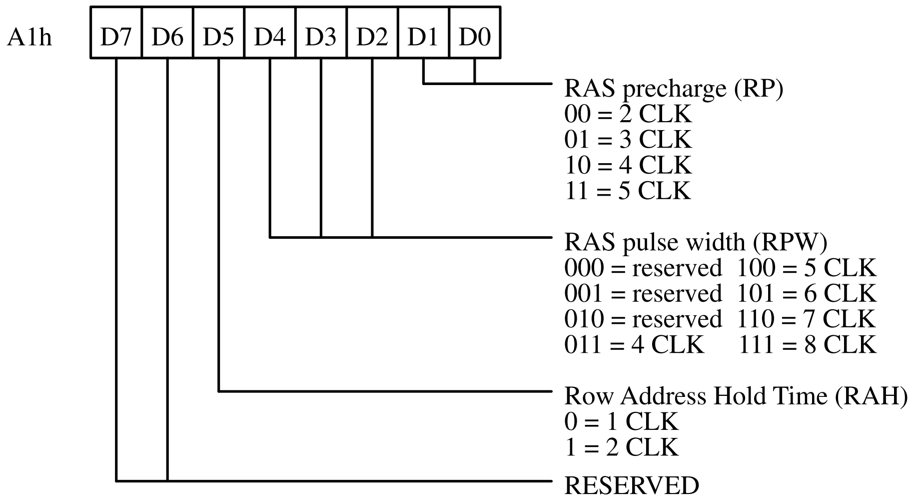
Index A1h	Read/Write	Reset to 3Fh

This BCR determines the timing of RAS# signal assertion for memory cycles. RAS# timing must support the worst-case timing for the slowest SIMM installed in the system. See Section 5.2.4.

Bits 1:0 These bits control the number of CPU clocks for RAS# precharge.

Bits 4:2 These bits control RAS pulse width except on refresh. For refresh, the RAS# pulse width is hard-coded to three PCI clocks.

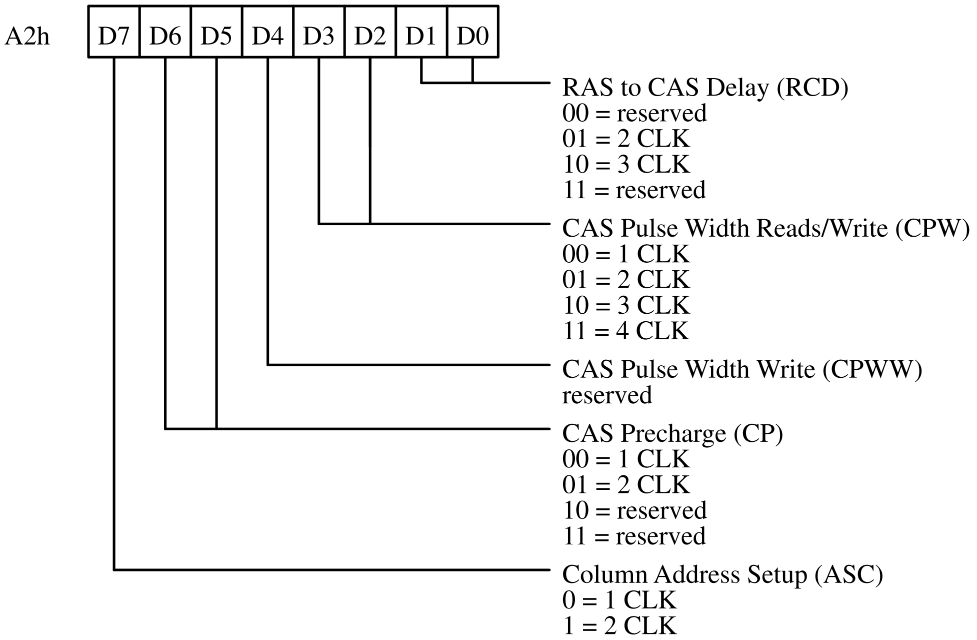
Bit 5 This bit controls the number of CPU clocks that the row address is held following the assertion of RAS#.



4.3.28 Memory Timing Register 2

Index A2h	Read/Write	Reset to AEh

This BCR determines the timing of CAS# signal assertion for memory cycles. CAS# timing must support the worst-case timing for the slowest DRAM installed in the system. See Section 5.2.4.

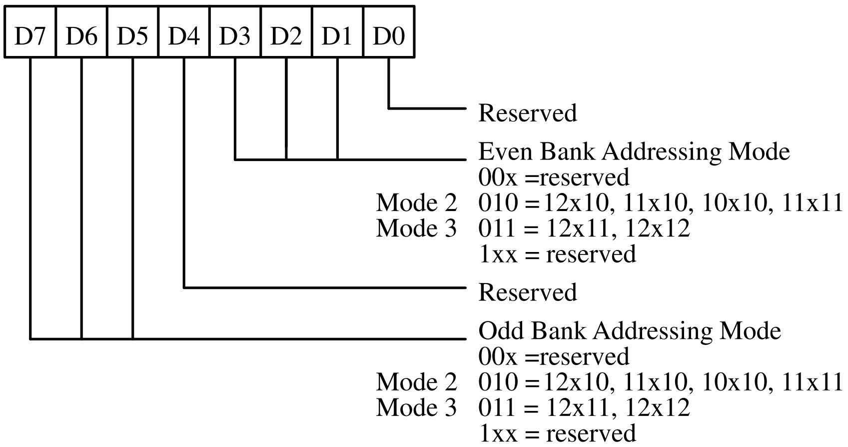


4

4.3.29 Memory Bank Addressing Mode Registers

Index A4 to A7h	Read/Write	Reset to 44h (each BCR)

This array of four 8-bit, read/write BCRs defines the format of the row and column addressing of each DRAM memory bank.

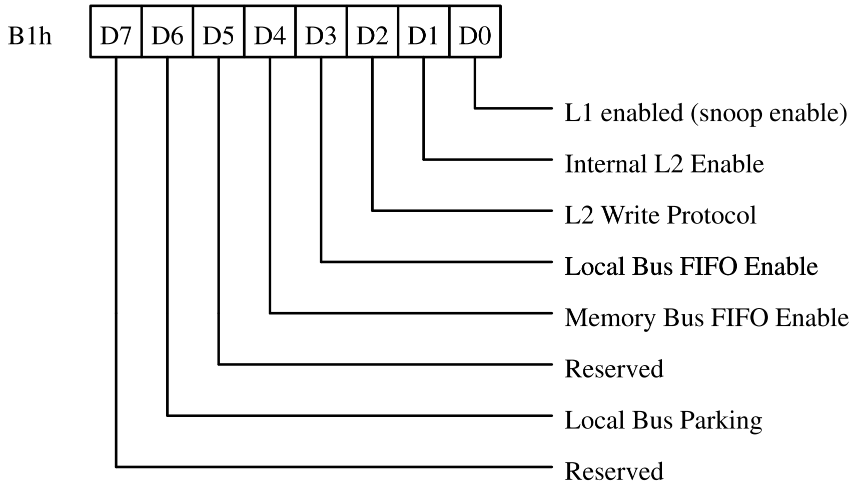


Register	Bits	Memory Bank	Bits	Memory Bank
A4h	3:0	0	7:4	1
A5h	3:0	2	7:4	3
A6h	3:0	4	7:4	5
A7h	3:0	6	7:4	7

4.3.30 Cache Status Register

Index B1h	Read/Write	Reset to 43h

This 8-bit, read/write BCR describes the status of the level one (L1) and level two (L2) caches and other features.



Bit 0 L1 enabled: Snooping is always enabled and this bit is hardcoded to a 1.
0 : L1 disabled
1 : L1 enabled

Bit 1 Internal L2 Enable: This bit is used with the L2 Cache Enable bit (bit 6 of the system control 81C BCR, section 4.2.2.3) to enable the internal L2 cache. The internal L2 is enabled only when both bits are set to 1.
0 : Internal L2 disabled.
1 : Internal L2 enabled (iff L2 Cache Enable bit is set).

Bit 2 L2 write protocol: The 660 Bridge L2 is write-through only and this bit is hardcoded to a 0 (this does not prevent an external L2 cache from being writeback).
0 : L2 cache is write-through
1 : L2 cache is writeback (store-in)

Bit 3 60X CPU bus FIFO enable: There is no 60X CPU bus FIFO and this bit is hardcoded to a 0.
0 : Disabled
1 : Enabled

Bit 4 Memory bus FIFO enable: There is no memory FIFO and this bit is hardcoded to a 0.
0 : Disabled
1 : Enabled

Bit 6 60X CPU bus parking: The 660 Bridge always parks the 60X CPU bus on the first CPU and this bit is hardcoded to a 1.
0 : Disabled
1 : Enabled

4

4.3.31 Refresh Cycle Definition Register

Index B4h	Read only	Reset to 07h

This BCR always returns 07h. The 660 Bridge supports CAS# before RAS# refresh only. Refresh is always enabled.

4.3.32 Refresh Timer Register B5 (not used)

Index B5h	Read/Write	Reset to 00h

This BCR always returns 00h. The refresh timer is programmed from the refresh timer divisor register (index D0).

4.3.33 RAS Watchdog Timer Register

Index B6h	Read/Write	Reset to 53h

This BCR limits the maximum RAS# active pulse width. The value of this BCR represents the maximum amount of time that any RAS# can remain active in units of eight CPU bus clocks. The timer (down-counter) associated with this BCR is reloaded on the assertion of any RAS# line. On expiration of the timer, the 660 Bridge drops out of page mode to deassert the RAS# lines.

In response to the RESET# signal, this register is reset to 53h. This value results in a maximum RAS# active time of just under 10us at 66MHz. This is the value required by most 4-byte and 8-byte SIMMs. The value of the BCR must be reprogrammed if the CPU bus frequency is not 66MHz or a different RAS# pulse width is required.

4.3.34 PCI Bus Timer Register (Not used)

Index B7h	Read	Reset to 00h

This BCR always returns 00h.

4.3.35 Single-Bit Error Counter Register

Index B8h	Read/Write	Reset to 00h

This 8-bit, read/write BCR contains the count of the number of ECC single-bit errors that have occurred. This register can be written by the CPU to set or clear the counter value. If a value greater than the trigger level register (see Section 4.3.36) is written, the TEA# or MCP# error is asserted.

4.3.36 Single-Bit Error Trigger Level Register

Index B9h	Read/Write	Reset to 00h

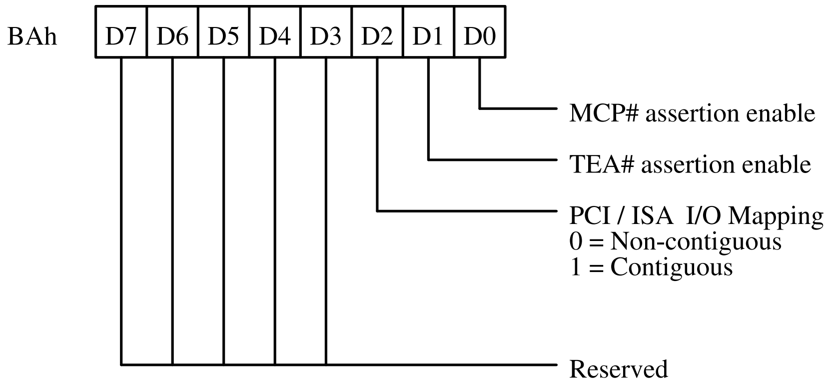
This 8-bit, read/write BCR contains the threshold value for generating an error to the CPU on ECC single-bit errors. When the single-bit error counter register value equals the value in this register, a single-bit ECC error (trigger exceeded) is generated to the CPU.

If the single-bit error trigger level register is set to 00h, no error is ever generated to the CPU.

4.3.37 Bridge Chip Set Options 1

Index BAh	Read/Write	Reset to 00h

This 8-bit, read/write BCR controls various operating parameters of the 660 Bridge.



Bit 0 MCP# assertion enabled: When set, the 664 Controller will assert the machine check pin to the 603/604 CPU or drive an interrupt to the 601 CPU with intent to error terminate. The state of this bit does not affect the operation of the error detection and handling logic in any other way.

0 : MCP# assertion disabled: In response to a system error, the 664 Controller will neither assert MCP# (603/604) nor generate an interrupt (601) with the intent to error terminate.

1 : MCP# assertion enabled: In response to a system error, the 664 Controller will assert MCP# (603/604) or generate an interrupt (601) with the intent to error terminate.

Bit 1 TEA# assertion enabled: When enabled, the 664 will assert TEA# when an error condition is detected. Some error conditions will cause a bus hang if the cycle is not terminated (for example. XATS# assertion) so the 664 Controller still terminates such cycles with TEA# to prevent a bus hang condition, even if this bit is set to 0. Note that this bit can also be written by means of the system control 8000 081Ch BCR.

1 : TEA# assertion enabled.

Bit 2 PCI/ISA I/O mapping. This bit can also be written by means of the I/O map type register (address 8000 0850h).

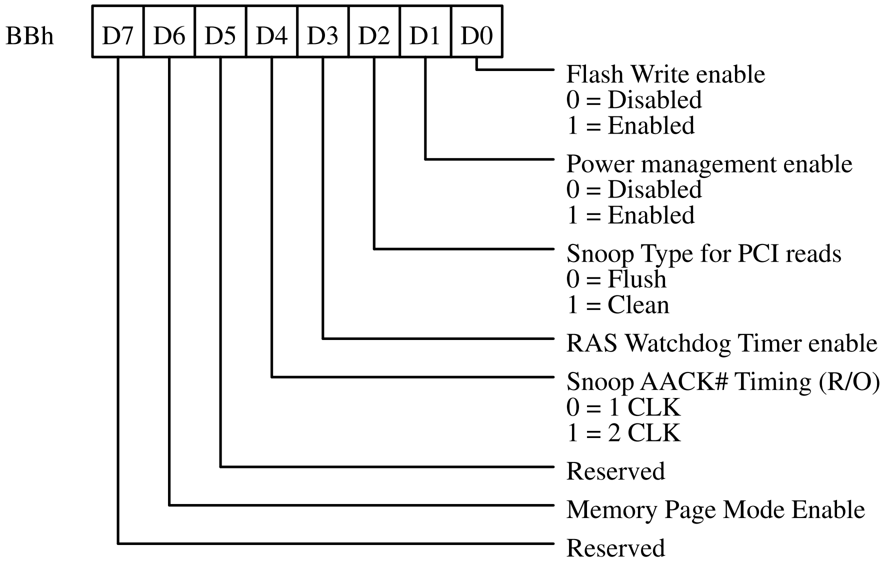
0 : Non–contiguous PCI / ISA address mapping is used. CPU addresses from 2G to 2G + 8M are translated where each 4K of CPU address space is mapped to 32 bytes. The 8M address space is compressed into 64K.

1 : Contiguous PCI/ISA address mapping is used. CPU addresses from 2G to 2G + 8M are not remapped.

Note that the state of bits 0 and 1 do not control MCP mode. These bits merely enable or disable the assertion of particular outputs. For more information on MCP mode, see section 4.3.39 Bridge Chipset Options 3.

4.3.38 Bridge Chip Set Options 2

Index BBh	Read/Write	Reset to 4Fh



Bit 0 Flash write enable: When the ROM is remotely attached, this bit controls write access to the flash ROM address space (4G – 2M to 4G). When enabled, writes to this space are forwarded to the PCI memory space at the same address. When disabled, writes to this space are treated as no-ops and an error is signalled. After the bit is set to 0 (disabled), it cannot be reset to 1 (enabled).

Bit 1 Power management enable: This bit is a mask of the STOP_CLK_EN# pin. When disabled, STOP_CLK_EN# is masked and suspend mode is never entered.

Bit 2 Snoop type generated for PCI reads to system memory: This bit is hardcoded as 1 since read snoops are always clean bus operations which result in pushing out modified data but not invalidating the cache line.

Bit 3 RAS watchdog timer enable: The RAS max pulse width is always checked and this bit is hardcoded to 1.

Bit 4 AACK# timing: Controls the minimum number of cycles between assertion of TS# (transfer start) and assertion of AACK# (address acknowledge) by the 660 Bridge for any cycle. The implementation of a 603 processor with its internal logic clocked at the same rate as the 60X CPU bus (1:1) requires this bit to be set. This bit is read-only and is set to the value defined by the configuration strapping options. (Reset value is determined by strapping pin.)

Bit 5 Reserved.

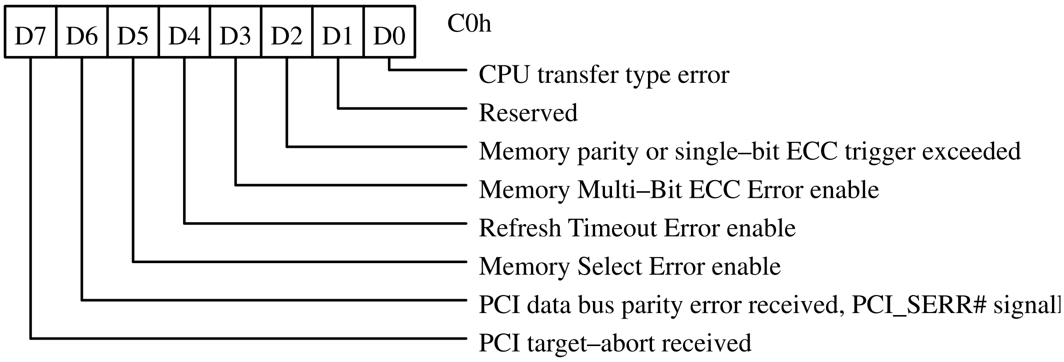
Bit 6 Memory page mode enable. Page mode operation is always enabled. This bit is hardcoded to a 1 (page mode enabled).

Bit 7 Reserved.

4.3.39 Error Enable 1

Index C0h	Read/Write	Reset to 01h

The error report enable 1 register is an 8-bit read/write BCR that selects the system error conditions to which the 660 Bridge responds. A bit set in this register enables the associated error condition to be detected.



Bit 0 CPU transfer type error enable: This bit enables the detection of unsupported transfer types and XATS# assertion.
0 : Disabled
1 : Enabled

Bit 2 Memory parity or single-bit ECC error trigger exceeded enable:
0 : Disabled
1 : Enabled

Bit 3 Memory multi-bit ECC error enable:
0 : Disabled
1 : Enabled

Bit 4 Refresh timeout error enable: The 660 Bridge does not allow refresh timeout errors to occur and this bit is hardcoded as a 0.

Bit 5 Memory select error enable:
0 : Disabled
1 : Enabled

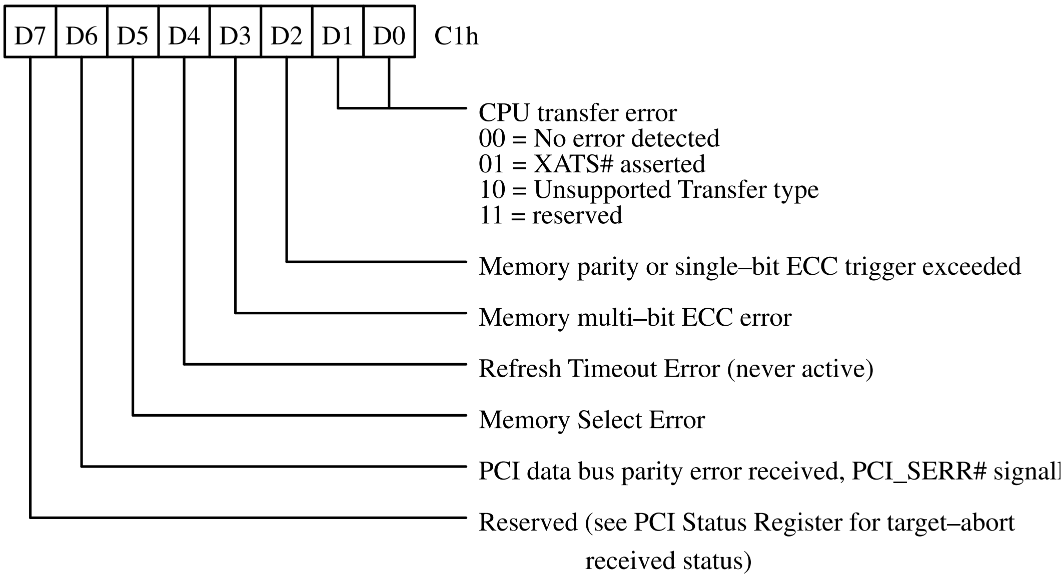
Bit 6 PCI data bus parity error received, PCI_SERR# signalled. This bit enables the 660 Bridge to drive PCI_SERR# in addition to PCI_PERR# when a PCI data bus parity error is detect when the PCI target (PCI write to system memory).
0 : Disabled
1 : Enabled

Bit 7 PCI target abort received. This bit enables detection of target aborts that occur while the 660 Bridge is the PCI master (CPU accesses to PCI).
0 : Disabled
1 : Enabled

4.3.40 Error Status 1

Index C1h	Read/Write	Reset to 00h

The error status 1 register is an 8-bit read/write BCR that contains status on error conditions that have been detected. Bits in this register can only be set to 1 as a result of a system error occurring. To reset any bit, the CPU can write a 1 to that bit location. Writing a 0 to a specific bit location leaves that bit unaltered. (Writing a 1 to bit 0 or bit 1 clears bit 0 and bit 1.) Bits cannot be reset if the error condition persists.



Bits 1:0 CPU transfer error:
00 : No error detected
01 : XATS# sampled asserted.
10 : Illegal transfer attribute encoding detected on the CPU bus.
11 : Reserved
Note: These bits can also be reset by means of the unsupported transfer type read and clear register.

Bit 2 If parity memory is implemented, this bit is set on a parity error. If ECC memory has been implemented, this bit is set in response to exceeding the Single–bit ECC error trigger threshold. This bit can also be reset by means of the memory parity error read and clear register (address 8000 0841h).

Bit 3 If ECC memory has been implemented, this bit is set in response to the detection of a multi–bit ECC error.

Bit 4 Refresh timeout errors never occur; therefore, hardcoded 0.

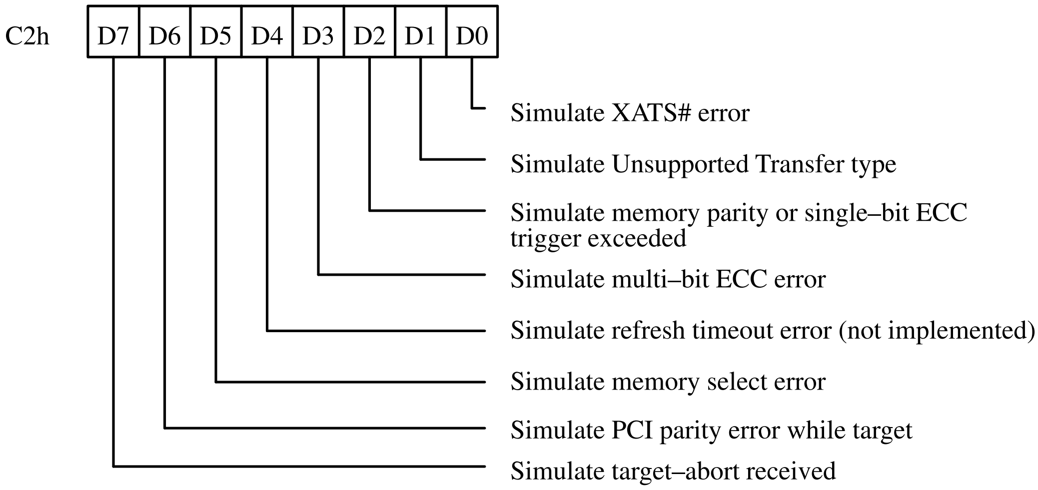
Bit 5 Memory select error: The CPU has run a memory cycle from 0 to 2G, but no memory is populated at that address.

Bit 6 PCI_SERR# has been asserted is response to the detection of a data bus parity error during a PCI master write to system memory.

4.3.41 Error Simulation 1

Index C2h	Read/Write	Reset to 00h

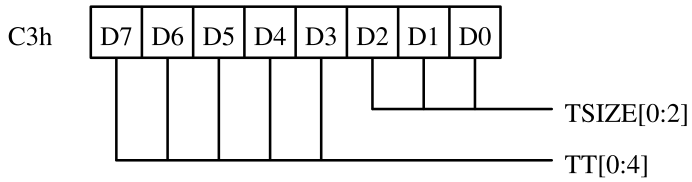
The error simulation 1 register is an 8-bit read/write BCR that causes the 660 Bridge to respond as though an actual system error has occurred. Setting a bit in this register sets the corresponding status bit if the error is enabled (mask is cleared). If an error simulation bit is set, the error condition persists until the error simulation bit is cleared. Note that since the error didn't really occur, the address and control information that is ordinarily saved when an error is detected is meaningless.



4.3.42 CPU Bus Error Status

Index C3h	Read Only	Reset: Undefined

The CPU bus error status register is an 8-bit, read-only BCR which contains information regarding the status of the CPU bus when an error is detected during a CPU cycle. This register is only valid if register C7 bit 4 is cleared.



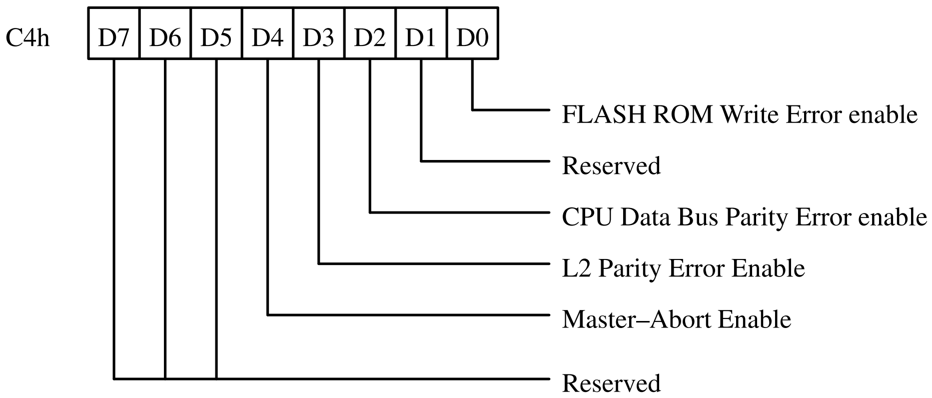
Bits 2:0 TSIZ[0:2]: These bits hold the TSIZ[0:2] encoding when the system error occurred.

Bits 7:3 TT[0:4]: These bits hold the TT[0:4] (transfer type) encoding on the 60X CPU bus when the system error occurred.

4.3.43 Error Enable 2

Index C4h	Read/Write	Reset to 00h

The error report enable 2 register is an 8-bit read/write BCR that selects which additional system error conditions get reported. A bit set in this register allows errors in the error status 2 register to generate TEA# or MCP#.

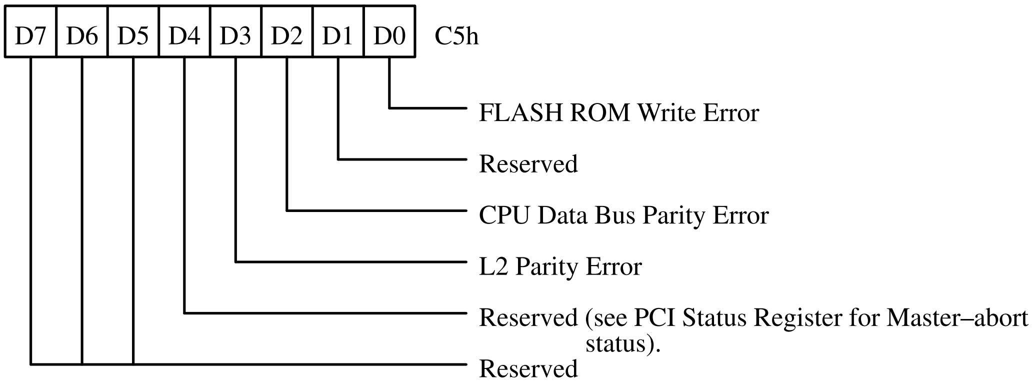


Bit 0	Flash/ROM write error enable: When this bit is set, attempts to write to the flash/ROM when it is locked result in an error.
Bit 2	CPU data bus parity error enable: When this bit is set, parity errors on the CPU data bus are detect.
Bit 3	L2 cache parity error enable: When this bit is set, parity errors during CPU reads from the L2 cache are detected. If the L2 is not present or does not implement parity, this bit should not be enabled.
Bit 4	Master Abort Enable: When this bit is set, PCI master aborts that occur while the 660 Bridge is the PCI master for CPU to PCI access are detected as errors. Note that this bit should not be enabled when running an operating system that does not consider master aborts to be errors.

4.3.44 Error Status 2

Index C5h	Read/Write	Reset to 00h

The error status 2 register is an 8-bit read/write BCR that contains additional error conditions status information. Bits in this register can only be set to 1 as a result of a system error occurring. A 1 written to a specific bit location resets the bit to 0. A 0 written to a specific bit location leaves that bit unaltered.

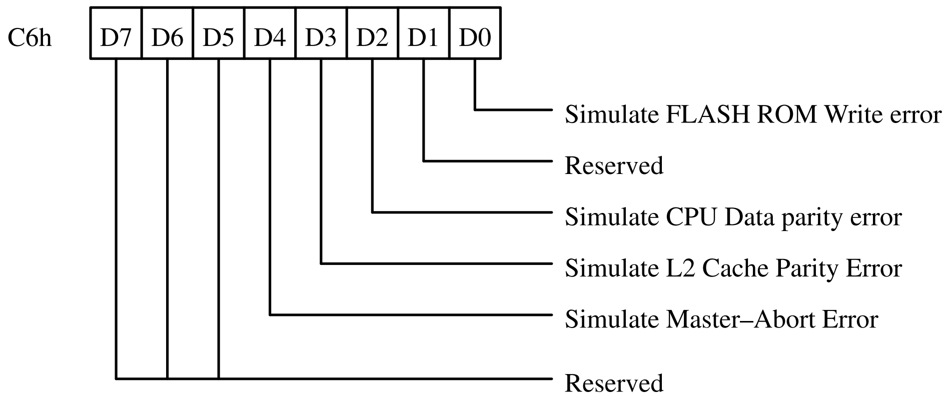


Bit 0	Flash ROM write error: When set, the 664 Controller has detected an attempt to write to the flash/ROM after the flash/ROM write lock–out bit has been set.
Bit 1	Reserved
Bit 2	CPU data bus parity error: When set, a parity error on the CPU data bus has been detected during a transfer between the CPU and the 660 Bridge.
Bit 3	L2 cache parity error: This bit is set if a parity error is detected on a CPU read from the L2 cache. Note that this bit can also be reset by means of the L2 cache parity error read and clear register.
Bit 7:4	Reserved

4.3.45 Error Simulation 2

Index C6h	Read/Write	Reset to 00h

The error simulation 2 register is an 8-bit read/write BCR that causes the 660 Bridge to respond as though an actual system had occurred. Setting a bit in this register sets the corresponding bit in the error status 2 register (index C5h). If a bit in this register is set and its corresponding enable in the error report enable register (C4h) is set, the 664 Controller responds by the assertion of either MCP# or TEA#. Note that since the error didn't really occur, the address and control information that is ordinarily saved when an error is detected is meaningless.



Bit 0 Simulate flash ROM write error when set.

Bit 2 Simulate CPU data bus parity error when set.

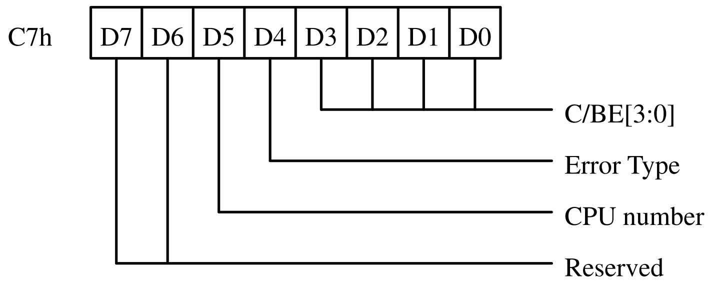
Bit 3 Simulate L2 cache parity error when set.

Bit 4 Simulate Master Abort Error

4.3.46 PCI Bus Error Status

Index C7h	Read Only	Reset: Undefined

The PCI bus error status register is an 8-bit read-only BCR that contains information regarding the status of the PCI bus and other status information when an error is detected.



Bits 3:0 PCI_C/BE#[3:0] : These bits hold the value of the PCI_C/BE#[3:0] bits during the address tenure of the cycle that caused a PCI bus error. Note that the PCI_C/BE# lines carry the bus command during the address tenure. These bits are only valid if an error occurred while the 660 Bridge was the PCI target (register C7 bit 4 = 1)

Bit 4 Error Type: This bit is set if an error occurred while the 660 Bridge is the PCI target. This bit is cleared if an error occurred during a CPU cycle.

Bit 5 CPU Number: This bit is 0 if CPU #1 (the parked CPU) is the CPU bus master when an error occurs. This bit is 1 if CPU #2 is the CPU bus master when an error occurs. This bit is only valid if the error occurred during a CPU cycle (register C7 bit 4 = 0).

4.3.47 CPU/PCI Error Address

Index C8 to CBh	Read/Write	Reset: Undefined (each BCR)

The CPU/PCI bus error address registers are four 8-bit read/write BCRs that contain the address of the CPU or PCI transaction that generated an error. Once an address has been latched, it remains in these registers until the corresponding error detect bit(s) in the error status 1 register (C1h) or the pci device status registers (06h, 07h) have been reset.

C8h CPU_ADDR[24:31] or PCI_AD[7:0]

C9h CPU_ADDR[16:23] or PCI_AD[15:8]

CAh CPU_ADDR[8:15] or PCI_AD[23:16]

CBh CPU_ADDR[0:7] or PCI_AD[31:24]

4.3.48 Single-Bit ECC Error Address

Index CC to CFh	Read/Write	Reset: Undefined (each BCR)

The single-bit ECC error address registers are four 8-bit read/write BCRs that contain the address of the memory transaction that generated a single-bit ECC error. This register is updated each time a single–bit ECC error is detected.

The value stored in this register is the offset into system memory. This means that if the error occurred during a CPU access, then the CPU address is saved. If the error occurred during a PCI access, then the PCI address minus 2G is saved.

CCh Most-significant byte of address

CDh Second-most-significant byte of address

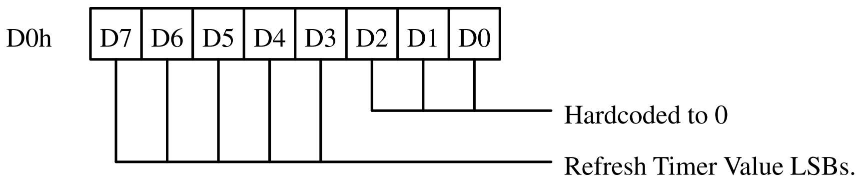
CEh Third-most-significant byte of address

CFh Least-significant byte of address

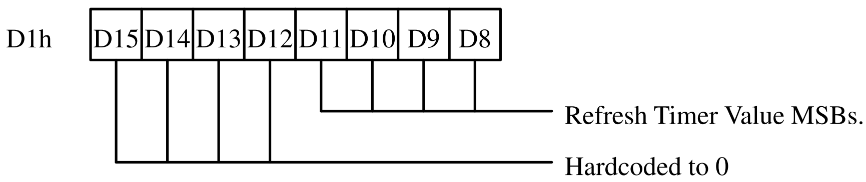
4.3.49 Refresh Timer Divisor Register

Index D0 to D1	Read/Write	Reset to F8 (D0) and 01 (D1)

The refresh timer register is a 16-bit BCR that determines the memory refresh rate. Typical refresh rates are 15.1 to 15.5 microseconds. If all DRAM in the system supports extended (slow) refresh, the refresh rate can be slower. The refresh timer is clocked by the PCI clock input to the 664 Controller. The reset value of 01F8h provides a refresh rate of 15.1 microseconds while the PCI clock is 33MHz. (01F8h equals 504 times 30ns equals 15.12us.) Bits 3–11 of the timer allow timer values from 8 to 4096.



Bits 7:3 Refresh timer (7:3) : These are the five least-significant bits of the refresh timer value.



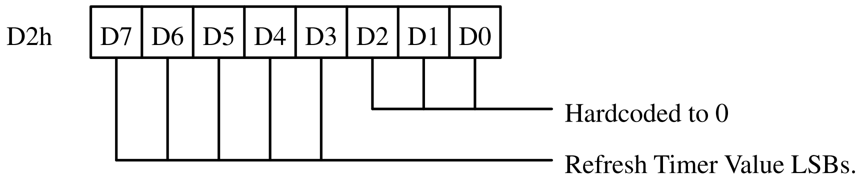
Bits 11:8 Refresh timer (11:8) : These are the four most-significant bits of the refresh timer value.

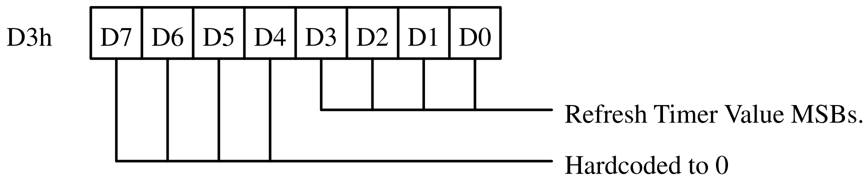
4.3.50 Suspend Refresh Timer Register

Index D2 to D3	Read/Write	Reset to F8 (D2) and 01 (D3)

The suspend refresh timer register is a 16-bit BCR that determines the memory refresh rate while the STOP_CLK_EN# pin to the 664 Controller is asserted. While in suspend, refresh is still generated from the PCI clock, but since this register determines the count, a different refresh rate or different PCI clock frequency can be used. See section 4.3.49 for programming details.

The suspend refresh timer register must be written by a half-word (16-bit) write. A read of the register is a don’t care for the number of bytes.





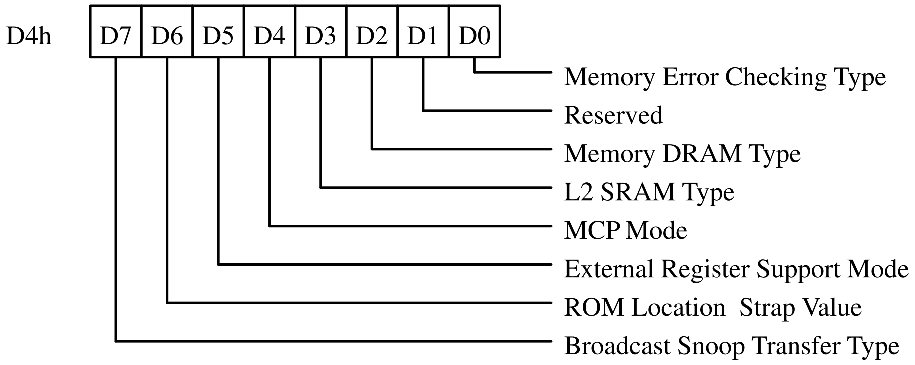
Bits 7:3 Suspend refresh timer (7:3) : These are the five least-significant bits of the refresh timer value.

Bits 3:0 Suspend refresh timer (11:8) : These are the four most-significant bits of the refresh timer value.

4.3.51 Bridge Chip Set Options 3 Register

Index D4h	Read/Write	Reset to 00h

This 8-bit read/write BCR controls various 660 Bridge functions.



Bit 0 Memory error checking type.
0 = parity
1 = ECC.

Bit 1 Reserved

Bit 2 DRAM memory type
0 = standard page mode DRAM
1 = hyper page mode (extended data out EDO) DRAM

Bit 3 Data SRAM type
0 = asynchronous
1 = synchronous

Bit 4 MCP# mode. This bit selects either the 604/603 MCP# error reporting mode or the 601 TEA# and interrupt error reporting mode. This is the mode select, not the output enable bit for the signal. See section 4.3.37, and section 9.2.
0 = MCP# mode selected. Use with 603/604.
1 = MCP# mode deselected. Errors are reported using the 601 protocol.

Bit 5 External register support mode.
1 = External register mode enabled. Reads to 8000 0814h, 8000 081Ch, and 8000 0850h are forwarded to the PCI bus and ignored by the internal BCRs. Writes are forwarded to the PCI bus and are also written to the internal BCRs.
0 = External register mode disabled. Accesses to these three registers go only to the internal BCRs, and are not forwarded to the PCI bus.

Bit 6 Value of the ROM location strapping pin (read–only)
0 = Direct (PCI based)
1 = Remote. ROM located on tertiary I/O bus.

Bit 7 Broadcast snoop transfer type. To maintain memory coherence, set this bit correctly before allowing PCI to memory accesses. See section 6.4.
0 = 601/604 transfer type encodings used for broadcasting PCI to memory accesses to the CPU bus.
1 = 603 transfer type encodings used.

Section 5
System Memory (DRAM)

The memory controller in the 664 Controller chip controls the system memory DRAM. The system memory can be accessed from both the 60X CPU bus and the PCI bus.

5.1 Features and Supported Devices

- Supports memory operations for the PowerPC Architecture™
- Data bus path 72 bits wide—64 data bits and eight bits of optional ECC or parity data
- Eight SIMM sockets supported, with empty SIMM sockets allowed at any position
- Eight RAS outputs, eight CAS outputs, and two write-enable outputs
- Supports industry-standard 8-byte (168-pin) SIMMs of 8M, 16M, 32M, 64M, and 128M that can be individually installed for a minimum of 8M and a maximum of 1G
- Supports industry-standard 4-byte (72-pin) SIMMs of 4M, 8M, 16M, 32M and 64M that must be installed in pairs for a minimum of 8M and a maximum of 512M
- Mixed use of different size SIMMs, including mixed 4-byte and 8-byte SIMMs
- Full refresh support, including refresh address counter and programmable DRAM refresh timer with low-power mode
- Burst-mode memory address generation logic
  - 32-byte CPU bursts to memory
  - Variable length PCI burst to memory
- Little-endian and big-endian addressing and byte swapping modes
- Provides row and column address multiplexing for DRAM SIMMs requiring the following addressing:

SIMM type	SIMM size	Addressing
72-pin	4 Meg	10 x 10
	8 Meg	10 x 10
	16 Meg	11 x 11
	32 Meg	11 x 11
	64 Meg	12 x 12
168-pin	8 Meg	10 x 10
	16 Meg	11 x 10
	32 Meg	12 x 10 or 11 x 11
	64 Meg	12 x 11
	128 Meg	12 x 12

5

5.1.1 SIMM Nomenclature

The term SIMM is used extensively to mean DRAM memory module, without implying the physical implementation of the module, which can be a SIMM, DIMM, or other package.

5.1.2 DRAM Timing

The memory controller timing parameters are programmable to allow optimization of timings based on speed of DRAM, clock frequency, and layout topology. Timing must be programmed based on the slowest DRAM installed.

- Support for fast page-mode DRAMs
- Support for extended-data-out (EDO) DRAM (hyper-page mode)
- If 70ns DRAM is used in a system with the 60X CPU bus at 66MHz, the minimum access times for initial (not pipelined) CPU to memory transfers with page mode and EDO DRAM are as follows:

Transfer	EDO DRAM	Page Mode DRAM	Note
Initial Read Burst	9–3–3–3	9–4–4–4	CPU clocks for 32 bytes
Initial Write Burst	7–3–3–3	7–4–4–4	CPU clocks for 32 bytes

- In the same system, the times for a pipelined burst following a read are as follows:

Transfer	EDO DRAM	Page Mode DRAM	Note
Page Hit Read	–3–3–3–3	–4–4–4–4	CPU clocks for 32 bytes
Page Hit Write	–4–3–3–3	–5–4–4–4	CPU clocks for 32 bytes

- In the same system, the times for a pipelined burst following a write are as follows:

Transfer	EDO DRAM	Page Mode DRAM	Note
Page Hit Read	–7–3–3–3	–7–4–4–4	CPU clocks for 32 bytes
Page Hit Write	–4–3–3–3	–5–4–4–4	CPU clocks for 32 bytes

- Other minimum memory timings are as follows:
  - L2 cache read at 66MHz with synchronous SRAM—2-1-1-1
  - L2 cache read at 66MHz with asynchronous SRAM—3-2-2-2
  - PCI to memory read at 66MHx CPU and 33MHz PCI
    5-1-1-1 -1-1-1-1 6-1-1-1 -1-1-1-1 6-1-1-1 -1-1-1-1 ... 6-1-1-1 -1-1-1-1
  - PCI to memory write at 66MHx CPU and 33MHz PCI
    5-1-1-1 -1-1-1-1 3-1-1-1 -1-1-1-1 3-1-1-1 -1-1-1-1 ... 3-1-1-1 -1-1-1-1

5.1.3 DRAM Error Checking

The 660 Bridge supports either no parity or one bit per byte parity DRAM SIMMs, in which one parity bit is associated and accessed with each byte. The bridge is BCR programmable to support either no parity, even parity, or ECC data error detection and correction. ECC is implemented using standard parity SIMMs. All installed SIMMs must support the selected error checking protocol.

Systems without error checking cost the least. Parity checking mode allows a standard level of error protection with no performance impact. ECC mode allows detection and correction of all single-bit errors and detection of all two-bit errors. ECC mode adds one CPU clock to the latency of CPU to memory reads, and does not effect the timing of writes

5.2 Memory Performance

Table 5-1 shows the cycle timings for read and write burst transfers from the CPU to system memory for both EDO and page mode DRAM. The first section of the table shows access times from the memory controller idle state, which it enters when it is not servicing a read or write request. The other two sections of the table show access times during back to back burst transfers. The middle section of the table shows access times for the second of any pair of back to back transfers where the first transfer is a read. The lower section of the table shows access times for the second of any pair of back to back transfers where the first transfer is a write.

Table 5-1. CPU To Memory Transfer Timing

Transfer	EDO DRAM	Page Mode DRAM	Note
Initial Read Burst	9–3–3–3	9–4–4–4	(3)
Initial Write Burst	7–3–3–3	7–4–4–4	
For a pipelined burst transfer immediately following a read:			
Page Hit Read	–3–3–3–3	–4–4–4–4	(3)
Page Hit Write	–4–3–3–3	–5–4–4–4	
Page Miss and Bank Miss Read	–7–3–3–3	–7–4–4–4	(1,3)
Page Miss and Bank Hit Read	–9–3–3–3	–9–4–4–4	(2,3)
Page Miss and Bank Miss Write	–7–3–3–3	–7–4–4–4	(1)
Page Miss and Bank Hit Write	–9–3–3–3	–10–4–4–4	(2)
For a pipelined burst transfer immediately following a write:			
Page Hit Read	–7–3–3–3	–7–4–4–4	(3)
Page Hit Write	–4–3–3–3	–5–4–4–4	
Page Miss and Bank Miss Read	–9–3–3–3	–9–4–4–4	(1,3)
Page Miss and Bank Hit Read	–12–3–3–3	–12–4–4–4	(2,3)
Page Miss and Bank Miss Write	–6–3–3–3	–7–4–4–4	(1)
Page Miss and Bank Hit Write	–8–3–3–3	–9–4–4–4	(2)

Notes for Table 5-1:

1) The RAS# of the new bank is high and has been high (precharging) for the minimum RAS# high time. The bridge places the address on the address lines and asserts RAS#.
2) The access is a page miss, but within the same bank, so the RAS# line must be sent high for at least the minimum RAS# high (precharge) time. The bridge also places the new address on the address lines and asserts RAS#.
3) If asynchronous SRAMs are used with the internal L2 controller, an additional clock cycle is added to the fourth beat of any CPU to memory read burst that causes a cache miss. For example, following a read—a pipelined page hit, cache miss, burst read with EDO DRAM, requires –3–3–3–3 CPU clocks when the L2 uses burst SRAMs and –3–3–3–4 CPU clocks when the L2 uses asynchronous SRAMs. This extra beat is caused by delaying the final TA# by one CPU_CLK to allow the asynchronous SRAM sufficient data hold time for the fourth beat.

5.2.1 Page Hit and Page Miss

PowerPC 60X CPU bus memory transfers have the following characteristic behavior. When a 60X CPU issues a memory access followed immediately by another memory access, the second access is typically from the same page of memory. On the other hand, if the second memory access does not immediately follow the first one (so that the CPU bus goes idle) then the second memory access is typically a page miss. Thus the majority of memory accesses following a bus idle condition are page misses. 660 Bridge memory performance is optimized by assuming that a CPU to memory transfer from bus idle will be a page miss.

When neither the CPU or the PCI is accessing memory, the memory controller goes to the idle state, and all RAS# lines are precharged (deasserted). Deasserting the RAS# lines at idle begins to satisfy the minimum RAS# precharge time requirement. Assuming that the first access out of idle will be a page miss, this technique allows the memory controller to reduce the time required for the initial beat of the burst DRAM read or write access by three CPU clocks. If the initial access is a page hit, this technique results in an increase in access time of two CPU clocks. A net gain is realized whenever the system is experiencing more page misses from bus idle than page hits from bus idle.

For the first beat of pipelined transactions, the memory controller checks the MA[11:0] memory address for a page hit. If the address is within the same 8K memory page as the previous memory access it is a page hit, the row address currently latched into the DRAM is considered valid, and the bridge accesses the DRAM using CAS# cycles. On page misses, the bridge latches the new row address into the DRAMs before it accesses the DRAM using CAS# cycles.

On CPU to memory bursts, only the address of the first beat of the burst is checked for page hits, because the following three beats are always within the same memory page.

On PCI to memory bursts, the address of each data phase of the burst is checked for page hits.

5.2.1.1 RAS# Precharge

This section describes the RAS# precharging technique in further detail, using, as an illustration, descriptions of how a memory read begins for an initial read entry from idle with the RAS# fully precharged (see Figure 5-1).

TBD

Figure 5-1. RAS# Precharge Sequence

5.2.1.2 Page and Bank Hits and Misses

This section describes how the memory controller handles page hits and misses. It also discusses bank hits and misses. Figure 5-2 through Figure 5-5 illustrate simplified timing diagrams showing the action of the relevent signals (following an initial read burst) for a pipelined:

read – page hit
write – page hit
read – page miss and bank miss
read – page miss and bank hit.

TBD

Figure 5-2. Read, Page Hit

TBD

Figure 5-3. Write, Page Hit

TBD

Figure 5-4. Read, Page Miss and Bank Miss

TBD

Figure 5-5. Read, Page Miss and Bank Hit

5.2.2 CPU to Memory Access Pipelining

CPU to memory accesses are pipelined, with the result that during a series of back-to-back CPU to memory accesses, all transfers following the initial transfer are faster. The information from the address tenure of the subsequent transfers is processed by the bridge while the data tenure of the preceding transfer is still active.

Table 5-1 shows the effects of this pipelining on memory performance. Considering a series of CPU to memory read transfers using EDO DRAM, the initial burst requires 9–3–3–3 CPU clocks. If this transfer is followed immediately (back-to-back) by another CPU to memory transfer, the required cycle times are shown in the middle section of the table. For example, a pipelined page hit read requires only –3–3–3–3 CPU clocks. As long as the transfers are back-to-back, they are pipelined, and can be retired at the pipelined rates.

Figure 5-6 shows the basic address pipelining sequence.

TBD

Figure 5-6. CPU to Memory Transfer Address Pipelining

5.2.3 Extended Data Out (EDO) DRAM

Description of EDO memory & operation
Description of how the memory controller uses EDO to improve performance.

5.2.4 Memory Timing Programming

Most memory controller timing parameters can be adjusted to maximize the performance of the system with the available resources. This adjustment is done by programming various memory controller BCRs. Figure 5-7 shows the various programmable memory timing variables. These variables control the number of CPU_CLKs between various events. The actual amount of time between the events shown will also be affected by various other factors such as clock to output delays. The CPU_CLK signal shown is not meant to be contiguous, as the number of clocks between various events is programmable.

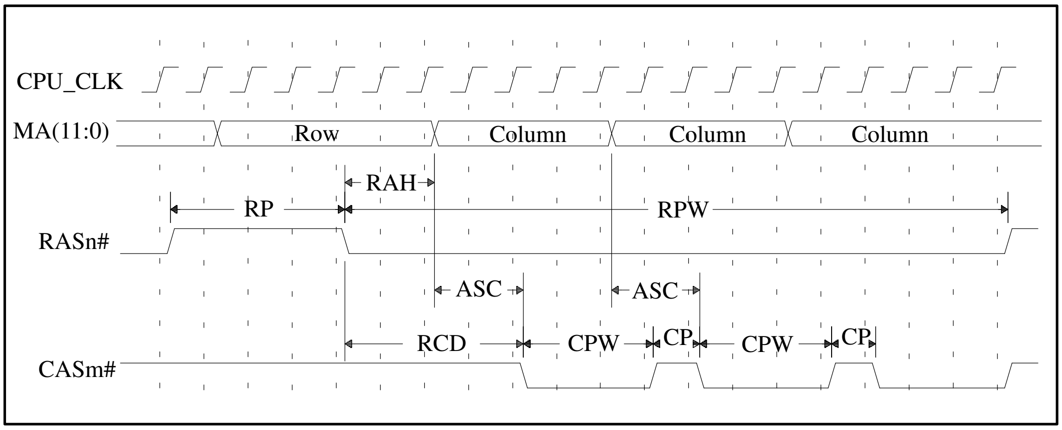


Figure 5-7. CPU to Memory Transfer Timing Programming

Table 5-2 shows the function, location, and section references for the variables shown in Figure 5-7.

Table 5-2. Supported SIMM Topologies

Variable	Function	BCR	Section
ASC	Column Address Setup (min)	Memory Timing Register 2	5.2.4.2
CP	CAS Precharge	Memory Timing Register 2	5.2.4.2
CPW	CAS Pulse Width (Read & Write)	Memory Timing Register 2	5.2.4.2
RAH	Row Address Hold (min)	Memory Timing Register 1	5.2.4.1
RCD	RAS to CAS Delay (min)	Memory Timing Register 2	5.2.4.2
RP	RAS Precharge	Memory Timing Register 1	5.2.4.1
RPW	RAS Pulse Width	Memory Timing Register 1	5.2.4.1

Note that ASC, RAH, and RCD are minimums. if RAH + ASC does not equal RCD, then the larger value will be used such that:

- If RCD < RAH + ASC, then the actual RCD will be stretched to equal RAH + ASC.
- If RCD > RAH + ASC, then the actual RAH will be stretched to equal RCD – ASC.

5.2.4.1 Memory Timing Register 1

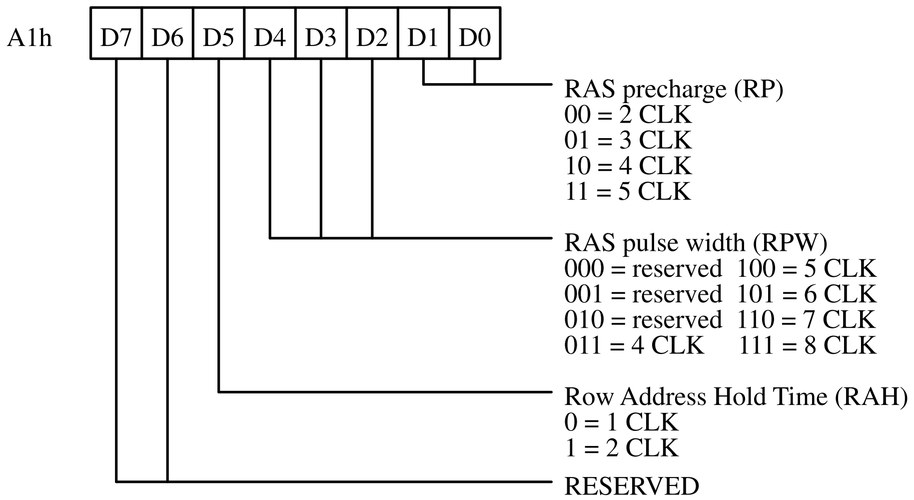
Index A1	Read/Write	Reset to 3Fh

This BCR determines the timing of RAS# signal assertion for memory cycles. RAS# timing must support the worst-case timing for the slowest DRAM installed in the system. See Section 5.2.4.

Bits 1:0 These bits control the number of CPU clocks for RAS# precharge.

Bits 4:2 These bits control RAS pulse width except on refresh. For refresh, the RAS# pulse width is hard-coded to three PCI clocks.

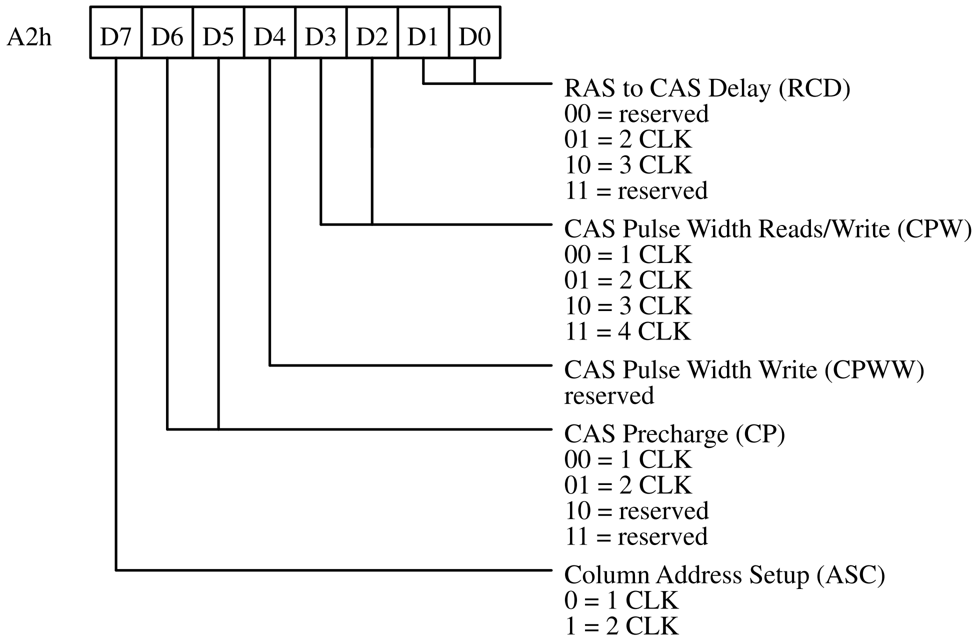
Bit 5 This bit controls the number of CPU clocks that the row address is held following the assertion of RAS#.



5.2.4.2 Memory Timing Register 2

Index A2	Read/Write	Reset to AEh

This BCR determines the timing of CAS# signal assertion for memory cycles. CAS# timing must support the worst-case timing for the slowest DRAM installed in the system. See Section 5.2.4.



5.2.5 RAS Watchdog Timer Register

Index B6	Read/Write	Reset to 53h

This BCR limits the maximum RAS# active pulse width. The value of this BCR represents the maximum amount of time that any RAS# can remain active in units of eight CPU bus clocks. The timer (down-counter) associated with this BCR is reloaded on the assertion of any RAS# line. On expiration of the timer, the 660 Bridge drops out of page mode to deassert the RAS# lines.

In response to the RESET# signal, this register is reset to 53h. This value results in a maximum RAS# active time of just under 10us at 66MHz. This is the value required by most 4-byte and 8-byte SIMMs. The value of the BCR must be reprogrammed if the CPU bus frequency is not 66MHz or a different RAS# pulse width is required.

5.3 System Memory Addressing

5.3.1 SIMM Topologies

Table 5-3 shows the various memory module (SIMM) topologies that the 660 Bridge supports. Each memory bank can be populated with any supported SIMM.

Table 5-3. Supported SIMM Topologies

SIMM type	Size	Depth	Width	Banks	Addressing	Addressing Mode (1)
4-Byte Wide (72-pin)	4 Meg	1M	4 bytes	1 + 1 empty	10 x 10	2
	8 Meg	1M	4 bytes	2	10 x 10	2
	16 Meg	4M	4 bytes	1 + 1 empty	11 x 11	2
	32 Meg	4M	4 bytes	2	11 x 11	2
	64 Meg	16M	4 bytes	1 + 1 empty	12 x 12	3
8-Byte Wide (168-pin)	8 Meg	1M	8 bytes	1	10 x 10	2
	16 Meg	2M	8 bytes	1	11 x 10	2
	32 Meg	4M	8 bytes	1	11 x 11 or 12 x 10	2
	64 Meg	8M	8 bytes	1	12 x 11	3
	128 Meg	16M	8 bytes	1	12 x 12	3

Note for Table 5-3:
(1) The memory bank addressing mode BCRs (see Section 5.3.6).

The 660 Bridge supports the 168-pin 8-byte SIMMs shown in Table 5-3, which are each arranged as a single bank of 8-byte-wide DRAM. Each SIMM requires a single RAS# line. These SIMMs do not have to be installed in pairs.

The bridge also supports the 72-pin 4-byte SIMMs shown in Table 5-3, which are each arranged as two banks of 4-byte-wide DRAM, only one bank of which may be accessed at a given time. Each bank requires a RAS# line, and each bank is addressed by the same address lines. These SIMMs must be installed in pairs (of identical devices), since it is necessary to use two (72-pin) 4-byte SIMMs to construct an 8-byte-wide memory array. Since each 72-pin 4-byte SIMM consists of two banks, this pair of SIMMs also requires two RAS# lines. The 660 Bridge addresses a given SIMM based on the value of the associated memory bank addressing mode BCR.

5.3.2 Row and Column Address Generation

The 660 Bridge formats the row and column addresses presented to the DRAM based on the organization of the DRAM. In memory bank addressing mode 2, the bridge is configured to address devices that require 12x10, 11x11, 11x10, or 10x10 bit (row x column) addressing. In memory bank addressing mode 3, the bridge is configured to address devices that require 12x12 or 12x11 bit (row x column) addressing (no other addressing modes are currently available).

Table 5-4 and Table 5-5 show which CPU address bits are driven onto the memory address bus during CPU to memory transfers. Table 5-6 and Table 5-7 show which PCI_AD address bits are driven onto the memory address bus during PCI to memory transfers. These address line assignments are not affected by the endian mode of the system. The addressing mode is selected using the memory bank addressing mode BCRs (see Section 5.3.6). The addressing mode of each bank of memory is individually configurable.

5

Table 5-4. Row Addressing (CPU Addressing)

Memory Bank Addressing Mode	Addressing Mode BCR	MA 11	MA 10	MA 9	MA 8	MA 7	MA 6	MA 5	MA 4	MA 3	MA 2	MA 1	MA 0
12x10, 11x10, 10x10, 11x11	010 (Mode 2)	A 7	A 8	A 9	A 10	A 11	A 12	A 13	A 14	A 15	A 16	A 17	A 18
12x12, 12x11	011 (Mode 3)	A 7	A 8	A 9	A 10	A 11	A 12	A 13	A 14	A 15	A 16	A 17	A 18

Table 5-5. Column Addressing (CPU Addressing)

Memory Bank Addressing Mode	Addressing Mode BCR	MA 11	MA 10	MA 9	MA 8	MA 7	MA 6	MA 5	MA 4	MA 3	MA 2	MA 1	MA 0
12x10, 11x10, 10x10, 11x11	010	A 5	A 7	A 19	A 20	A 21	A 22	A 23	A 24	A 25	A 26	A 27	A 28
12x12, 12x11	011	A 5	A 6	A 19	A 20	A 21	A 22	A 23	A 24	A 25	A 26	A 27	A 28

Table 5-6. Row Addressing (PCI Addressing)

Memory Bank Addressing Mode	Addressing Mode BCR	MA 11	MA 10	MA 9	MA 8	MA 7	MA 6	MA 5	MA 4	MA 3	MA 2	MA 1	MA 0
12x10, 11x10, 10x10, 11x11	010	AD 24	AD 23	AD 22	AD 21	AD 20	AD 19	AD 18	AD 17	AD 16	AD 15	AD 14	AD 13
12x12, 12x11	011	AD 24	AD 23	AD 22	AD 21	AD 20	AD 19	AD 18	AD 17	AD 16	AD 15	AD 14	AD 13

Table 5-7. Column Addressing (PCI Addressing)

Memory Bank Addressing Mode	Addressing Mode BCR	MA 11	MA 10	MA 9	MA 8	MA 7	MA 6	MA 5	MA 4	MA 3	MA 2	MA 1	MA 0
12x10, 11x10, 10x10, 11x11	010	AD 26	AD 24	AD 12	AD 11	AD 10	AD 9	AD 8	AD 7	AD 6	AD 5	AD 4	AD 3
12x12, 12x11	011	AD 26	AD 25	AD 12	AD 11	AD 10	AD 9	AD 8	AD 7	AD 6	AD 5	AD 4	AD 3

5.3.3 DRAM Pages

The 660 Bridge uses an 8K page size for DRAM page-mode determination.

5.3.4 Supported Transfer Sizes and Alignments

The 660 Bridge supports all CPU to memory transfer sizes and alignments that do not cross an 8-byte boundary.

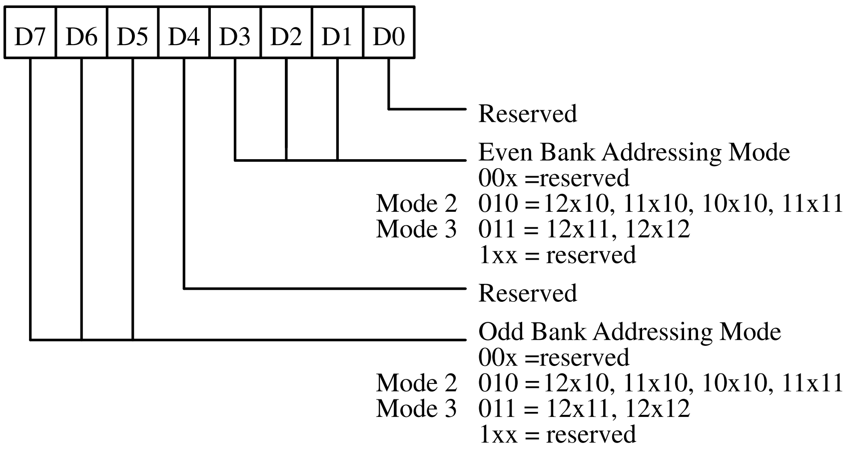
5.3.5 Unpopulated Memory Locations

Physical memory does not occupy the entire address space assigned to system memory in the memory map. When the CPU reads an unpopulated location, the bridge returns all-ones, and completes the transfer normally. When the CPU writes to an unpopulated location, the bridge signals normal transfer completion to the CPU, but does not write the data to memory. The memory select error bit in the error status 1 register (index C1h) is set in both cases.

5.3.6 Memory Bank Addressing Mode BCRs

Index A4 to A7	Read/Write	Reset to 44h (each BCR)

This array of four 8-bit, read/write BCRs defines the format of the row and column addressing of each DRAM memory bank.



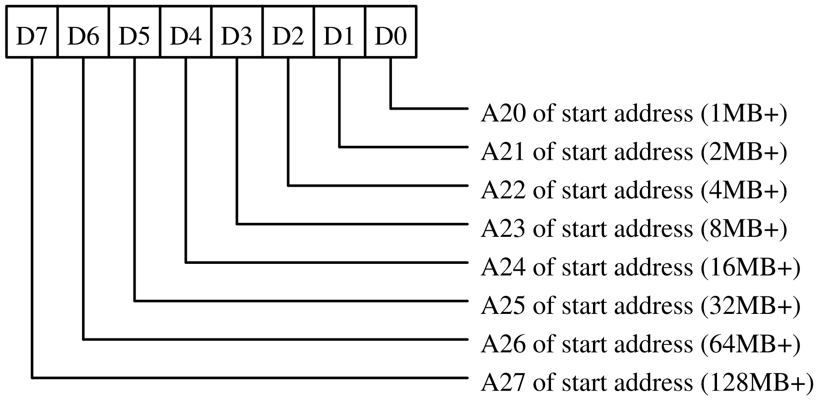
Register	Bits	Memory Bank	Bits	Memory Bank
A4h	3:0	0	7:4	1
A5h	3:0	2	7:4	3
A6h	3:0	4	7:4	5
A7h	3:0	6	7:4	7

5.3.7 Memory Bank Starting Address BCRs

Index 80 to 87h	Read/Write	Reset to 00h (each BCR)

This array of eight BCRs (along with the eight extended starting address registers) contains the starting address for each memory bank. Each pair of registers maps to the corresponding RAS# decode. For example, RAS[4]# corresponds to the BCRs at index 84h and 8Ch. The eight least-significant bits of the bank starting address are contained in the starting address register, and the most-significant bits come from the corresponding extended starting address register. The starting address of the bank is entered with the least significant 20 bits truncated. These BCRs must be programmed in conjunction with the ending address and extended ending address registers.

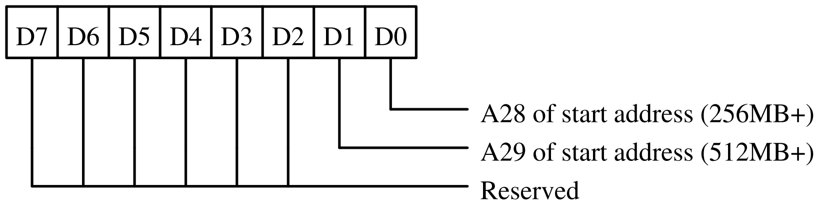
Program the banks in ascending order, such that (for n = 0 to 6) the starting address of bank n+1 is higher than the starting address of bank n. Each bank must be located in the 0 to 1G address range. See section 5.3.12.



5.3.8 Memory Bank Extended Starting Address BCRs

Index 88 to 8F	Read/Write	Reset to 00h (each BCR)

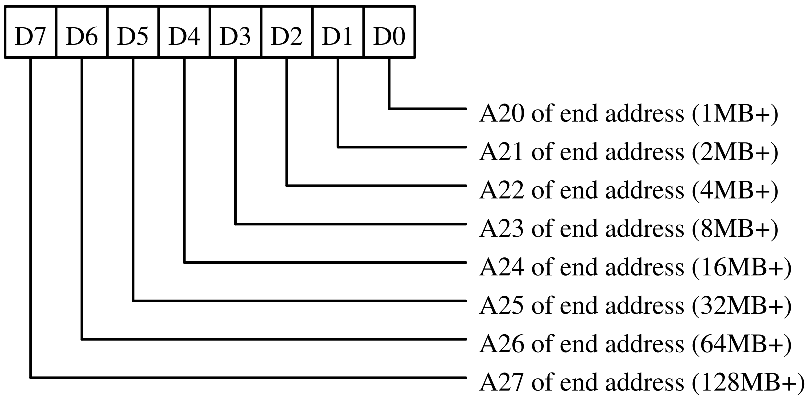
This array of eight BCRs (along with the eight starting address registers) contains the starting address for each memory bank. These BCRs contain the most-significant address bits of the starting address of the corresponding bank.



5.3.9 Memory Bank Ending Address BCRs

Index 90 to 97h	Read/Write	Reset to 00h (each BCR)

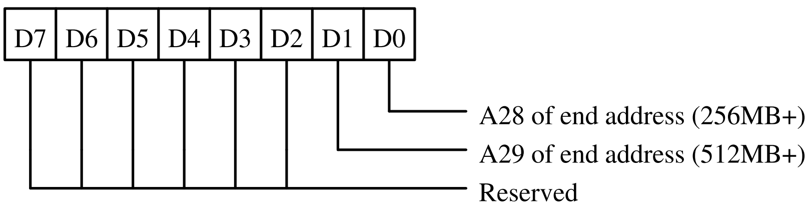
This array of eight BCRs (along with the eight extended starting address registers) contains the ending address for each memory bank. Each pair of registers maps to the corresponding RAS# decode. For example, RAS[4]# corresponds to the BCRs at index 94h and 9Ch. The eight least-significant bits of the bank ending address are contained in the ending address register, and the most-significant bits come from the corresponding extended ending address register. The ending address of the bank is entered as the address of the next highest memory location minus 1, with the least significant 20 bits truncated. Each bank must be located in the 0 to 1G address range. These BCRs must be programmed in conjunction with the ending address and extended ending address registers. See section 5.3.12.



5.3.10 Memory Bank Extended Ending Address BCR

Index 98 to 9F	Read/Write	Reset to 00 (each BCR)

This array of eight 8-bit, read/write registers (along with the eight ending address registers) contains the ending address for each memory bank. These BCRs contain the most-significant address bits of the ending address of its bank.

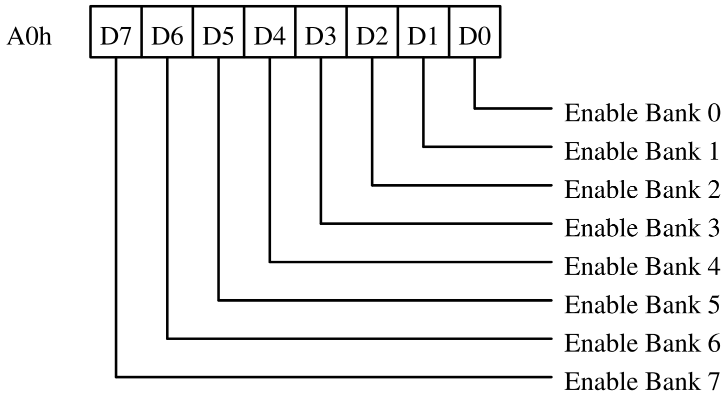


5.3.11 Memory Bank Enable BCR

Index A0	Read/Write	Reset to 00h

This BCR contains a control enable for each bank of memory. Each bank of memory must be enabled for proper refreshing. For each bit, a 0 disables that bank of memory and a 1 enables it.

This register must be programmed in conjunction with the starting address and ending address registers. If a bank is disabled by this register, the corresponding starting and ending address register entries become don't cares.



5

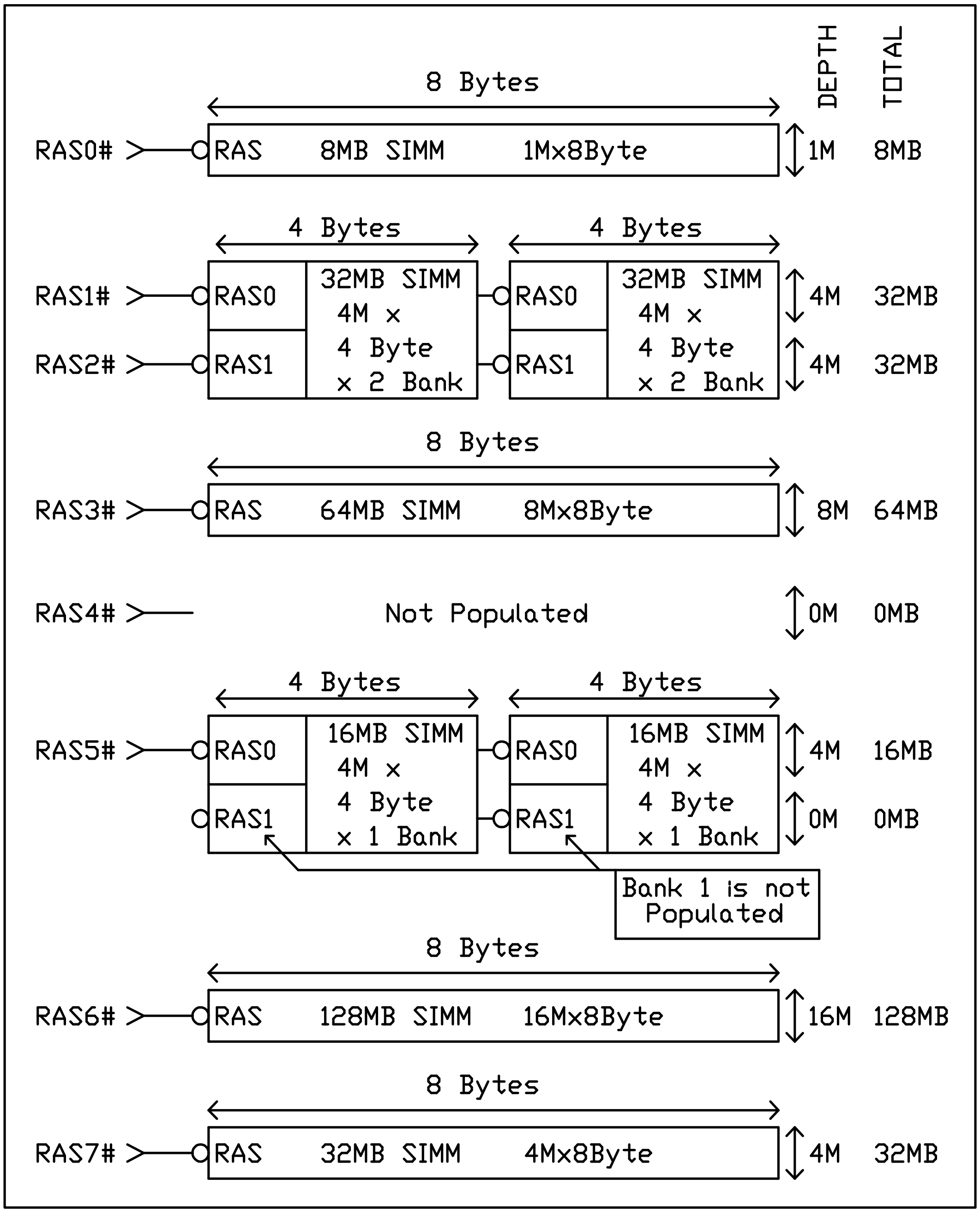


Figure 5-8. Example Memory Bank Configuration

5.3.12 Memory Bank Configuration Example

In the example memory bank configuration shown in Figure 5-8, the eight memory banks are populated by different size and organization devices. For convenience, this example shows the bridge configured to address each memory bank in order with no gaps in the populated address range but this is not required. Any bank can be placed in any (1MB aligned) non-populated address range from 0 to 1G.

5.3.12.1 Memory Bank Enable BCR

Program indexed BCR A0h (memory bank enable) to EFh to enable all banks except # 4.

5.3.12.2 Memory Bank Addressing Mode

As shown in Table 5-8, memory bank 0 (connected to RAS0#) contains an 8-byte x 1M SIMM (8M) with one bank (one RAS# line). It is addressed with 10 row and 10 column bits. Program bits 3:1 of indexed BCR A4h with 010b (mode 2).

5

Table 5-8. Example Memory Bank Addressing Mode Configuration

Bank	SIMM Type	SIMM Depth	SIMMs Per Bank	SIMM Bank Topology	SIMM Size	Row x Col	BCR ()	Bits ()	Mode
0	8-Byte	1M	1	8B x 1M x 1 bank	8M	10 x 10	A4	3:1	2
1	4-Byte	4M	2	4B x 4M x 2 bank	32M	11 x 11	A4	7:5	2
2	4-Byte	4M	2	4B x 4M x 2 bank	32M	11 x 11	A5	3:1	2
3	8-Byte	8M	1	8B x 8M x 1 bank	64M	12 x 11	A5	7:5	3
4	none	—	—	—	0M	—	A6	3:1	—
5	4-Byte	4M	2	4B x 4M x 2 bank	32M	11 x 11	A6	7:5	2
6	8-Byte	16M	1	8B x 16M x 1 bank	128M	12 x 12	A7	3:1	3
7	8-Byte	4M	1	8B x 4M x 1 bank	32M	11 x 11 12 x 10	A7	7:5	2

5.3.12.3 Starting and Ending Addresses

As shown in Table 5-9, program indexed BCR 88h with 00h to configure address bits 27:20 of the bank 0 starting address. Program indexed BCR 80h with 00h to configure address bits 29:28. Also program indexed BCR 98h with 00h to configure address bits 27:20 of the bank 0 ending address, and program indexed BCR 90h with 07h to configure address bits 29:28.

The next two physical SIMM units are 4-byte x 4M x 2 bank (16M x 2 bank = 32M) SIMMs, used side by side to achieve 8-byte width. They form banks 1 and 2, each of which is 8-byte x 4M (32M). Note that bank 7 is also 32M, populated by an 8-byte x 8M (32M) SIMM. Using the same configuration, banks 1 and 2 could also be implemented using two 8-byte x 4M (32M) SIMMs. Bank 3 is configured to the same size as bank 1 plus bank 2 and is implemented using a single 8-byte x 8M (64M) SIMM.

Bank 4 is not populated, so set memory bank enable BCR bit D4 to 0. The data in the starting, extended starting, ending, and extended ending address BCRs for bank 4 is ignored.

For proper operation, program the bridge to execute memory accesses at the speed of the slowest device. If ECC or parity is selected, all devices must support the capability. And if the bridge is programmed to utilize hyper-page mode, all devices must support extended-data out transfers.

Table 5-9. Example Memory Bank Starting and Ending Address Configuration

	SIMM	Starting	Extended		Base		Ending	Extended		Base	
Bank	Size	Address	BCR	Data	BCR	Data	Address	BCR	Data	BCR	Data
0	8M	0000 0000	88	00	80	00	007F FFFF	98	00	90	07
1	32M	0080 0000	89	00	81	08	027F FFFF	99	00	91	27
2	32M	0280 0000	8A	00	82	28	047F FFFF	9A	00	92	47
3	64M	0480 0000	8B	00	83	48	087F FFFF	9B	00	93	87
4	0M	Don't Care	8C	xx	84	xx	Don't Care	9C	xx	94	xx
5	32M	0880 0000	8D	00	85	88	0A7F FFFF	9D	00	95	A7
6	128M	0A80 0000	8E	00	86	A8	127F FFFF	9E	01	96	27
7	32M	1280 0000	8F	01	87	28	147F FFFF	9F	01	97	47

5.4 Error Checking and Correction

The 660 Bridge provides three levels of memory error checking—no checking, parity checking, and error checking and correction (ECC). If no memory checking is enabled, the system can be configured to use lower-cost, non-parity DRAM.

While the system is configured for parity checking, the 660 Bridge performs as follows:

- Uses odd parity checking
- Detects all single bit errors
- Allows full-speed memory accesses

While the system is configured for ECC, the 660 Bridge performs as follows:

- Uses an H–matrix and syndrome ECC protocol
- Uses the same memory devices and connectivity as for parity checking
- Detects and corrects all single-bit errors
- Detects all two-bit errors
- The first beat of CPU to memory reads require one additional CPU clock cycle
- CPU to memory burst writes and 8-byte single-beat writes are full-speed
- CPU to memory single-beat writes of less than eight bytes are implemented as read-modify-write (RMW) cycles
- PCI to memory reads are full-speed
- PCI to memory writes are full-speed while data can be gathered into 8-byte groups before being written to memory. Single beat or ungatherable writes require a read-modify-write cycle

5.4.1 Memory Parity

While the 660 Bridge memory controller is configured for memory parity checking, the bridge implements an odd parity generation and checking protocol, generating parity on memory writes and checking parity on memory reads. One parity bit is associated with each data byte and is accessed with it. When a parity error is detected during CPU to memory reads, the error is reported by means of TEA# or MCP#. When a parity error is detected during PCI to memory reads, the error is reported by means of PCI_SERR#.

5.4.2 ECC Overview

While ECC is enabled, the 660 Bridge uses the ECC logic to detect and correct errors in the transmission and storage of system memory data. The bridge implements the ECC protocol using the same connectivity and memory devices as parity checking.

While neither parity or ECC is enabled, the bridge executes memory writes of less than eight bytes in response to bus master requests to write less than eight bytes of data. The bridge always (whether parity, ECC or neither is enabled) reads data from memory in 8-byte groups, even if the bus master is requesting less than eight bytes.

While parity is enabled, the bridge also executes memory writes of less than eight bytes in response to bus master requests to write less than eight bytes of data, since writing a byte to memory also updates the associated parity bit. During memory writes, the bridge generates one parity bit for each byte of data and stores it with that byte of data. This parity bit is a function only of the data byte with which it is associated. During memory reads, the integrity of the data is parity checked by comparing each data byte with its associated parity bit.

However, when ECC is enabled the bridge reads from and writes to system memory only in 9-byte groups (as a 72-bit entity), even though the bus master may be executing a less-than-8-byte read or write. There is one byte of ECC check byte information for eight bytes of data. During memory writes, the eight check bits are generated as a function of the 64 data bits as a whole, and the check bits are stored with the data bits as a 72-bit entity. During memory reads, all 72 bits are read, and the eight check bits are compared with the 64 data bits as a whole to determine the integrity of the entire 72-bit entity.

In ECC mode, single-bit errors are corrected. When a multi-bit error is detected during CPU to memory reads, the error is reported by means of TEA# or MCP#. When a multi-bit parity error is detected during PCI to memory reads, the error is reported by means of PCI_SERR#. Note that the DRTRY# function of the 60X CPU is not used, even when the 660 Bridge is in ECC mode (the Bridge will not assert DRTRY#).

5.4.3 ECC Data Flows

While ECC is enabled, the 660 Bridge always reads eight data bytes and one check byte from memory during CPU and PCI reads. During bus master 8-byte writes to memory, the bridge writes eight data bytes and one check byte to memory. When the bus master writes less than eight data bytes to memory, it is possible for each check bit to change due to a write to any one of the eight data bytes. The Bridge then executes a read-modify-write (RMW) cycle—reading all eight data bytes from memory, modifying the appropriate bytes with the new data, recalculating all of the check bits, and writing the new data and check bits to memory.

5.4.3.1 Memory Reads

Figure 5-9 shows a simplified data flow in a 660 Bridge system during CPU to memory read transfers. Figure 5-10 shows the simplified data flow during PCI to memory reads. The data and check bits flow from system memory into the bridge where the checking logic combines the data with the check bits to generate the syndrome. If there is a single-bit error in the data, the correction logic corrects the data and supplies it to the requesting agent. If there is a multiple-bit error in the data, the bridge signals an error to the requesting agent.

Note that the structure of the bridge as shown in Figure 5-9 through Figure 5-13 is considerably simplified and does not show the posted write buffers and other internal details.

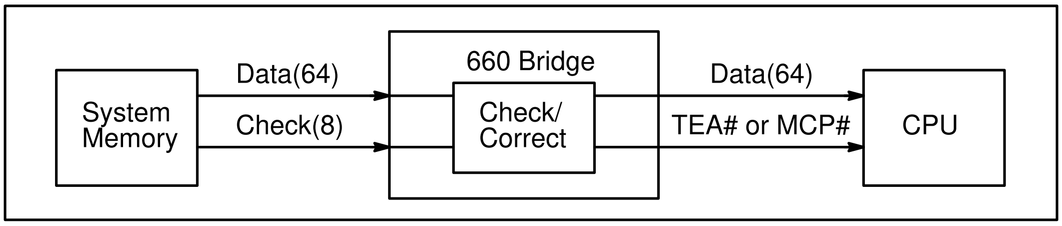


Figure 5-9. CPU Read Data Flow

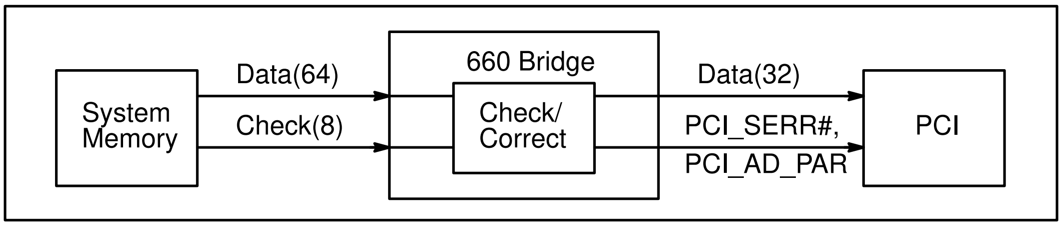


Figure 5-10. PCI Read Data Flow

5.4.3.2 Eight-Byte Writes

Figure 5-11 shows the simplified data flow in a 660 Bridge system during 8-byte CPU to memory writes. The data flows from the CPU into the bridge and out onto the memory data bus. The bridge generates the check bits based on the eight bytes of data, and stores them in memory with the data.

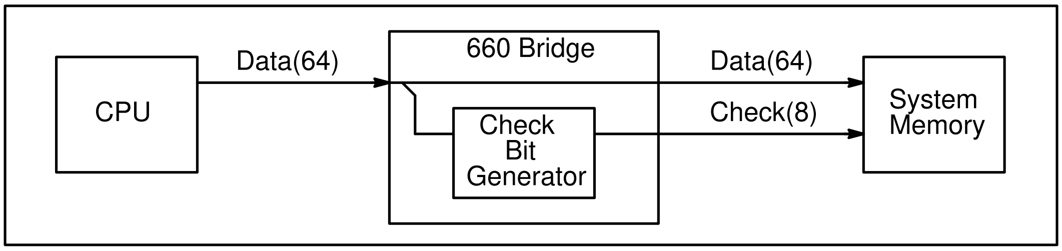


Figure 5-11. CPU 8-Byte Write Data Flow

Figure 5-12 shows the simplified data flow in a 660 Bridge based system during gathered PCI to memory 8-byte writes. During the first of the two gathered data phases (A), the data flows from the PCI bus into a 4-byte hold latch in the bridge. On the next data phase (B), the next 4–bytes of PCI data flows into the bridge, where it is combined with the data from the previous data phase. The 8-byte data then flows onto the memory data bus. The bridge generates the check bits based on the 8-byte data, and stores them in memory with the data.

Note that if either or both of the two gathered PCI data phases is a write of less than 4 bytes, the two data phases will still be gathered, and then the bridge will execute a RMW cycle, filling in the unwritten bytes (in the group of 8 bytes) with data from those locations in memory. The same write scenario while ECC is disabled does not cause a RMW cycle; the bridge merely writes only the indicated bytes, leaving the write enables of the unaccessed bytes deasserted.

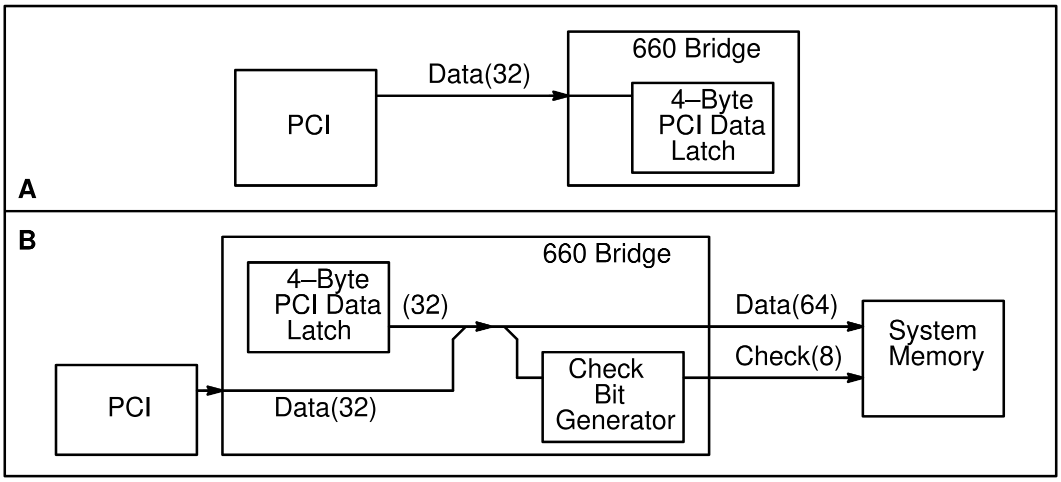


Figure 5-12. PCI 8-Byte Write Data Flow

5.4.3.3 Less-Than Eight-Byte Writes

Figure 5-13 shows the simplified data flow during a CPU or PCI bus master to memory write of less than eight bytes, during which the bridge executes a RMW cycle. In Figure 5-13(A), the bridge reads in the data and check bits from the addressed memory locations and places this data (corrected as necessary) in a register. In Figure 5-13(B), this data is modified by the bridge, which replaces the appropriate memory data bytes with write data from the bus master. The bridge then recomputes all of the check bits and writes the new data and check bits to memory.

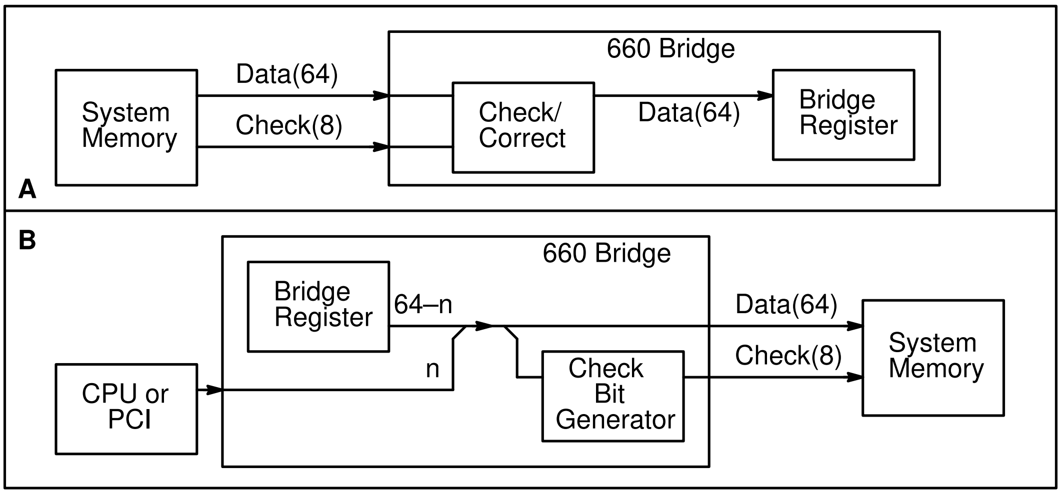


Figure 5-13. PCI or CPU Read-Modify-Write Data Flow

5.4.4 Memory Performance In ECC Mode

Enabling ECC mode on the 660 Bridge affects memory performance in various ways, depending on the transaction type. The effect is the same for both EDO and page mode DRAMs.

5.4.4.1 CPU to Memory Read in ECC Mode

ECC mode adds one CPU_CLK to single–beat CPU to memory read transfers and to the first beat of CPU to memory burst read transfers. The other beats of the burst are unaffected. During the extra CPU_CLK, the 663 Buffer holds the data while checking (and if necessary, correcting) it.

The 660 Bridge does not use the DRTRY# function, even while in ECC mode. In no–DRTRY# mode, during memory reads the 604 uses the data bus data internally as soon as it samples TA# active. The 660 Bridge supports this mode by presenting correct(ed) data to the 60X before driving TA# valid. The Bridge does not speculatively present data to the 60X (using TA#) and then assert DRTRY# if there is a data error.

By allowing the 604 to run in no–DRTRY# node, the 660 Bridge enables the 604 to use data from the L2 cache at the full speed of the L2 cache, without requiring the 604 to insert an additional (internal to the 604) one CPU clock delay on reads.

In DRTRY# mode, during memory reads the 604 holds the data internally for one CPU clock after sampling TA# active. This delay is inserted by the 604 to allow DRTRY# to be sampled before the data is used. Thus during CPU to memory reads in ECC mode while a 604 is in DRTRY# mode, the 660 Bridge inserts a one CPU clock delay to check/correct the data, and then the 604 adds a one CPU clock delay to check for DRTRY#. Thus two CPU clocks are added to single–beat CPU to memory read transfers and to the first beat of CPU to memory burst read transfers.

5.4.4.2 CPU to Memory Write in ECC Mode

ECC mode adds no additional clock cycles to 8-byte CPU to memory write transfers. Note that all 60X bursts are composed of 8-byte beats. CPU to memory writes of less than eight bytes are handled as RMW cycles, which usually require four additional CPU clocks as compared to 8-byte writes.

5.4.4.3 PCI to Memory Read in ECC Mode

ECC mode adds no additional clock cycles to PCI to memory read transactions.

5.4.4.4 PCI to Memory Write in ECC Mode

ECC mode has a complex effect on PCI to memory writes. During PCI to memory writes, the bridge attempts to gather adjacent 4-byte data phases into 8-byte memory writes. In response to conditions during a given data phase, the bridge either gathers, writes 8 bytes, or read-modify-writes, as shown in Table 5-10. Gather and 8-byte write operations incur no performance penalties, but RMW cycles add from two to four (usually three) PCI clock cycles to the transaction time. The consequences of ECC mode delays on PCI to memory write bursts are minor and are best understood from a few examples. The following examples assume page hits and no snoop hits.

Table 5-10. Bridge Response to Various PCI Write Data Phases

Bridge Operation	Conditions	Description of Operation
Gather	This data phase is not the last data phase, and This is a 4-byte transfer (BE[3:0]#=0000), and This data phase is to the lower 4-byte word.	The bridge latches the four bytes from this data phase into the low four bytes of a hold register.
Eight-Byte Write	The previous data phase caused a gather, and This is a 4-byte transfer, and This data phase is to the upper 4-byte word.	The bridge combines the (high) four bytes from this data phase with the (low) four bytes from the previous data phase, and writes all eight bytes to memory.
Read-Modify-Write	This is a single phase transaction, or This is the first data phase and is to the upper 4-byte word, or This is the last data phase and is to the lower 4-byte word, or This is a less-than-4-byte transfer.	The bridge Reads eight bytes from memory, modifies the data by replacing the appropriate four bytes with the data from this data phase, and then writes all eight bytes to memory.

5

Best case (see Table 5-11) is a burst starting on a low word (memory address is 0 mod 8), composed of an even number of data phases, in which all data phases transfer four bytes of data. Notice that the first data phase is gathered and the second data phase causes an 8-byte memory write. The following pairs of data beats follow the same pattern. No page misses, snoop hits, or data beats of less than 4 bytes are encountered. This case adds no PCI clock cycles to the best-case transaction time.

Table 5-11. Bridge Response to Best Case PCI Write Burst

Data Phase	Word	Bridge Operation	Special Conditions	Performance Impact
n (last)	High	8-byte Write	None	None
n –1	Low	Gather	None	None
...	...	...	...	...
4	High	8-byte Write	None	None
3	Low	Gather	None	None
2	High	8-byte Write	None	None
1 (first)	Low	Setup + Gather	None	None

Table 5-12 shows a case where the first data phase is to a high word (memory address is 4 mod 8), and is composed of an odd number of data phases in which all data phases transfer four bytes of data. Here the total effect is to add three PCI clocks to the transaction time, which is shown during the first data phase in Table 5-12.

5

Table 5-12. Bridge Response to Case 2 PCI Write Burst

Data Phase	Word	Bridge Operation	Special Conditions	Performance Impact
n (last)	High	8-byte Write	None	None
n –1	Low	Gather	None	None
...	...	...	...	...
3	High	8-byte Write	None	None
2	Low	Gather	None	None
1 (first)	High	Setup + RMW	None	Add 3 PCI clocks

Table 5-13 shows the effect of several conditions on the transaction time. Rows 1 through 6 show the effects of a less-than-4-byte transfer at the high word location that occurs during a burst. The term *None* in the column titled *Performance Impact* in row 6 indicates that there are no residual performance penalties due to the events of rows 1 through 5.

Rows 7 through 9 show the effect of a less-than-4-byte transfer at the low word location that occurs during a burst. The term None in the column titled Performance Impact in row 9 indicates that there are no residual performance penalties due to the events of rows 7 and 8.

Rows 10 through 12 show the effect of a burst that ends at a low word location. The total performance impact from this burst is to add 12 PCI clocks to the transaction time.

Table 5-13. Bridge Response to Various PCI Write Bursts

Row	Data Phase	Word	Bridge Operation	Special Conditions	Performance Impact
12	n (last)	Low	RMW	None	Add 3 PCI clocks
11	n – 1	High	8-byte Write	None	None
10	n – 2	Low	Gather	None	None
9	...	...	...	...	None
8	4	High	RMW	None	Add 3 PCI clocks
7	3	Low	RMW	Less than 4-byte transfer	Add 3 PCI clocks
6	...	...	...	...	None
5	10	High	RMW	Less than 4-byte transfer	Add 3 PCI clocks
4	9	Low	Gather	None	None
3	...	...	...	...	None
2	2	High	8-byte Write	None	None
1	1 (first)	Low	Setup + Gather	None	None

Note that the performance penalty for single data phase PCI writes is three additional PCI clocks, whether the destination is the high word or the low word.

5.4.5 ECC Related BCRs

See section 9 on Error Handling for details of the BCRs associated with ECC errors.

5.4.6 Check Bit Calculation

The 660 Bridge generates the check bits based on Table 5-14 (which is shown using little-endian bit numbering).

Table 5-14. Check Bit Calculation

CB(x)	Data Bits. CB(x) = XOR of Data Bits (0 is LSb)
0	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,33,34,35,39,41,42,43,47,49,50,51,55,57,58,59,63
1	8,9,10,11,12,13,14,15,24,25,26,27,28,29,30,31,32,34,35,38,40,42,43,46,48,50,51,54,56,58,59,62
2	16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,35,37,40,41,43,45,48,49,51,53,56,57,59,61
3	0,1,2,3,4,5,6,7,16,17,18,19,20,21,22,23,32,33,34,36,40,41,42,44,48,49,50,52,56,57,58,60
4	1,2,3,7,9,10,11,15,17,18,19,23,25,26,27,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47
5	0,2,3,6,8,10,11,14,16,18,19,22,24,26,27,30,40,41,42,43,44,45,46,47,56,57,58,59,60,61,62,63
6	0,1,3,5,8,9,11,13,16,17,19,21,24,25,27,29,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63
7	0,1,2,4,8,9,10,12,16,17,18,20,24,25,26,28,32,33,34,35,36,37,38,39,48,49,50,51,52,53,54,55

5.4.7 Syndrome Decode

The syndrome consists of an 8-bit quantity which is generated by the 660 Bridge as it is comparing the 8 data bytes to the check byte. This syndrome contains the results of the comparison, as shown in Table 5-15, where:

- **`ne`** means that no error has been detected.
- **`cbx`** means that check bit x is inverted (a single-bit error).
- **`dx`** means that data bit x is inverted (a single-bit error).
- **`blank`** means that a multiple bit error has occurred.

Based on the information in the syndrome, the 660 Bridge corrects all single-bit errors and signals an error to the requesting agent on multiple-bit errors.

5

Table 5-15. Syndrome Decode

S0 S1 S2 S3 / S7 S6 S5 S4	0 0 0 0	1 0 0 0	0 1 0 0	1 1 0 0	0 0 1 0	1 0 1 0	0 1 1 0	1 1 1 0	0 0 0 1	1 0 0 1	0 1 0 1	1 1 0 1	0 0 1 1	1 0 1 1	0 1 1 1	1 1 1 1
0 0 0 0	ne	cb0	cb1		cb2				cb3							
0 0 0 1	cb4			d8			d24			d0			d16			
0 0 1 0	cb5			d9			d25			d1			d17			
0 0 1 1		d40	d41		d42			d44	d43			d45		d46	d47	
0 1 0 0	cb6			d10			d26			d2			d18			
0 1 0 1																
0 1 1 0		d56	d57		d58			d60	d59			d61		d62	d63	
0 1 1 1				d12			d28			d4			d20			
1 0 0 0	cb7			d11			d27			d3			d19			
1 0 0 1		d32	d33		d34			d36	d35			d37		d38	d39	
1 0 1 0																
1 0 1 1				d13			d29			d5			d21			
1 1 0 0		d48	d49		d50			d52	d51			d53		d54	d55	
1 1 0 1				d14			d30			d6			d22			
1 1 1 0				d15			d31			d7			d23			
1 1 1 1																

5.4.8 Dynamic Parity to ECC Mode Switching

TBD

5.5 DRAM Refresh

The memory controller provides DRAM refresh logic for system memory. The memory controller supports CAS#-before-RAS# refresh only, which provides lower power consumption and lower noise generation than RAS#-only refresh. Refresh of the odd banks of memory is staggered from the refresh of the even banks of memory to further reduce noise. First the odd RAS# lines are strobed, followed by the even RAS# lines.

Refresh requests are generated internally by dividing down the PCI_CLK. The divisor value is programmed in the refresh timer divisor register. This register is initialized to 504, a value that provides a refresh rate of 15.1us when PCI_CLK rate is 33MHz. For other PCI_CLK frequencies, the refresh rate register must be properly configured before accessing system memory.

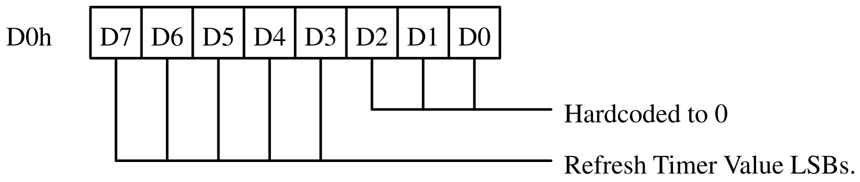
For power saving modes, refresh continues to occur even if CPU_CLK is stopped. See Section 10.3 for more details.

5

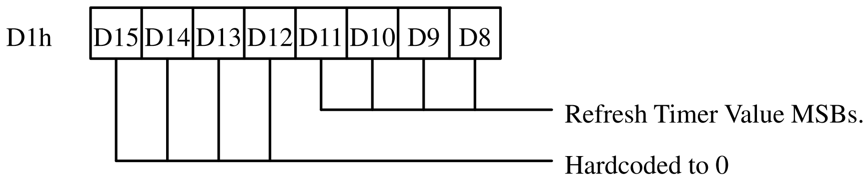
5.5.1 Refresh Timer Divisor Register

Index D0 to D1	Read/Write	Reset to F8 (D0) and 01 (D1)

The refresh timer register is a 16-bit BCR that determines the memory refresh rate. Typical refresh rates are 15.1 to 15.5 microseconds. If all DRAM in the system supports extended (slow) refresh, the refresh rate can be slower. The refresh timer is clocked by the PCI clock input to the 664 Controller. The reset value of 01F8h provides a refresh rate of 15.1 microseconds while the PCI clock is 33MHz. (01F8h equals 504 times 30ns equals 15.12us.) Bits 3–11 of the timer allow timer values from 8 to 4096.



Bits 7:3 Refresh timer (7:3) : These are the five least-significant bits of the refresh timer value.

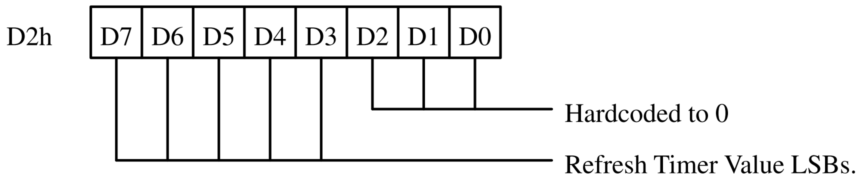


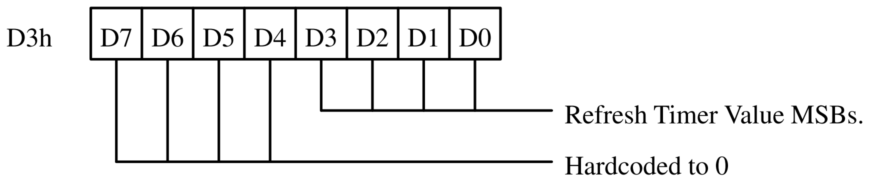
Bits 11:8 Refresh timer (11:8) : These are the four most-significant bits of the refresh timer value.

5.5.2 Suspend Refresh Timer BCR

Index D2 to D3	Read/Write	Reset to F8 (D2) and 01 (D3)

The suspend refresh timer register must be written by a half-word (16-bit) write. A read of the register is a don't care for the number of bytes.





Bits 7:3 Suspend refresh timer (7:3) : These are the five least-significant bits of the refresh timer value.

Bits 3:0 Suspend refresh timer (11:8) : These are the four most-significant bits of the refresh timer value.

5.6 DRAM Cycle Timing

TBD

5.6.1 Page Mode DRAM

TBD

5.6.2 Extended Data Out DRAM

TBD

5.7 CPU to Memory Transfers

TBD

5.7.1 CPU to Memory Read

TBD

5.7.2 CPU to Memory Write

TBD

5.7.3 Posted Writes

TBD

5.7.4 Collisions With PCI to Memory Transactions

TBD

5.7.5 Other CPU to Memory Topics
TBD

5.8 Atomic Memory Transfers
The 660 Bridge supports atomic memory transfers by supporting the 60X reservation protocol and the PCI lock protocol.

5.8.1 Memory Locks and Reservations
The 660 Bridge supports the 60X lwarx and stwcx atomic memory update protocol by broadcasting snoop cycles to the CPU bus during PCI to memory transactions. The bridge does not otherwise take any action, nor does it enforce an external locking protocol for CPU bus masters. See Section 5.8.1.1.

PCI bus masters can lock and unlock a 32-byte block of system memory in compliance with the PCI specification. This block can be located anywhere within the populated system memory space, aligned on a 32-byte boundary. Only a single lock may be in existence at any given time. The bridge does not implement complete bus locking. See Section 5.8.1.2.

5.8.1.1 CPU Reservation
The CPU indicates a reservation request by executing a memory read atomic (TT[0:3]=1101). If there is no PCI lock on the addressed block of memory, the bridge allows the transfer, but takes no other action. If there is a PCI lock on that block, the bridge terminates the CPU transfer with ARTRY#, does not access the memory location, and takes no other action.

The CPU removes a reservation by executing a memory read with intent to modify atomic (TT[0:3]=1111) or a memory write with flush atomic (TT[0:3]=1001). The bridge treats these accesses as a normal memory transfers.

5.8.1.2 PCI Lock
The bridge responds to the PCI lock request protocol in compliance with the PCI specification. If an agent requests a lock, and no PCI lock is in effect, the lock is granted. Once a PCI lock is granted, no other PCI locks are granted until the current lock is released.

The 660 Bridge prevents CPU bus masters and other PCI bus masters from reading or writing within a block of memory which is locked by a PCI bus master. CPU bus master accesses to a locked block are retried with ARTRY#. PCI bus master accesses to a locked block are retried with the PCI bus retry protocol. PCI and CPU bus master accesses to other areas of system memory are unrestricted.

PCI to memory transactions cause the bridge to broadcast a snoop cycle to the CPU bus. When a PCI agent is granted a memory block lock, the bridge broadcasts a write with flush (TT[0:3]=0001) cycle on the CPU bus, which causes the L1 and L2 caches to invalidate that sector (if there is an address match). This insures that there will not be a cache hit during a CPU bus accesses to a memory block which is locked by a PCI agent.

5.8.1.3 PCI Lock Release
The bridge responds to the PCI lock release protocol in compliance with the PCI Specification. If an agent releases the lock that it owned, the bridge releases the lock. The bridge generates a normal snoop cycle on the CPU bus.

5.9 DRAM Module Electrical Considerations

The 660 Bridge directly drives up to eight 168 pin DRAM modules, which typically exhibit an input capacitance of less than 20pf. Table 5-16 shows some maximum input capacitance numbers that are typical of the various DRAM modules on the market. System designers may wish to buffer the DRAM if the system design requires the use of a set of DRAM modules such that the combined input capacitance on the address or WE# lines exceeds 400pf.

Table 5-16. Typical DRAM Module Maximum Input Capacitance

SIMM Type	SIMM Size	Maximum Input Capacitance	
		Address	WE#
72-pin	4 Meg	50pf	50pf
	8 Meg	90pf	95pf
	16 Meg	80pf	95pf
	32 Meg	160pf	190pf
168-pin	8M, 16M, 32M,128M	13pf	13pf
	64M	18pf	18pf

Section 6
The PCI Bus

6

The 660 Bridge supports a 32-bit PCI expansion bus at frequencies up to 33MHz. The PCI bus is compatible with the PCI Specification, revision 2.0.

The PCI bus can be run an one-third, one-half, or the same frequency as the 60X CPU bus. The 664 Controller automatically detects the frequency ratio.

6.1 PCI Interface Memory Mapping

Table 6-1 shows the mapping of PCI memory accesses to system memory. This is the mapping of the bridge as a PCI target when a PCI device performs memory accesses. The bridge is never a target on PCI I/O, PCI configuration, and PCI interrupt acknowledge transactions that are initiated by masters on the PCI bus.

When the IGN_PCI_AD31# signal is asserted, the 660 Bridge maps PCI memory accesses from 0 to 2G directly to system memory at 0 to 2G. If the IGN_PCI_AD31# signal is negated, PCI memory accesses in the 0 to 2G address range are ignored by the 660 Bridge.

The 60X CPU can generate PCI memory accesses with PCI addresses from 0 to 1G – 2M. The range of addresses to locate PCI memory that can be addressed by both PCI devices and the 60X CPU is from 0 to 1G – 16M.

The 660 Bridge maps PCI master memory accesses from 2G to 4G as system memory from 0 to 2G. All PCI master accesses to system memory are snooped by the caches.

Table 6-1. 660 Bridge Mapping of PCI Memory Space

PCI Bus Address	Other Conditions	Target Cycle Decoded	Target Address	Notes
0 to 2G	IGN_PCI_AD31# Deasserted	Not Decoded	N/A	
	IGN_PCI_AD31# Asserted	System Memory *	0 to 2G	Snooped by caches.
2G to 4G		System Memory *	0 to 2G	Snooped by caches.
*Memory does not occupy this entire address space. Accesses to unoccupied space are not decoded.				

6.2 PCI Operations

6.2.1 PCI Arbitration

The 660 Bridge requires an external PCI arbiter such as those provided in the Intel SIO or IBM27-82376.

6.2.2 PCI Lock

The 660 Bridge does not set PCI locks when acting as the PCI master. The PCI_LOCK# signal in the 660 Bridge supports resource locking of one 32-byte cache sector (block) of system memory. Once a PCI lock is established, the block address is saved. Subsequent accesses to that block from other PCI bus masters or from the CPU bus are retried until the lock is released.

The bridge generates a flush-sector snoop cycle on the CPU bus when a PCI bus master sets the PCI lock. The flush-sector snoop cycle causes the L1 and L2 caches to invalidate the locked block, which prevents cache hits on accesses to locked blocks. If the L1 contains modified data, the PCI cycle is retried and the modified data is pushed out to memory.

6

Note: The 60X processors do not have bus-locking functions. Instead, they use the *load reserve* and *store conditional* instructions (lwarx and stwcx) to implement exclusive access. To work with the lwarx and stwcx instructions, the 660 Bridge generates a flush-sector operation to the CPU in response to the PCI read that begins a PCI lock.

6.3 CPU Access to PCI targets

CPU accesses to addresses within the 2G to 4G range produce the various PCI transactions. When a CPU access is decoded for the PCI, the 660 Bridge requests the PCI bus and runs a PCI cycle after the PCI arbiter grants the bus.

6.3.1 CPU to PCI Read

If the CPU to PCI cycle is a read, a PCI read cycle is run. If the PCI read cycle completes, the data is passed to the CPU and the CPU cycle is ended. If the PCI cycle is retried, the CPU cycle is retried. If a PCI master access to system memory is detected before the PCI read cycle is run then the CPU cycle is retried (and no PCI cycle is generated).

Minimum read time from the PCI is two CPU_CLKs plus three PCI_CLKs plus one CPU_CLK. If the PCI is run at 33MHz and the 60X CPU bus is at 66MHz, the minimum PCI read cycle is nine CPU clocks. This assumes that the clock period that TS# goes active is the first clock counted and the clock period that TA# goes active is the last clock counted. The maximum throughput is four bytes of data every 10 CPU clocks (every five PCI clocks).

6.3.2 CPU to PCI Write

If the CPU to PCI cycle is a write, a PCI write cycle is run. CPU to PCI I/O writes are not posted, as per the *PCI Local Bus Specification* version 2.1. If the PCI transaction is retried, the Bridge retries the CPU.

CPU to PCI memory writes are posted, so the CPU write cycle is ended as soon as the data is latched. If the PCI cycle is retried, the Bridge retries the cycle until it completes.

Minimum write time to the PCI is four CPU clocks. When the PCI write cycle occurs, it takes a minimum of three PCI clocks. The maximum throughput is four bytes of data every eight CPU clocks (every four PCI clocks).

6.3.2.1 Eight-Byte Writes to the PCI (Memory and I/O)

The 660 Bridge supports 1-byte, 2-byte, 3-byte, and 4-byte transfers to and from the PCI. The 660 Bridge also supports 8-byte memory and I/O writes (writes only, not reads) to the PCI bus. This enables the use of the 604 store multiple instruction to PCI devices.

When an 8-byte write to the PCI is detected, it is not posted initially. Instead the CPU waits until the first 4-byte write occurs, then the second 4-byte write is posted. If the PCI retries on the first four byte transfer or a PCI master access to system memory is detected before the first 4-byte transfer then the CPU is retried. If the PCI retries on the second 4-byte transfer then the 660 Bridge retries the PCI write.

Minimum write time is eight bytes for every 12 CPU clocks (six PCI clocks).

6.3.3 PCI Retry

CPU to PCI transactions that the PCI target retries, cause the 660 Bridge to deassert its PCI_REQ# (the Bridge follows the PCI retry protocol). The Bridge stays off of the PCI bus for two PCI clocks before reasserting PCI_REQ# (or FRAME#, if the PCI bus is idle and the PCI_GNT# to the Bridge is active).

6.3.4 CPU Access to PCI Configuration Space

The 660 Bridge allows access to PCI configuration space via two different mechanisms.

CPU accesses to addresses from 2G + 8M to 2G + 16M or from 8000 0CFCh to 8000 0CFFh (the configuration data register) when the configuration address indicates a device number greater than zero generate a PCI type 0 configuration cycle. (See Section 4.2 and 4.3.)

Type 1 configuration transactions can also be initiated via the PCI/BCR configuration address and data BCRs.

6.3.4.1 PCI Type 0 Configuration Transaction (650 Compatible)

CPU accesses to the address range 2G+8M to 2G+16M (as shown in Table 6-2) cause the bridge to arbitrate for the PCI bus and then to execute a type 0 PCI configuration transaction as described in the PowerPC Reference Platform Specification and implemented by the IBM27–82650 PowerPC to PCI Bridge.

Table 6-2. PCI Configuration Addresses

Bridge Control Register	CPU Bus Address	R/W	Bytes	Cycle and IDSEL
PCI Slot 0 Configuration Space	8080 08xx	R/W	4	PCI Config, IDSEL = PCI_AD[11]
PCI Slot 1 Configuration Space	8080 10xx	R/W	4	PCI Config, IDSEL = PCI_AD[12]
PCI Slot 2 Configuration Space	8080 20xx	R/W	4	PCI Config, IDSEL = PCI_AD[13]
PCI Slot 3 Configuration Space	8080 40xx	R/W	4	PCI Config, IDSEL = PCI_AD[14]
PCI Slot 4 Configuration Space	8080 80xx	R/W	4	PCI Config, IDSEL = PCI_AD[15]
PCI Slot 5 Configuration Space	8081 00xx	R/W	4	PCI Config, IDSEL = PCI_AD[16]
PCI Slot 6 Configuration Space	8082 00xx	R/W	4	PCI Config, IDSEL = PCI_AD[17]
PCI Slot 7 Configuration Space	8084 00xx	R/W	4	PCI Config, IDSEL = PCI_AD[18]

When the PCI bus is acquired, the address is driven for one PCI clock before PCI_FRAME# is asserted. This allows the IDSELs to be resistively connected to the PCI_AD[31:0] bus at the system level. This method of accessing PCI configuration space does not allow access to the PCI configuration registers in the bridge chip and it should not be used unless required to maintain 650 compatibility.

6.3.4.2 PCI Type 0 Configuration Transaction (CFC/CF8)

PCI configuration space can be accessed via the CFC/CF8 register pair as described in section 4.3.

6.3.5 PCI Interrupt Acknowledge Transaction

Reading the interrupt acknowledge address (BFFF FFF0h) causes the bridge to arbitrate for the PCI bus and then to execute a standard PCI interrupt acknowledge transaction. The system interrupt controller claims the transaction and supplies the 1-byte interrupt vector. There is no physical interrupt vector BCR in the bridge. Other PCI bus masters can initiate interrupt acknowledge transactions.

6.4 PCI Access to System Memory

PCI memory reads and writes by PCI bus masters are decoded by the 660 Bridge to determine if they access system memory. (PCI memory reads and writes to addresses from 2G to 4G on the PCI bus are mapped by the 660 Bridge as system memory reads and writes from 0G to 2G.)

When a PCI access is decoded as a system memory read or write, the memory and 60X CPU bus are requested and, when granted, a snoop cycle to the 60X CPU bus and a memory cycle to system memory are generated. If the processor indicates a snoop hit in the L1 cache (ARTRY# asserted), the memory cycle is aborted and the PCI cycle is retried. The CPU then does a snoop push. The L2 cache does not need to do a snoop push because it is write-through, and, therefore, system memory always contains the result of all write cycles.

Section 3.3 contains a detailed description of the snoop process. Table 6-3 shows the TT[0:3] (transfer type) of snoop operations generated to the CPU bus when a PCI master accesses system memory.

Table 6-3. Types of Snoop Cycles for PCI to Memory Operations

PCI Bus Cycle	CPU Bus Snoop, non 603		CPU Bus Snoop, 603	
	Operation	TT[0:4]	Operation	TT[0:4]
Memory Read	Clean	00000	Single-Beat Read	01010
Memory Write	Flush Sector	00100	Single-Beat Write with Flush	00010
Initiate Lock (Read)	Single-Beat Write with Flush	00010	Single-Beat Write with Flush	00010

If the PCI access is a memory burst write access, the 664 Controller PCI interface performs data gathering before initiating the cycle to the memory controller. The data gathering involves combining two PCI write cycles into one memory write cycle if the address of the first write cycle is even.

Minimum initial write access time when the 60X CPU bus is 66MHz and the PCI bus is 33MHz is 5-1-1-1 -1-1-1-1 PCI clocks for 4-4-4-4 -4-4-4-4 bytes of data (12 PCI clocks for 32 bytes of data). Subsequent data phases of the same burst are generally serviced at -3-1-1-1 -1-1-1-1 (10 PCI clocks for 32 bytes of data). See Section 6.4.2.3.

If the PCI access is a memory burst read access, the 664 Controller PCI interface performs memory pre-fetching when it initiates cycles to the memory controller. The pre-fetching involves loading or pre-loading 32 bytes from the memory for eight 4-byte PCI read cycles. Pre-fetching is only done within the same cache line.

Minimum initial read access time when the 60X CPU bus is 66MHz and the PCI bus is 33MHz is 5-1-1-1 -1-1-1-1 PCI clocks for 4-4-4-4 -4-4-4-4 bytes of data (12 PCI clocks for 32 bytes of data). Subsequent data phases of the same burst are generally serviced at -6-1-1-1 -1-1-1-1 (13 PCI clocks for 32 bytes of data). See Section 6.4.2.2.

PCI accesses to system memory are not limited to 32 bytes. PCI burst-mode accesses are limited only by the size of memory, PCI bus latency restrictions, and the PCI disconnect counter.

6.4.1 660 Bridge Responses to PCI_C/BE[3:0] Bus Commands

Table 6-4 shows the bridge responses to bus commands it receives from other masters on the PCI bus.

Table 6-4. 660 Bridge Responses to PCI_C/BE[3:0] Bus Commands

C[3:0]	PCI Bus Command	660 Bridge Operation During PCI to Memory Transaction
0000	Interrupt Acknowledge	No response
0001	Special Cycle	No response
0010	I/O Read	No response
0011	I/O Write	No response
0100	Reserved	No response
0101	Reserved	No response
0110	Memory Read	Memory Read. See Table 6-1.
0111	Memory Write	Memory Write. See Table 6-1.
1000	Reserved	No response
1001	Reserved	No response
1010	Configuration Read	No response
1011	Configuration Write	No response
1100	Memory Read Multiple	Memory Read. See Table 6-1.
1101	Dual Address Cycle	No response
1110	Memory Read Line	Memory Read. See Table 6-1.
1111	Memory Write and Invalidate	Memory Write. See Table 6-1.

6

6.4.2 PCI to Memory Burst Transfers

PCI to memory burst transfers continue to normal completion unless one of the following occurs:

1. The initiating PCI bus master disconnects. The 660 handles all master disconnects correctly.
2. The 660 Bridge target disconnects on a 1 M boundary. The Bridge disconnects on all 1M boundaries.
3. The 660 Bridge target disconnects because the PCI disconnect timer has timed out. See section 6.4.2.1.
4. The CPU retries the snoop cycle that the 660 Bridge broadcast on the CPU bus. In this case the Bridge target retries the PCI bus master. (Note that L2 hits do not affect the PCI to memory transaction. Read hits have no effect, and write hits cause the internal L2 to invalidate the block.)
5. The 660 Bridge will target disconnect the PCI bus master if either the refresh timer (see section 4.3.49) or the suspend refresh timer (see section 4.3.50), whichever is active, times out. In either case the Bridge will disconnect at the end of the current data phase for writes, or at the end of the current cache block, for reads.

6.4.2.1 PCI Disconnect Counter

Index 42	Read/Write	Reset to 00

This BCR determines the maximum number of PCI clocks (from 1 to 255) that a PCI master can burst access system memory before a target disconnect is initiated. Loading the counter with 00 disables it. The counter begins to count down when FRAME# is sampled active, and is reloaded between PCI to memory accesses. If the counter reaches 0, the 660 Bridge target disconnects the PCI bus master.

6.4.2.2 Detailed Read Burst Sequence Timing

TBD

6.4.2.3 Detailed Write Burst Sequence Timing

The basic PCI to memory write sequence is 5–1–1–1 –1–1–1–1 –3–1–1–1 –1–1–1–1 –3..., giving a peak burst write rate of 32 bytes in 10 PCI clocks, or about 106MBps with a 33MHz PCI clock. This scenario holds while the RAS# timer (10us typical) does not time out, the burst remains within the same 4K memory page, and no refresh is requested (15us typ).

The actual detailed write sequence is affected by several factors, such as refresh requests, memory arbitration delays, page and/or bank misses, and cache boundary alignment. Table 6-5 shows the details of the various sequences that a PCI to memory burst write will experience, depending on the address (relative to a cache block boundary) of the first data phase of the transaction. The starting address of the numbering sequence shown on the top row was arbitrarily chosen as xx00, and could be any 32–byte aligned boundary. The times shown in Table 6-5 are in PCI clock cycles, and do not include any cycles that the PCI master spends acquiring the PCI bus from the PCI bus arbiter. The initial data phase is timed from the assertion of FRAME# to the PCI clock at which the PCI master samples TRDY# active. Subsequent data phase times are from the PCI clock at which the previous TRDY# was sampled active to the PCI clock at which the current TRDY# is sampled active.

Table 6-5. PCI to Memory Write Burst Sequence Timing

S								S									
00	04	08	0C	10	14	18	1C	20	24	28	2C	30	34	38	3C	40	...
W	1	1	1	Y	1	Z	1	X	1	Z	1	Z	1	Z	1	X	...
	W	1	1	Y	1	Z	1	X	1	Z	1	Z	1	Z	1	X	...
		W	1	1	1	Y	1	X	1	Z	1	Z	1	Z	1	X	...
			W	1	1	Y	1	X	1	Z	1	Z	1	Z	1	X	...
				W	1	1	1	G	1	Z	1	Z	1	Z	1	X	...
					W	1	1	G	1	Z	1	Z	1	Z	1	X	...
						W	1	G	1	Z	1	Z	1	Z	1	X	...
							W	G	1	Z	1	Z	1	Z	1	X	...
								W	1	1	1	Y	1	Z	1	X	...

S indicates a cache block boundary at 0 mod 32. Snoops are broadcast to the CPU bus when a PCI burst crosses one of these boundaries.

W is a function of a 1.5 PCI clock snoop delay and memory arbitration delays. If the CPU is accessing memory when the PCI agent begins the memory write burst, the 660 Bridge waits until the CPU completes the current CPU access before allowing the PCI to memory write to proceed. If the RAS# watchdog timer has timed out, the memory controller will precharge the RAS# lines, and if the refresh timer has timed out, the memory controller will do a refresh operation.

W (min)	= 5	This occurs when the memory controller is idle and no refresh or RAS# timeout occurs.
W (typ)	= 6 or 7	This occurs if the memory controller is in the middle (beat 3 of 4) of serving a CPU burst transfer when the PCI burst starts, and no refresh or RAS# timeout occurs.
W (max)	= 10	This occurs when the CPU is just starting a burst transfer to memory, after which a refresh happens to be required.

X is a function of snoop delays only. Whenever the memory access crosses a cache block boundary, the Bridge broadcasts a snoop cycle on the CPU bus. (Due to the posted write buffer structure, delays incurred by crossing a page boundary here do not show up until later in the sequence.)

6

X= 3 Always. (The only benifit to disabling PCI snooping or enabling pre–snooping is to reduce this delay to 1. Otherwise neither function increases performance.)

Y is a function of memory latency. This page and/or bank miss delay can only be incurred at a page boundary, but shows up here due to the posted write buffer structure. The Bridge has a 4 x 4 posted PCI write buffer, which allows it to accept data phases from the PCI bus while the memory controller is busy servicing page misses. This minimizes the transfer delays caused by these memory overhead functions.

Y (typ)	= 1	This occurs for a page hit with no refresh. This is also the minimum.
Y (mid)	= 2	This occurs for a page miss with no refresh.
Y (max)	= 4	This occurs for a refresh (which also forces a page miss).

Z is a function of a subset of the **W** factors (RAS# timeouts and refresh operations). This delay is only incurred due to a RAS# timeout or refresh request that has occured since the last **W**, **Y**, **Z**, or **G**.

Z (typ)	= 1	This occurs for no refresh and no RAS# timeout.
Z (max)	= 2 to 4	This occurs for either a RAS timeout or a refresh operation.

G is the combination of **X** and **Y**, and is equal to the longer of **X** and **Y**.

Section 7
L2 Cache

The L2 cache controller in the 664 Controller chip controls external tag RAM and data SRAMs for a second level cache. The L2 cache is for the system memory space from 0 to 1G. The cache architecture is direct-mapped, look-aside, and write-through. Either synchronous (burst mode) SRAMs or asynchronous SRAMs are supported for a choice of best performance or lowest cost.

7.1 L2 Controller Features

7.1.1 Cache Size

The cache size is determined externally by the tagRAM and data SRAM sizes and how they are attached. The L2 cache controller supports all cache sizes. Loading constraints limit practical cache size to the 256K to 1M range for 66MHz operation.

7.1.2 Cache Responses

On memory reads, the L2 cache only responds to bursts from the CPU. Single–beat reads are ignored. Table 7-1 shows the actions taken by the L2 cache based on transfer type and single-beat or burst mode.

Note: The 60X cache inhibit (CI#) signal is not used because cache-inhibited bus operations are always single-beat.

7.1.3 Cache Configuration

The L2 controller does not have any configuration bits for the size or organization of the tag and data SRAMs. The connections to the 60X address bus determine the size of the tag RAM and data SRAMs. There is a configuration bit that identifies the SRAM type as burst or asynchronous.

Since the L2 cache only caches addresses that are present in system memory, it does not have any configuration bits for cacheable address space.

7.1.4 L2 Performance

In a system using burst SRAMs, access time is 3, 1, 1, 1 CPU_CLKs for 8, 8, 8, 8 bytes of data (six CPU_CLKs for 32 bytes of data). In a system using asynchronous SRAMs, access time is 3, 2, 2, 2 CPU_CLKs for 8, 8, 8, 8 bytes of data (ten CPU_CLKs for 32 bytes of data).

7.2 L2 Cache Responses to CPU Bus Operations

The L2 caches data for the CPU bus masters. It supplies data on read hits, and snarfs data on writes. Table 7-1 details the operation of the L2 cache for various transfer type codes issued by a CPU bus master. The 660 Bridge does not use TT[4].

Table 7-1. L2 Cache Responses to CPU Bus Cycles

TT[0:3]	Type	CPU Bus Cycle	Cache Hit Action	Cache Miss Action
0000		Clean sector	Ignore	Ignore
0001	Single	Write with flush	Invalidate	Ignore
	Burst	Write with flush	Update as data is written to memory	Update as data is written to memory
0010		Flush sector	Invalidate	Ignore
0011	Single	Write with kill	Invalidate	Ignore
	Burst	Write with kill	Update as data is written to memory	Update as data is written to memory
0101	Single	Read	Ignore	Ignore
	Burst	Read	Claim cycle and supply data to CPU bus	Update as data is read from memory
0110		Kill sector	Invalidate	Ignore
0111	Always Burst	Read with intent to modify	Claim cycle and supply data to CPU bus	Ignore
1000		Reserved	Ignore	Ignore
1001	Always Single	Write with flush atomic	Invalidate	Ignore
1010		External control out	Ignore	Ignore
1011		Reserved	Ignore	Ignore
1100		TLB invalidate	Ignore	Ignore
1101	Single	Read atomic	Ignore	Ignore
	Burst	Read atomic	Claim cycle and supply data to CPU bus	Update as data is read from memory
1110		External control in	Ignore	Ignore
1111	Always Burst	Read with intent to modify atomic	Claim cycle and supply data to CPU bus	Ignore

7.3 L2 Cache Responses to PCI Bus Mastered Transactions

The 660 Bridge maintains L2 coherency during PCI to memory transactions as shown in Table 7-2 for non-603 operation, and in Table 7-3 for 603 operation. The L2 does not supply data directly to the PCI bus. The L2 is not updated during a PCI transaction.

Table 7-2. L2 Operations for PCI to Memory Transactions, Non-603 Mode

PCI Bus Transaction	CPU Bus Broadcast Snoop Cycle		L2 Operation	
	Operation	TT[0:4]	L2 Hit	L2 Miss
Memory Read	Clean	00000	Ignore	Ignore
Memory Write	Flush Sector	00100	Invalidate Block	Ignore
Initiate Lock (Read)	Single-Beat Write with Flush	00010	Invalidate Block	Ignore

Table 7-3. L2 Operations for PCI to Memory Transactions, 603 Mode

PCI Bus Transaction	CPU Bus Broadcast Snoop Cycle		L2 Operation	
	Operation	TT[0:4]	L2 Hit	L2 Miss
Memory Read	Single-Beat Read	01010	Ignore	Ignore
Memory Write	Single-Beat Write with Flush	00010	Invalidate Block	Ignore
Initiate Lock (Read)	Single-Beat Write with Flush	00010	Invalidate Block	Ignore

7

7.4 Error Checking Support

The 660 Bridge L2 cache can be configured as 64 or 72 bits wide. The 64 bit configuration offers decreased cost and no error checking. The 72 bit configuration supports parity bit storage and checking. If used, parity checking must be enabled via the error enable 2 register (index C4), bit 3. The parity bits are then stored when an L2 entry is updated. Parity checking is done when data is supplied from the L2.

ECC mode operation does not effect SRAM error checking because ECC mode operations are confined to the 660 Bridge to DRAM interface. The SRAM is a CPU bus device which is not connected to the DRAM subsystem.

7.5 External L2 Cache Operation

The onboard L2 cache controller can be disabled to incorporate an external L2. The only constraints that the 660 Bridge imposes on such an external L2 are those imposed on all CPU bus agents (see Section 3.1). The internal L2 cache controller can be disabled to accommodate an external L2 cache with greater associativity, write-back capability, or other improved capabilities.

7.6 TagRAM

The L2 controller requires external tagRAM. The following tagRAM parts are known to be supported by the 660 Bridge: IDT71B74S10 (asynchronous), and IDT71215S10 and IDT71216S10 (both synchronous). These are 10ns parts, but tagRAM speed should be determined based on total system considerations. Typical systems will operate with 12ns tagRAMs at 66MHz.

Connectivity of the 660 Bridge with an IDT71216 16k x 15 TagRAM is illustrated by the example system implementation schematics for 256K, 512K, and 1M cache sizes, using the IDT 71216 tagRAM. Several other variations are mentioned in the notes.

7.7 SRAM

The 660 L2 controller requires external SRAM, which can be either synchronous or asynchronous.

7.7.1 Synchronous

Synchronous SRAM must be capable of linear burst ordering (00–01–10–11–00–...). When synchronous SRAM is used, the SRAM_ALE output is unused (and driven high) because the synchronous SRAMs contain an address latch.

The speed of the SRAM should be determined based on total system considerations. However, typical systems will operate with 9ns devices at 66MHz.

7.7.2 Asynchronous

While asynchronous SRAM is in use, the SRAM_ALE signal controls an external address latch (74F373, for example).

The speed of the SRAM should be determined based on total system considerations. However, typical systems will operate with 15ns devices at 66MHz.

7.7.3 Dual (Sync and Async) Capable Systems

Asynchronous SRAM uses the SRAM_ADS#/ADDR0 and SRAM_CNT_EN#/ADDR1 signals as the two least significant address bits, while synchronous SRAM uses them as control signals. Synchronous SRAMs internally latch in the initial value of the two least significant address bits from the CPU address bus. Systems designed to support either type of SRAM must use either jumpers or (fast) external logic to provide the necessary signal routing.

7.8 SRAM and TagRAM Connections

Figure 7–1 through Figure 7–11 show a detailed example of the connections between the 660 Bridge, the SRAM, and the TagRAM. The information in these figures is presented in block diagram form and illustrates several different combinations of synchronous and asynchronous SRAM and tagRAM that can be used to create different sizes of L2. Table 7-4 is a guide to the configuration illustrations. For example, to use synchronous SRAM and asynchronous tagRAM to implement a 512K L2, see Figure 7–2 for an SRAM connection example, and Figure 7–11 for a tagRAM connection example.

Table 7-4. Index of SRAM and TagRAM Example Configurations

SRAM	Tag RAM	256K L2		512K L2		1M L2	
		SRAM	TagRAM	SRAM	TagRAM	SRAM	TagRAM
Sync	Sync	Figure 7–1	Figure 7–7	Figure 7–2	Figure 7–8	Figure 7–3	Figure 7–9
Sync	Async	Figure 7–1	Figure 7–10	Figure 7–2	Figure 7–11	—	—
Async	Sync	Figure 7–4	Figure 7–7	Figure 7–5	Figure 7–8	Figure 7–6	Figure 7–9
Async	Async	Figure 7–4	Figure 7–10	Figure 7–5	Figure 7–11	—	—

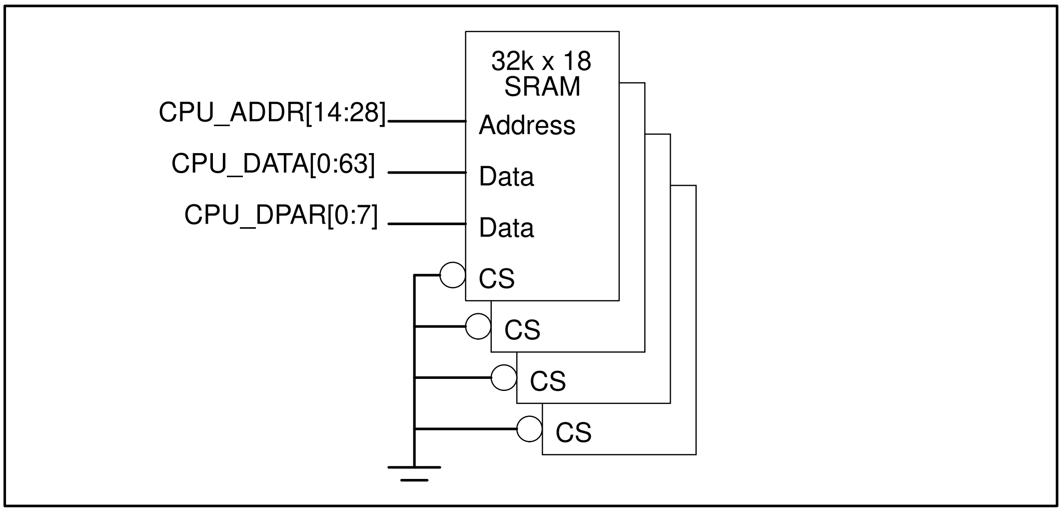


Figure 7–1. Synchronous SRAM, 256K L2

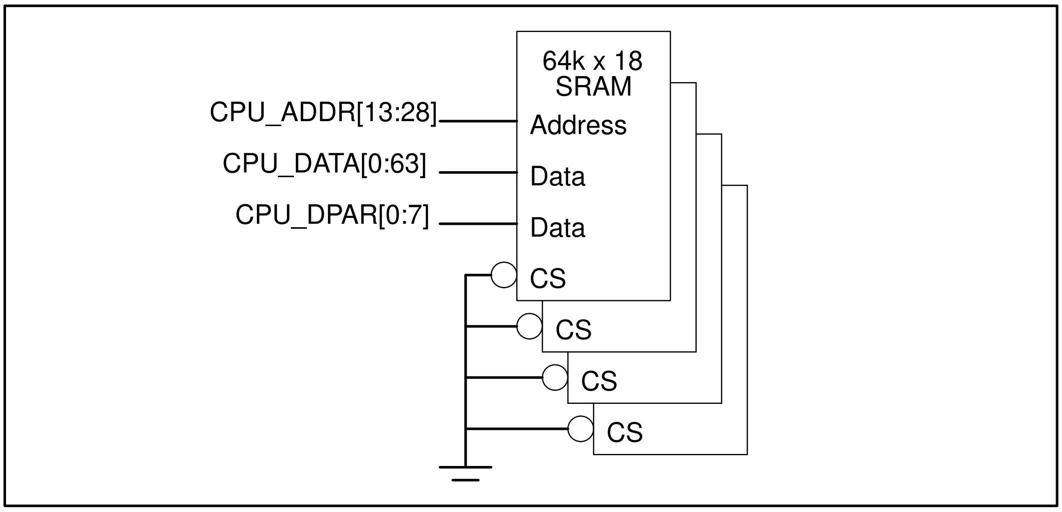


Figure 7–2. Synchronous SRAM, 512K L2

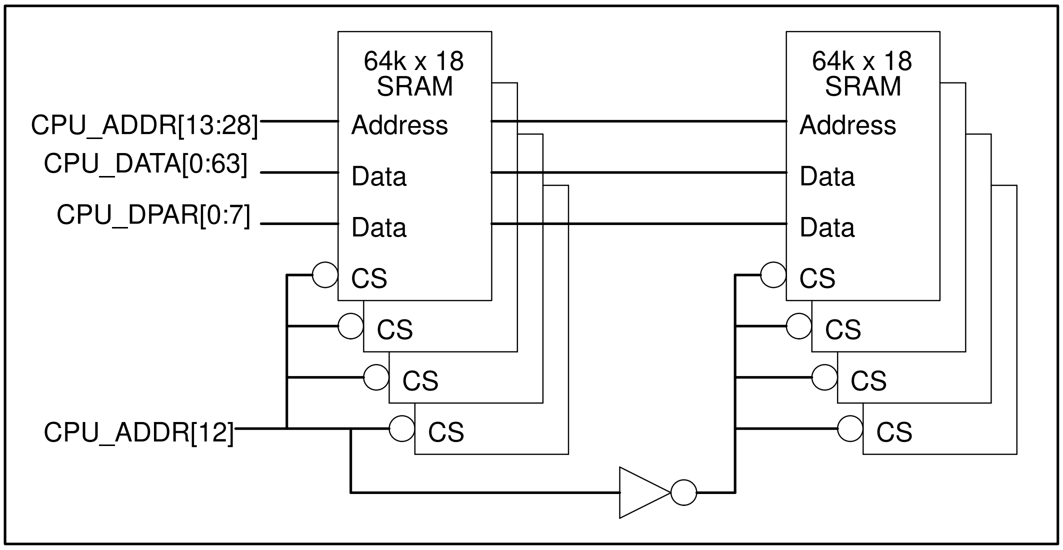


Figure 7–3. Synchronous SRAM, 1M L2

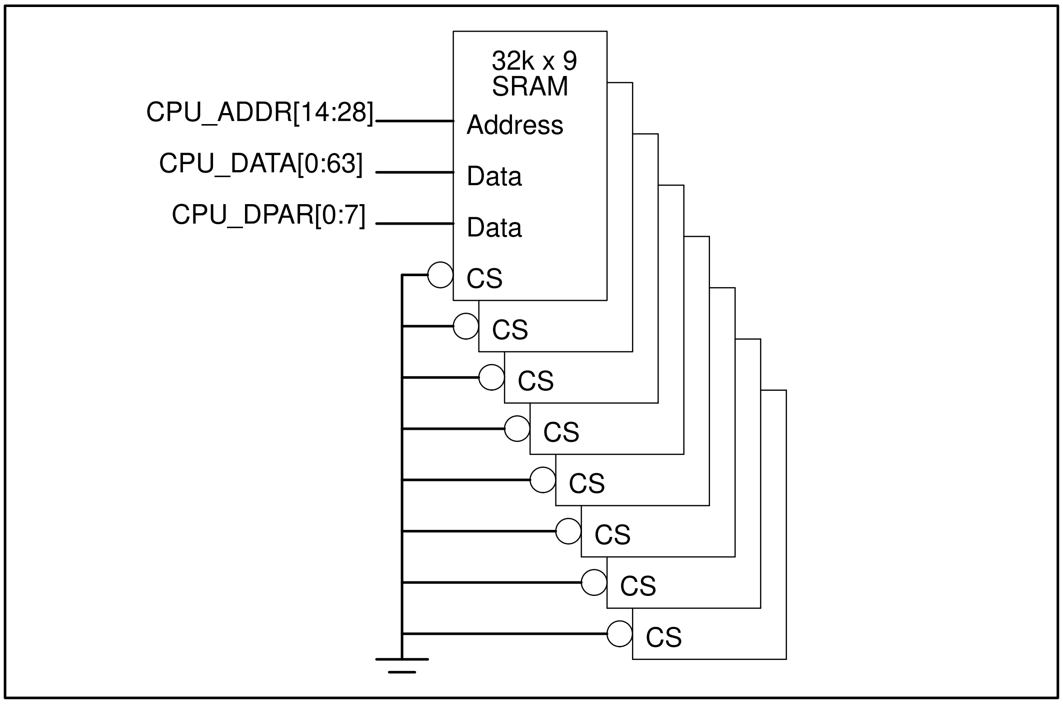


Figure 7–4. Asynchronous SRAM, 256K L2

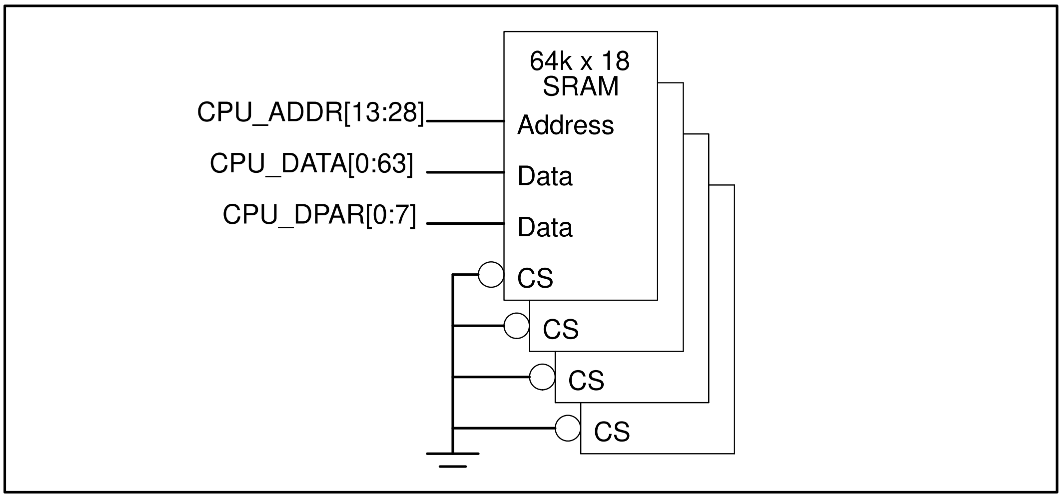


Figure 7–5. Asynchronous SRAM, 512K L2

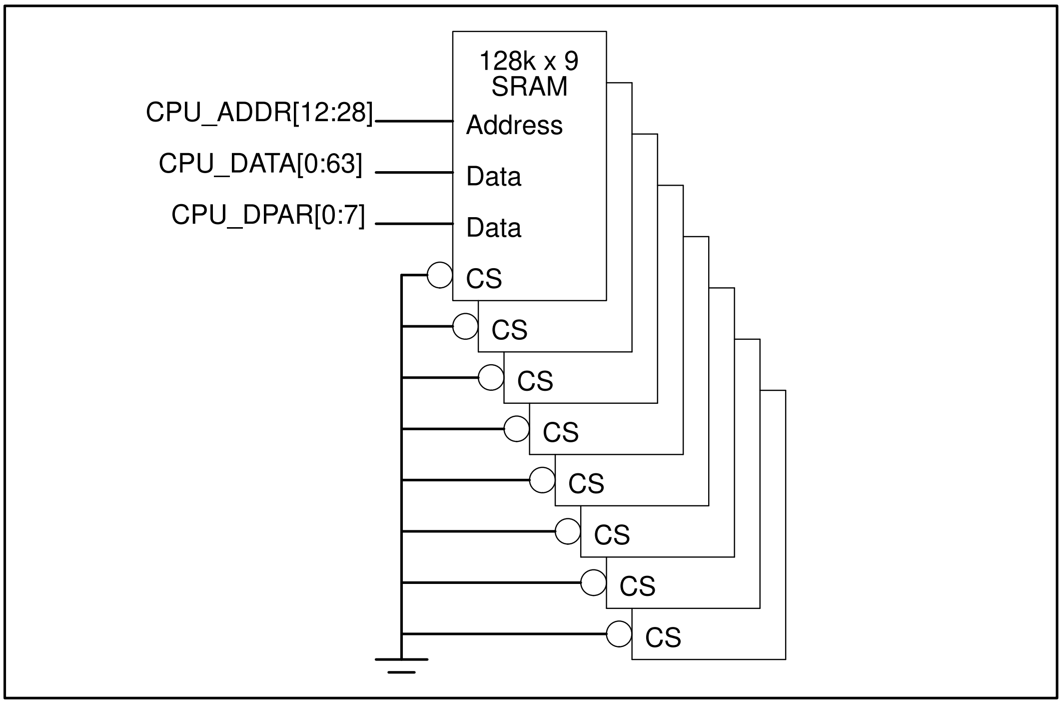


Figure 7–6. Asynchronous SRAM, 1M L2

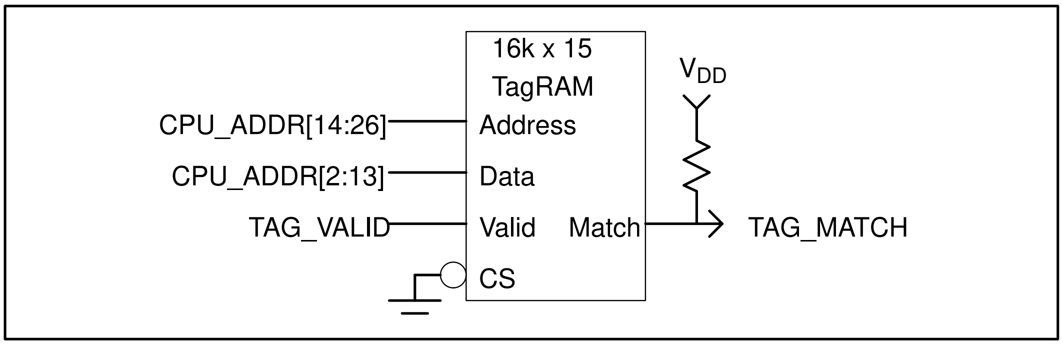


Figure 7–7. Synchronous TagRAM, 256K L2

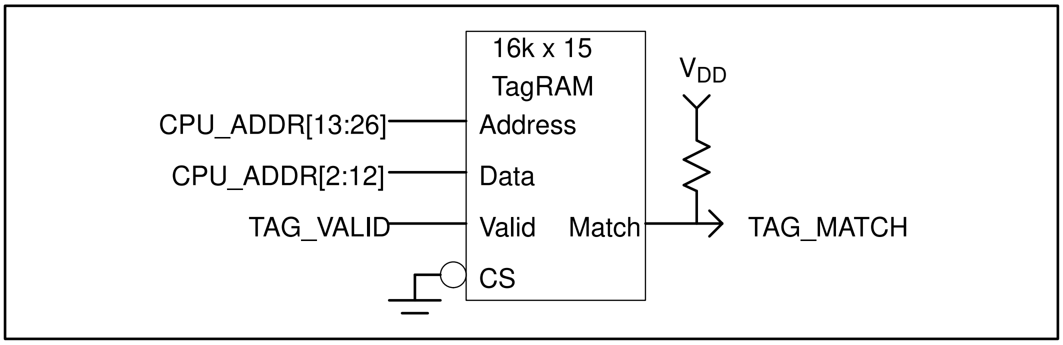


Figure 7–8. Synchronous TagRAM, 512K L2

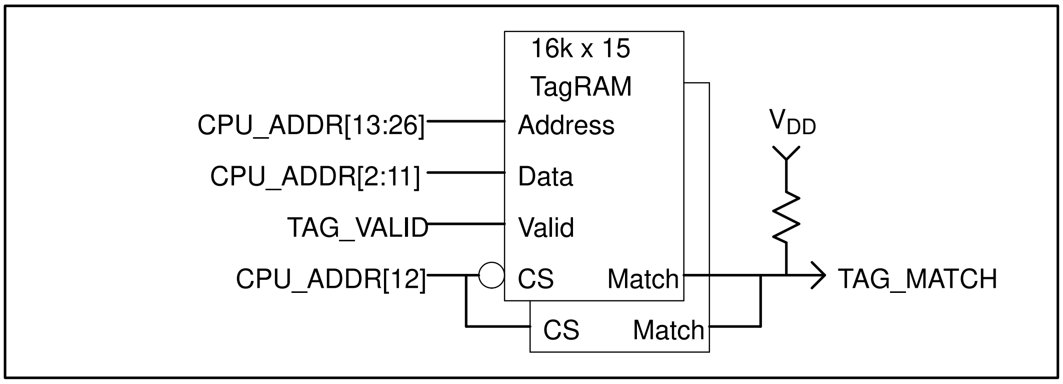


Figure 7–9. Synchronous TagRAM, 1M L2

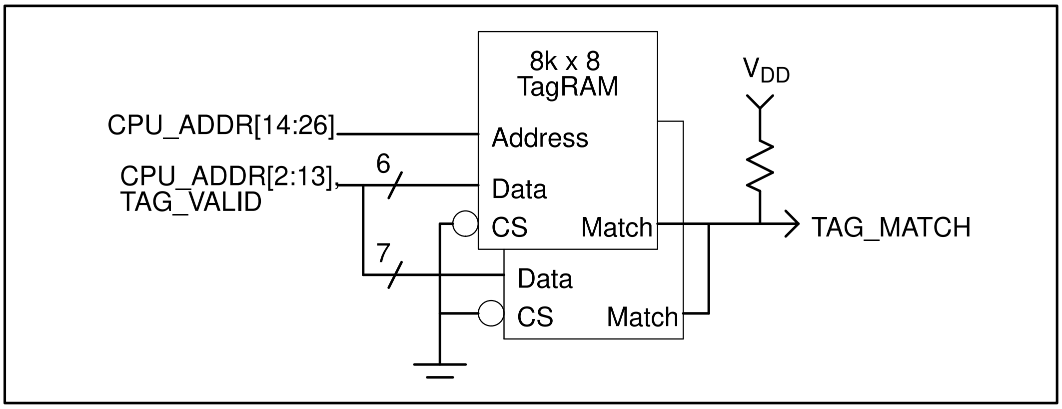


Figure 7–10. Asynchronous TagRAM, 256K L2

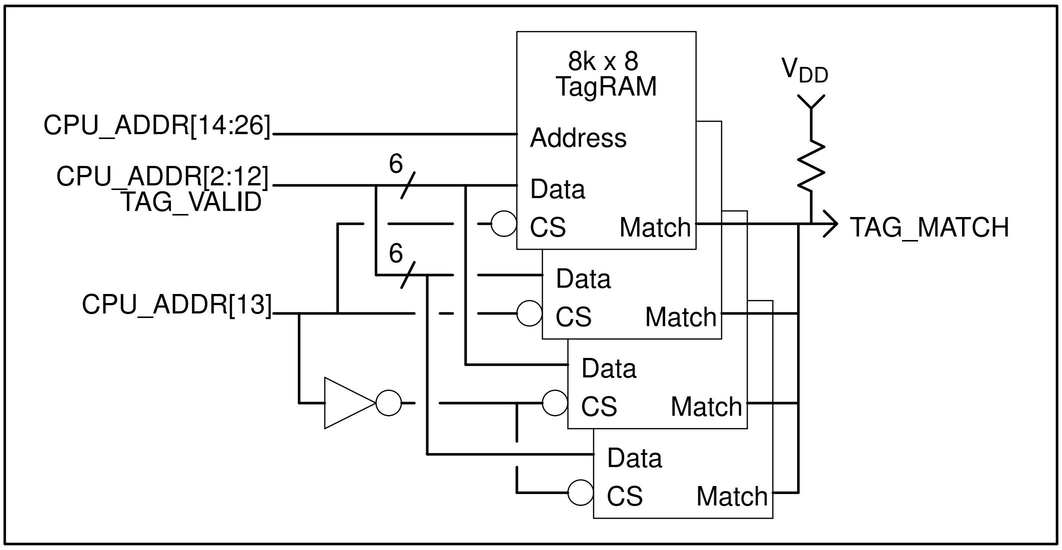


Figure 7–11. Asynchronous TagRAM, 512K L2

7

Section 8
ROM Operations

The *PowerPC Reference Platform Specification* allocates the upper 8M of the 4G CPU address space as ROM space. The 660 Bridge implements a 2M ROM space from 4G–2M to 4G. The Bridge provides two boot ROM device access methods which minimize pin and package count while still allowing a byte-wide ROM device to source 8-byte wide data. In either case, the designer can choose the ROM device from EEPROM, non–volatile RAM, EPROM, PROM, ROM, PCMCIA, or combination of technologies to meet timing and system requirements.

The first ROM access method (remote ROM mode) attaches the ROM device to an external PCI agent which supports the PowerPC Reference Platform ROM space map and access protocol. CPU bus master transfers to ROM space are forwarded to the PCI bus and claimed by the PCI agent, which supplies the ROM device data. The IBM27–82376 PCI to ISA Bridge offers this remote ROM capability. The ROM device attaches to the IBM PCI to ISA Bridge through the ISA bus lines, thereby saving a PCI bus load. The IBM PCI to ISA Bridge also supports ROM device writes and write-protect commands.

The second ROM access method (direct-attach ROM mode) attaches the ROM directly to the 660 Bridge using the PCI_AD lines. This mode is compatible with that used by the 650 Bridge, and is required when using the Intel SIO ISA bridge chip because the SIO does not support mapping of the ROM to the specified address space. The direct-attach mode also supports ROM device writes and write-protect commands.

The ROM mode is indicated to the 660 Bridge on the strapping pin configuration bits during power-on-reset (POR). See Section 10.4.

8.1 Remote ROM Mode

In a system that uses the remote ROM mode, the ROM device attaches to a PCI agent. When a CPU bus master reads from memory addresses mapped to ROM space, the bridge arbitrates for the PCI bus and then masters a memory read transaction on the PCI bus. The PCI agent claims the transaction and supplies the ROM device data. CPU writes to the ROM and ROM write-protection operations are also forwarded to the PCI agent.

As shown in Figure 8-1, the ROM access flows from the CPU to the bridge over the CPU bus, from the bridge to the PCI agent over the PCI bus, and from the PCI agent to the ROM device. The ROM device attaches to the PCI agent, not to the PCI_AD lines, so a PCI bus load is saved by the remote ROM method.

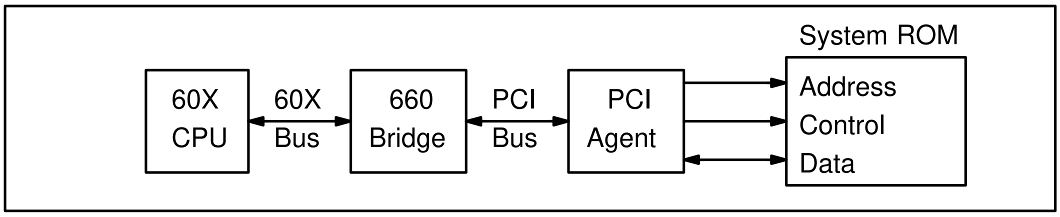


Figure 8-1. Remote ROM Connections

8.1.1 Remote ROM Reads

For remote ROM reads, the 660 Bridge arbitrates for the PCI bus, initiates eight single-byte PCI accesses, releases the PCI bus, and completes the CPU transfer. The eight single bytes of ROM data are assembled into a double-word in the 663 Buffer and passed to the CPU.

During and following reset, compliant PCI agents are logically disconnected from the PCI bus except for the ability to respond to configuration transactions. These agents have not yet been configured with necessary operational parameters. The IBM27–82376 PCI to ISA Bridge and other PCI agents capable of the remote ROM access protocol reset with the ability to respond to remote ROM accesses before being fully configured. The 60X CPU begins reading instructions at FFF0 0100h before it can configure the PCI devices.

The ROM read discussion assumes that the system is in big-endian mode. For the effects of little-endian mode operation on ROM reads, see Section 8.1.1.4.

8

8.1.1.1 Remote ROM Read Sequence

In response to a CPU bus read in the 4G – 2M to 4G address range, the bridge requests the PCI bus from the PCI arbiter. When the PCI bus is granted (or if the bus is already parked on the CPU), the bridge initiates a series of PCI memory-read transactions as shown in Table 8-1 for a CPU read from FFE0 0000h to FFFF FFFF. Note: The last column in Table 8-1 shows the effect of little-endian mode operation. See Section 8.1.1.4.

The address of the first transaction is the low-order byte of the double-word pointed to by the CPU address (see Section 8.1.1.2). The 660 Bridge expects the low-order byte of ROM data in the 8-byte double-word to be returned on PCI byte lane 0, PCI_AD[7:0]. The bridge then masters seven more PCI read transactions, each time receiving back one byte of ROM data and driving it onto the CPU data bus as shown in Table 8-1. Note that the byte enables are incrementing within each 4-byte word pointed to by the PCI address.

At the completion of the eighth PCI read, the bridge drives the assembled double-word onto the CPU data bus. The bridge then signals completion of the transfer to the CPU.

Remote ROM reads are not pipelined. The 660 Bridge does not assert AACK# to the CPU until the end of the remote ROM read sequence. The bridge asserts PCI_REQ# throughout the entire remote ROM read sequence.

Table 8-1. Remote ROM Read Sequence, CPU Address = FFFX XXX0

PCI Bus Read Memory Address PCI_AD[31:0]	Byte Enables PCI_C/BE[3:0]#	ROM Data	CPU_DATA [0:63]	Little-Endian CPU_DATA [0:63]
FFFX XXX0h	0001	a	axxx xxxx	xxxx xxxa
FFFX XXX0h	0010	b	baxx xxxx	xxxx xxab
FFFX XXX0h	0100	c	cbax xxxx	xxxx xabc
FFFX XXX0h	1000	d	dcba xxxx	xxxx abcd
FFFX XXX4h	0001	e	edcb axxx	xxxa bcde
FFFX XXX4h	0010	f	fedc baxx	xxab cdef
FFFX XXX4h	0100	g	gfed cbax	xabc defg
FFFX XXX4h	1000	h	hgfe dcba	abcd efgh

8.1.1.2 Address, Transfer Size, and Alignment

The initial PCI address generated during the remote ROM read sequence is formed by copying the high-order 29 bits of the CPU address, and forcing the three low order bits PCI_AD[2:0] to 000b. This generates a base address that is aligned on an 8-byte boundary. While reading the lower 4 bytes, the bridge indicates which byte it is requesting using the PCI byte enables C/BE[3:0]#. After the first four bytes of ROM data are read, the bridge increments the address on the PCI_AD lines by 4 before executing the second four PCI reads.

8

The CPU read address need not be aligned on an 8-byte boundary. A CPU read from any address (in ROM space) of any length that does not cross an 8–byte boundary within a double-word returns all eight bytes of that double-word data from the ROM. For example, the operations shown in Table 8-1 could have been caused by a CPU memory read to FFF0 0100h, FFF0 0101h, or FFF0 0105h.

Errors occurring during remote ROM reads are handled as usual for the error type (see Section 9). No special rules are in effect.

8.1.1.3 Burst Reads

The PowerPC 601 microprocessor begins instruction fetching in burst mode after a reset (the 603 CPU and 604 CPU do not come up with burst mode enabled). To support burst mode, the 660 Bridge allows a pseudo burst mode, which supplies the same eight bytes of data (from the ROM) to the CPU on each beat of a 4-beat burst.

A burst ROM read begins with the 660 Bridge executing a single-beat ROM read operation, which assembles eight bytes of ROM data into a double-word on the CPU data bus. For a burst ROM read, the 660 Bridge asserts TA# for four CPU_CLK cycles, with AACK# asserted on the fourth cycle. The same data remains asserted on the CPU data bus for all four of the data cycles.

For a single-beat read, the 660 Bridge asserts TA# and AACK# for one CPU_CLK cycle, and the 60X CPU completes the transfer.

8.1.1.4 Endian Mode Considerations

In little-endian mode, the address munging done by the CPU has no effect because PCI_AD[2:0] are forced to 000 during the address phase by the bridge at the beginning of the transaction. However, in little-endian mode the byte swapper is enabled, so the bytes of ROM data returned to the CPU are swapped as shown in the last column of Table 8-1.

8.1.2 Remote ROM Writes

While the 660 Bridge is configured for remote ROM operation, the 660 Bridge forwards all CPU to ROM write transfers to the PCI bus as memory writes. The PCI agent that is controlling the remote ROM acts as the PCI target during CPU to ROM write transfers, executes the write cycle to the ROM, and may provide ROM write-protection.

8.1.2.1 Write Sequence

A CPU bus master begins a remote ROM write transaction by initiating a one-byte, single-beat memory write transfer to CPU bus address range 4G – 2M to 4G (FF80 0000h to FFFF FFFFh).

The 660 Bridge decodes the CPU transfer, arbitrates for the PCI bus, and initiates a memory write PCI transaction to the same address in the 4G – 2M to 4G address range.

The PCI agent that is controlling the remote ROM (such as the IBM PCI to ISA Bridge), claims the transaction, manages the write cycle to the ROM device, and signals TRDY#.

The bridge then completes the PCI transaction, and signals AACK# and TA# to the CPU. Note that remote ROM writes are neither posted or pipelined.

8

8.1.2.2 Write Protection

Write protection can be provided by the PCI agent (such as the IBM PCI to ISA Bridge) that controls the ROM. In addition, some flash ROM devices can have the means to permanently lock out sectors by writing control sequences. The 660 Bridge also has a write lockout in the bridge chipset options 2 register (bit 0 of index BBh).

8.1.2.3 Address, Size, Alignment, and Endian Mode

In remote ROM mode, CPU memory writes from 4G – 2M to 4G cause the 660 Bridge to generate PCI bus memory write transactions to 4G – 2M to 4G. The bridge does not allow CPU masters to access the rest of the PCI memory space from 2G to 4G.

In remote ROM mode, PCI bus master memory write transactions from 4G – 2M to 4G are ignored by the bridge. However, the PCI agent that controls the ROM responds to these transactions. In contrast, in direct-attach ROM mode, the bridge forwards PCI bus master memory transactions from 2G to 4G (to populated memory locations) to system memory from 0 to 2G.

Remote ROM writes must be one-byte, single-beat transfers.

The endian mode of the system has no net effect on a ROM write because the transfer size is one byte. The address is munged by the CPU and unmunged by the 660 Bridge. The data comes out of the CPU on the byte lane associated with the munged address, and then is swapped by the Bridge to the byte lane associated with the unmunged address. Thus a ROM write in little-endian mode puts the data byte in the same ROM location as does the same ROM write in big-endian mode.

8.2 Direct-Attach ROM Mode

In a system that uses the direct-attach ROM mode, the ROM device attaches to the 660 Bridge by means of control lines and the PCI_AD[31:0] lines. When a CPU bus master reads from the ROM, the bridge masters a BCR transaction, during which it reads the ROM and returns the data to the CPU. CPU writes to the ROM and ROM write-protection operations are also forwarded to the ROM device.

Direct-attach ROM accesses flow from the CPU bus to the bridge. As shown in Figure 8-2, the data and address flow from the bridge to the ROM over the PCI_AD lines. ROM control flows from the bridge to the ROM over control lines that are not a part of the PCI bus.

Although connected to the PCI_AD lines, the direct-attach ROM is not a PCI agent. The ROM and the PCI agents do not interfere with each other because the ROM is under bridge control, and the bridge does not enable the ROM except during ROM cycles. The bridge accesses the ROM by means of BCR transactions. Other PCI devices cannot read or write the ROM because they cannot generate BCR transactions.

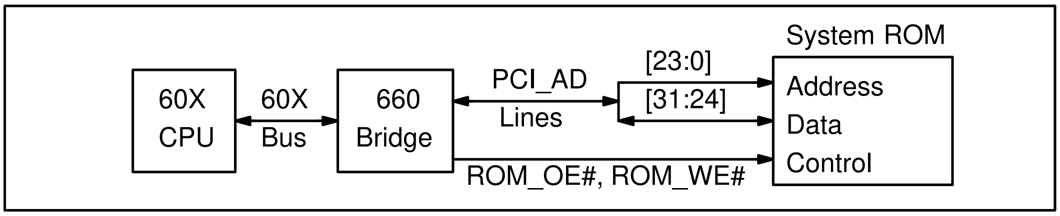


Figure 8-2. Direct-Attach ROM Connections

8.2.1 Direct-Attach ROM Reads

When a CPU bus master reads from memory addresses mapped to ROM space, the bridge arbitrates for the PCI bus and then masters a BCR transaction on the PCI bus. During this transaction, the bridge reads the ROM eight times, accumulates the data, and returns the double-word to the CPU. The bridge then completes the PCI transaction and releases the PCI bus.

The 664 Controller drives the address of the required byte over PCI_AD[23:0] to the ROM address pins. The ROM drives back the data on PCI_AD[31:24], where it is received by the 663 Buffer.

This ROM read discussion assumes that the system is in big-endian mode. For the effects of little-endian mode operation on ROM reads, see Section 8.2.1.3.

8.2.1.1 Direct-Attach ROM Read Timing Diagram

Figure 8-3 is a timing diagram of a CPU to ROM read transaction. This case assumes that the PCI bus is parked on the CPU, so that the bridge has a valid PCI bus grant when the CPU starts the 60X bus transfer.

Initially, the CPU drives the address and address attributes onto the 60X bus and asserts TS#. The bridge decodes the CPU transfer as a ROM read transaction. It is possible for TS# to be asserted across either a rising or falling edge of PCI_CLK. The bridge must only assert (and negate) PCI bus signals on the rising edge of PCI_CLK, so if TS# is asserted across a rising edge of PCI_CLK, the bridge waits one CPU_CLK to synchronize to the PCI bus.

TBD

Figure 8-3. Direct-Attach ROM Read Timing Diagram

The bridge initiates a BCR transaction by asserting PCI_FRAME# on the rising edge of PCI_CLK. Note that the bridge is driving PCI_AD[23:0] with the ROM address of byte 0 of the 8-byte aligned double-word. The bridge leaves PCI_AD[31:24] tri-stated, and asserts ROM_OE# to enable the ROM to drive the data onto these bits. On the next PCI_CLK, the bridge negates PCI_FRAME# and asserts PCI_IRDY#.

The ROM drives the requested data onto its data pins, across PCI_AD[31:24], and into the bridge. Seven PCI_CLKs after the bridge asserts PCI_FRAME#, it sends ROM_LOAD low. On the next clock, the bridge latches in the ROM data on PCI_AD[31:24], sends ROM_LOAD# high, and increments the ROM address on PCI_AD[23:0]. The byte from the ROM is latched into a byte shift register, which accumulates the bytes in an 8-byte doubleword. The contents of the shift register move through the bridge and onto the CPU data bus.

The ROM then drives the next data byte onto PCI_AD[31:24]. Seven PCI_CLKs after it negated ROM_LOAD for the previous byte, the bridge again negates ROM_LOAD, and also shifts the previous byte of ROM data to the next position. On the next PCI_CLK, the bridge sends ROM_LOAD high and increments the ROM address on PCI_AD[23:0]. This pattern is repeated until all eight bytes have been loaded into the shift register.

After the last byte has been latched into the bridge by the falling edge of ROM_LOAD, the bridge completes the PCI transaction by deasserting PCI_IRDY#. It also negates ROM_OE# to clear the PCI bus. After the last byte of data has had time to propagate through onto the CPU data bus, the bridge signals TA# to the CPU. Table 8-2 shows the data and address flow during the transaction.

On a single-beat transfer, the CPU asserts and negates AACK# concurrently with TA#. For a burst transfer, the bridge asserts TA# for four CPU_CLKs to return four identical doublewords to the CPU. This is the only difference between single-beat and burst ROM reads.

Also note that PCI_DEVSEL#, PCI_TRDY#, PCI_STOP#, and ROM_WE# are negated throughout the transaction.

8

Table 8-2. Direct-Attach ROM Read Data and Address Flow

ROM Data Byte	ROM Address PCI_AD[23:0]	ROM Data PCI_AD[31:24]	CPU_DATA[0:63] After Shift	CPU_DATA[0:63] After Shift (LE Mode)
Byte 0	XX XXX0h	a	axxx xxxx	xxxx xxxa
Byte 1	XX XXX1h	b	baxx xxxx	xxxx xxab
Byte 2	XX XXX2h	c	cbax xxxx	xxxx xabc
Byte 3	XX XXX3h	d	dcba xxxx	xxxx abcd
Byte 4	XX XXX4h	e	edcb axxx	xxxa bcde
Byte 5	XX XXX5h	f	fedc baxx	xxab cdef
Byte 6	XX XXX6h	g	gfed cbax	xabc defg
Byte 7	XX XXX7h	h	hgfe dcba	abcd efgh

8.2.1.2 Address, Transfer Size, and Alignment

During ROM reads, system ROM is linear-mapped to CPU memory space from 4G – 2M to 4G (FFE0 0000h to FFFF FFFFh). This address range is translated onto PCI_AD[23:0] as 0 to 2M (0000 0000h to 001F FFFFh). Since the 60X CPU begins fetching instructions at FFF0 0100h after a reset, the most convenient way to use a 512K device as system ROM with the 60X CPU is to use it from 4G – 512K to 4G. Connecting PCI_AD[18:0] to ROM_A[18:0] with no translation implements this. With this connection, the system ROM is aligned with 4G – 2M, but with alias addresses every 512K up to 4G. Other size devices can also be implemented this way.

The CPU read address need not be aligned on an 8-byte boundary. A CPU read from any ROM address of any length that does not cross an 8–byte boundary within a double-word returns all eight bytes of that double-word from the ROM. For example, the operations shown in Table 8-2 could have been caused by a CPU memory read to FF80 0100h, FF80 0101h, or FF80 0105h.

Other errors occurring during remote ROM reads are handled as usual for the error type (see Section 9). No special rules are in effect.

8.2.1.3 Direct-Attach Endian Mode Considerations

In little-endian mode, the address munging done by the CPU has no effect because PCI_AD[2:0] are forced to 000 during the address phase by the bridge at the beginning of the transaction. However, in little-endian mode the byte swapper is enabled, so the bytes of ROM data returned to the CPU are swapped as shown in the last column of Table 8-2.

8.2.2 Direct-Attach ROM Writes

The 660 Bridge decodes a CPU store word instruction to CPU address FFFF FFF0h as a ROM write cycle. (Note that the 660 Bridge treats any access from FFE0 0000h to FFFF FFFE, with CPU_A[31]=0, as an access to FFFF FFF0.) The three low-order bytes of the CPU data word are driven onto the ROM address lines, and the high-order byte is driven onto the ROM data lines. For example, a store word instruction with data = 0012 3456h writes 56h to ROM location 00 1234h. Only single-beat, four-byte write transfers (store word) are supported. A ROM write is considered to be a BCR operation. The ROM write BCR is detailed in Section 8.3.1.

8

The ROM write discussion assumes that the system is in big-endian mode. For the effects of little-endian mode operation on ROM reads, see Section 8.2.2.3.

In direct-attach mode, the ROM is attached to the 660 Bridge as shown in Figure 8-4.

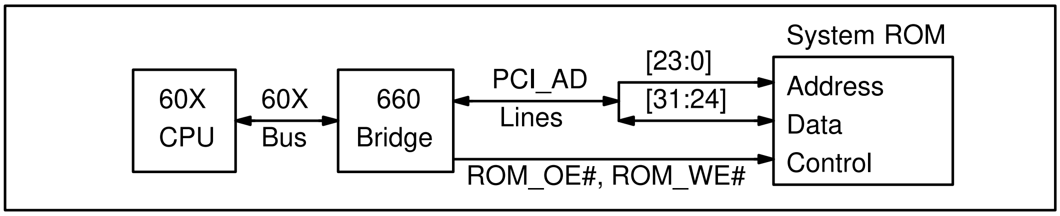


Figure 8-4. Direct-Attach ROM Connections

8.2.2.1 Direct-Attach ROM Write Timing Diagram

Figure 8-5 shows a timing diagram for a ROM write operation. This particular case assumes that the PCI bus is parked on the CPU. Initially, the CPU drives the address and address attributes onto the 60X bus and asserts TS#. The bridge decodes the CPU transfer as a ROM write transaction, a BCR transaction.

TBD

Figure 8-5. Direct-Attach ROM Write Timing Diagram

The bridge initiates a BCR transaction by asserting PCI_FRAME# on the rising edge of PCI_CLK. Note that the bridge is driving PCI_AD[23:0] with the ROM address and PCI_AD[31:24] with the ROM data. On the next PCI_CLK, the bridge negates PCI_FRAME# and asserts IRDY#. Seven PCI_CLKs after the bridge asserts PCI_FRAME#, it asserts ROM_WE# for CPU_CLKs.

The bridge completes the PCI transaction by deasserting PCI_IRDY#. The bridge signals TA# and AACK# to the CPU to signal transfer completion to the CPU. Also note that PCI_DEVSEL#, PCI_TRDY#, PCI_STOP#, and ROM_OE# are negated throughout the transaction.

8.2.2.2 Write Protection

Write protection for direct-attach ROM is provided through the ROM lockout BCR (see Section 8.3.2). ROM write-lockout operations are compatible with the 650 Bridge.

When a CPU bus master writes any data to memory address FFFF FFF1h, the bridge locks out all subsequent ROM writes until the bridge is reset. In addition, flash ROM devices can have the means to permanently lock out sectors by writing control sequences. Flash ROM specifications contain details. Note that the 660 Bridge treats any access from FFE0 0001 to FFFF FFFF, with CPU_A[31]=1, as an access to FFFF FFF1.

8.2.2.3 Data Flow In Little-Endian Mode

Figure 8-6 and Table 8-3 show the flow of CPU Data through the bridge to the ROM while the system is in little-endian mode. Note that the CPU Data bus is labeled in big-endian order, the PCI bus is labeled in little-endian order, and the bridge is labeled to match (and the bit significance within the bytes is maintained).

When the CPU executes a store word instruction to FFFF FFF0h, the contents of the source register appear on CPU_DATA[32:63]. CPU_ADDR[29] is 1 (after the CPU munges the address), so the 660 Bridge selects CPU data byte lanes 4 through 7 as the source of the data. The system is in little-endian mode, so the buffer swaps the data bytes. If the register data is AB012345h, then ABh is written to address 012345h of the ROM. Only single-beat, four-byte write transfers (store word) are supported.

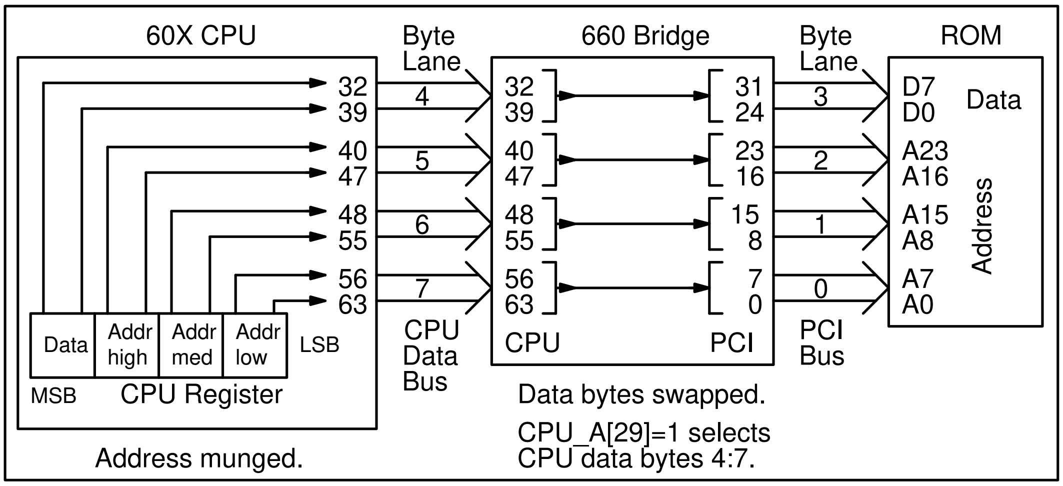


Figure 8-6. Direct-Attach ROM Data and Address Flow In Little Endian Mode

Table 8-3. Direct-Attach ROM Write Data Flow in Little-Endian Mode

CPU Register	CPU DATA[0:63]	Content	PCI_AD[31:0]	ROM Signal
MSB	32:39	ROM Data	31:24	D[7:0]
	40:47	ROM Address high byte	23:16	A[23:16]
	48:55	ROM Address mid byte	15:8	A[15:8]
LSB	56:63	ROM Address low byte	7:0	A[7:0]

8.2.2.4 Data Flow In Big-Endian Mode

Figure 8-7 and Table 8-4 show the flow of CPU Data through the bridge to the ROM while the system is in big-endian mode. Note that the CPU Data bus is labeled in big-endian order, the PCI bus is labeled in little-endian order, and the bridge is labeled to match (and the bit significance within the bytes is maintained).

When the CPU executes a store word instruction to FFFF FFF0h, the contents of the source register appear on CPU_DATA[0:31]. CPU_ADDR[29]=0, so the 660 Bridge selects CPU data byte lanes 0 through 3 as the source of the data. The system is in big-endian mode, so the buffer does not swap the data bytes. If the register data is 452301ABh, then ABh is written to address 012345h of the ROM. Only single-beat, four-byte write transfers (store word) are supported.

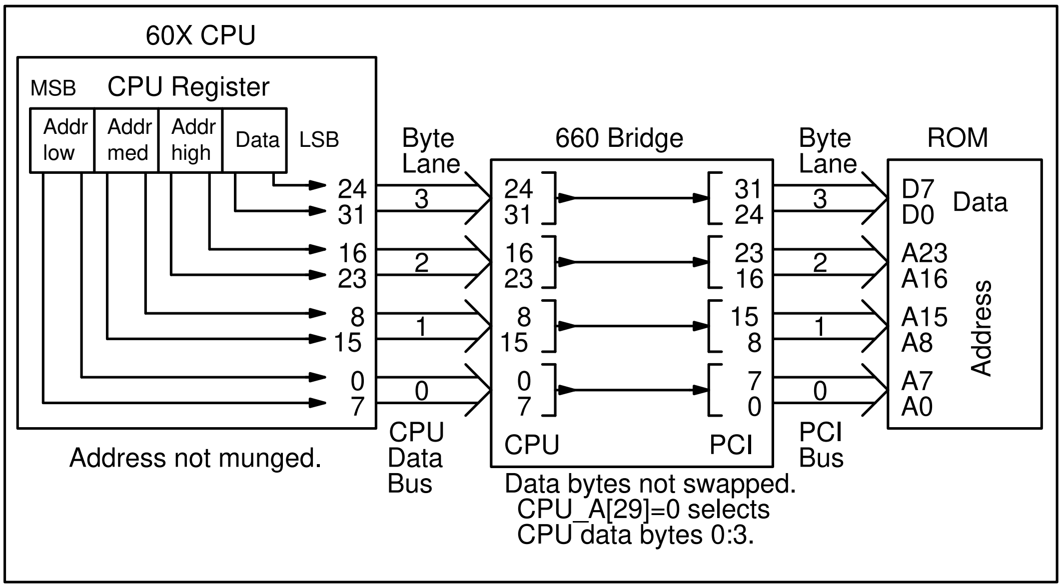


Figure 8-7. Direct-Attach ROM Data and Address Flow In Big Endian Mode

Table 8-4. Direct-Attach ROM Write Data Flow in Big-Endian Mode

CPU Register	CPU DATA[0:63]	Content	PCI_AD[31:0]	ROM Signal
MSB	0:7	ROM Address low byte	7:0	A[7:0]
	8:15	ROM Address mid byte	15:8	A[15:8]
	16:23	ROM Address high byte	23:16	A[23:16]
LSB	24:31	ROM Data	31:24	D[7:0]

8.3 ROM-Related Bridge Control Registers

The two BCRs most closely related to the ROM system are the ROM write BCR and the ROM lockout register. Writes to the ROM are accomplished through the ROM write BCR. Write-protection is provided by means of the ROM lockout BCR.

8.3.1 ROM Write Bridge Control Register

Direct Access FFFF FFF0h	Write Only	Reset NA

This 32-bit, write-only register is used to program the ROM in direct-attach ROM systems (see section 8.2.2). This register must be written by means of a 4-byte transfer. Bits are shown with little-endian labels.

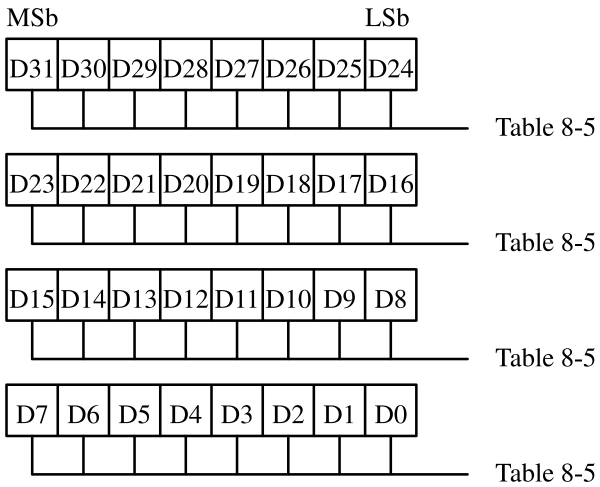


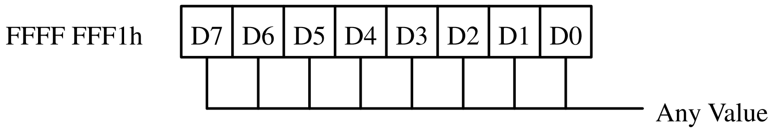
Table 8-5. Direct-Attach ROM Write BCR Contents

BCR Byte	Content in Little-Endian System	Content in Big-Endian System
MSB	ROM Data	ROM Address low byte
	ROM Address high byte	ROM Address mid byte
	ROM Address mid byte	ROM Address high byte
LSB	ROM Address low byte	ROM Data

8.3.2 ROM Lockout BCR (Direct-Attach ROM)

Direct Access FFFF FFF1h	Write Only	Reset NA

After it has been written once, this 8-bit, write-only register prevents ROM writes. This register is only used in direct-attach ROM systems.



Bits 7:0 Writing any value to the register prevents all future writes to a ROM that is connected directly to the 660 Bridge through the PCI_AD lines.

8.3.3 ROM Lockout Bit for Remote-Attach ROM

The ROM write-protect bit for remote ROM is in the bridge options 2 register (index BBh). While enabled (bit 0 is 1), writes to the remote ROM are forwarded to the PCI memory space. While disabled (bit 0 is 0), writes to the remote ROM are treated as no-ops and an error is signalled. After the bit is set to 0 (disabled), it cannot be set to 1 (enabled).

8.4 Programming the ROM Boot For 601 Burst Reads

To construct the bootstrap portion of the code that is required for use with the 601 CPU pseudo burst mode ROM reads described in Section 8.1.1.3, the first part of the system ROM can be coded as follows:

Instruction 1
Branch to instruction 2
No–op
No–op
No–op
No–op
No–op
No–op
Instruction 2
Branch to instruction 3
6 no–ops
Instruction 3
Branch to instruction 4
...

The six no-op instructions serve as filler for the unexecuted phases of the burst reads of the system ROM. The no-op codes are not transferred during the burst read, only the first two instructions (64 bits) are read and then passed four times to the 601 CPU during a startup burst read of system ROM.

When enough instructions have been executed, the bootstrap code can turn off the 601 cache, and the remaining ROM data can be read with single-beat reads.

Section 9
Error Handling

9.1 Introduction

The 660 Bridge supports the detection and reporting of several types of errors. The errors are reported to the CPU or the PCI and status information is saved in the 660 register set so that error type determination can be done by the CPU.

All errors are associated with a either a cycle on the CPU bus or a cycle on the PCI bus. (There is also one error condition that is not associated with any bus.) Errors detected on the memory bus are associated with a cycle on either the CPU bus (when the CPU accesses memory) or the PCI bus (when a PCI master accesses memory). Errors detected on the PCI bus when the 660 Bridge is the PCI master are associated with a CPU bus cycle; therefore, the only errors associated with a PCI bus cycle are errors that are detected while the 660 Bridge is a PCI target.

Errors associated with a CPU bus cycle are reported to the CPU by means of the MCP# or the TEA# signal. Errors associated with a PCI bus cycle are reported to the PCI by means of the PCI_PERR# or the PCI_SERR# signals.

Each error that can be detected has a mask associated with it. If the mask is set, then the detection of that error condition is disabled. There are also masks for the MCP#, TEA#, and PCI_SERR# signals that prevent reporting of any error by means of that signal (these masks do not affect the detection of the error).

9.2 Error Handling Protocols

The following protocols describe the error handling actions taken by the 660 Bridge. Once an error is detected and the appropriate status, address, and control information is saved, the detection of all subsequent errors is disabled until the error is reset.

9.2.1 CPU Bus Cycle Errors

CPU bus cycles that result in the following errors use the following protocol.

9.2.1.1 CPU Bus Cycle Types of Errors

- CPU bus unsupported transfer type
- CPU bus XATS# asserted
- CPU bus data bus parity error
- CPU bus write to locked flash
- CPU bus memory select error
- Memory parity error (during CPU access to memory)
- Memory single-bit ECC error trigger exceeded (during CPU access to memory)
- Memory multi-bit ECC error (during CPU access to memory)
- L2 cache parity error
- PCI data bus parity error (while PCI master)
- PCI bus target abort received (while PCI master)
- PCI bus master abort generated (while PCI master)

9.2.1.2 Error Handling Protocol

If the error is masked, do not detect the error.

If the error is detected, perform the following steps.

1. Set status bit indicating error type.
2. Set status bit indicating error during CPU cycle.
3. Save CPU address and control bus values.
4. Report error to the CPU. (Reported by means of TEA# if the CPU cycle is still active or by means of MCP# if the CPU cycle has ended.)

Note: PCI data bus parity errors also cause PCI_PERR# to be asserted.

9

Note: There is a status bit (PCI Status Register bit 15) that is set whenever any type of PCI bus parity error is detected. The setting of this status bit is not maskable.

9.2.2 Error Reporting to 601 CPU

The 601 CPU does not have an MCP# pin; therefore, in cases where MCP# would be asserted, the 660 Bridge does the following:

1. Asserts CPU_INT.
2. On the next PCI interrupt acknowledge from the CPU, no interrupt acknowledge is generated to the PCI. Instead, the CPU cycle is ended by asserting TEA# and returning a value of FFh.

The 660 Bridge must be set up to operate in this manner when the CPU is a 601. No MCP mode is enabled by bit 4 in the bridge chip set options 3 register (index D4h).

9.2.3 PCI Bus Cycle Errors

CPU bus cycles that result in the following errors use the following protocol.

9.2.3.1 PCI Bus Cycle Types of Errors

- PCI address bus parity error (while PCI target)
- PCI data bus parity error (while PCI target)
- PCI memory select error (while PCI target)
- Memory parity error (during PCI master access to memory)
- Memory signal–bit ECC error trigger exceeded (during PCI master access to memory)

- Memory multi-bit ECC error (during PCI master access to memory)

9.2.3.2 Error Handling Protocol

If the error is masked, do not detect the error.

If the error is detected, perform the following steps.

1. Set status bit indicating error type.
2. Set status bit indicating error during PCI cycle.
3. Save PCI address and control bus values.
4. Report error to the PCI. (If the error is a data bus parity error then report by means of PCI_PERR#. If the error is not a data bus parity error then report by means of PCI_SERR#. Also, if the PCI cycle is still active, then target abort the cycle.)

Note: Data bus parity errors can be enabled to report errors by means of PCI_SERR#. This method is only used if it is determined that PCI_PERR# is not supported by some (or all) of the PCI masters in the system.

9.2.4 Other Error

The only error condition not associated with a CPU or PCI bus cycle is a sampling of NMI active.

9.2.4.1 Error Handling Protocol

Report error to the CPU by means of MCP#.

Note: Logic external to the 660 Bridge typically provides mask and status bits for NMI.

9.3 Description of Error Types

The following sections describe the methods used to detect errors.

9

9.3.1 CPU Bus Unsupported Transfer Types and Sizes

This error is generated when the CPU generates a bus operation that is not supported by the 660 Bridge. This includes unsupported bus operation types and unsupported transfer sizes. An error is not generated if the cycle is claimed by another CPU device (CPU_BUS_CLAIM# asserted).

9.3.1.1 CPU Bus Unsupported Transfer Types

Table 9-1 shows the CPU bus operations that are not supported:

Table 9-1. Invalid CPU Bus Operations

TT[0:4]	Operation
1000x	Reserved
1010x	**ecowx**
1011x	Reserved
1110x	**eciwx**

9.3.1.2 CPU Bus Unsupported Transfer Sizes

The following transfer sizes are supported. Transfer sizes that are not in the following list are not supported.

- One-byte to eight-byte single-beat reads or writes to memory within a double-word boundary
- Burst reads or writes to memory (32 bytes, aligned to double-word)
- One-byte to four-byte single-beat reads or writes to the PCI bus that do not cross a word boundary
- Eight-byte single-beat writes to the PCI bus within a double word boundary
- All addresses not to memory or PCI with sizes of one to four bytes within a word boundary
- ROM reads support the same sizes as memory

9.3.1.3 Error Control Registers

Transfer type and size errors can be controlled by means of registers in the primary register set. The mask is at register C0h bit 0. If an error is detected, status bits at register C1h bits 1:0 are set to 10. Register C7h bit 4 is cleared to indicate an error on a CPU cycle. The CPU address is saved in register C8h. The CPU control is saved in register C3h and the CPU number is saved in register C7h bit 5. Transfer type and size errors are reset by writing a 1 to register C1h bit 0 or register C1h bit 1. Note that the primary register set uses the same mask and error reset bits for XATS# that it uses for unsupported transfer types.

Transfer type and size errors can also be controlled by means of registers in the 650-compatible register set. The mask cannot be controlled by means of this register set. If an error is detected, the status bit at 8000 0844h bit 0 is cleared. The address is saved at BFFF EFF0h. This error can be reset by reading BFFF EFF0h. Note that the 650-compatible register set does not differentiate between XATS# errors and unsupported transfer type errors.

9.3.2 CPU Bus XATS# Asserted

This error is generated when the CPU asserts the XATS# signal for a PIO instruction.

The XATS# error can be controlled by means of registers in the primary register set. The mask is at register C0h bit 0. If an error is detected, the status bits at register C1h bits 1:0 are set to 01. Register C7h bit 4 is cleared to indicate an error on a CPU cycle. The CPU address is saved in register C8h. The CPU control is saved in register C3h and the CPU number is saved in register C7h bit 5. This error can be reset by writing a 1 to register C1h bit 0 or register C1h bit 1. Note that the primary register set uses the same mask and error reset bits for XATS# and for unsupported transfer types.

This error can also be controlled by means of registers in the 650-compatible register set. The mask cannot be controlled by means of this register set. If an error is detected, the status bit at 8000 0844h bit 0 is cleared. The address is saved at BFFF EFF0h. This error can be reset by reading BFFF EFF0h. Note that the 650-compatible register set does not differentiate between XATS# errors and unsupported transfer type errors.

9.3.3 CPU Data Bus Parity Error

This error is generated when a parity error on the CPU data bus is detected during a transfer between the CPU and the 660 Bridge. The full CPU data bus is always checked for parity regardless of which bytes lanes actually carry valid data. The parity is odd, which means that an odd number of bits, including the parity bit, are driven high. The 660 Bridge directly checks the parity during CPU write cycles. The 660 Bridge determines parity errors by sampling the DPE# signal from the CPU during CPU read cycles.

CPU_DPAR[0] indicates the parity for CPU_DATA[0:7]. CPU_DPAR[1] indicates the parity for CPU_DATA[8:15] and so on.

This error can be controlled by means of registers in the primary register set. The mask is at register C4h bit 2. If an error is detected, the status bit at register C5h bit 2 is set. Register C7h bit 4 is cleared to indicate an error on a CPU cycle. The CPU address is saved in register C8h, the CPU control is saved in register C3h, and the CPU number is saved in register C7h bit 5. This error can be reset by writing a 1 to register C5h bit 2.

This error cannot be controlled by means of the 650-compatible register set.

9.3.4 CPU Bus Write to Locked Flash

This error is generated when the CPU attempts to write to flash memory when write to flash ROM has been disabled (locked out). If the flash ROM is directly attached to the 660 Bridge (see configuration strapping), CPU writes to FFFF FFF0h are detected as an error if writing has been locked out by means of 660 compatible register FFFF FFF1h (see note in Sections 8.2.2 and 8.2.2.2). If the flash is remotely attached then CPU writes to the 4G – 2M to 4G address space are detected as an error if writing has been locked out by means of register BBh bit 0.

This error can be controlled by means of registers in the primary register set. The mask is at register C4h bit 0. If an error is detected, the status bit at register C5h bit 0 is set. Register C7h bit 4 is cleared to indicate error on a CPU cycle. The CPU address is saved in register C8h. The CPU control is saved in register C3h and the CPU number is saved in register C7h bit 5. This error can be reset by writing a 1 to register C5h bit 0.

9

This error cannot be controlled by means of the 650-compatible register set.

9.3.5 Memory Select Error

This error is generated if a device addresses the system memory space (CPU addresses from 0 to 2G and PCI addresses from 2G to 4G) when memory is not present at that address. The 660 Bridge only claims PCI accesses by asserting PCI_DEVSEL# when the access is to an address where memory is present.

The 660 Bridge disconnects PCI burst cycles at 1M boundaries. This ensures that a PCI master cannot begin a transfer at an address where memory is present and then burst (incrementing the address) to an address where memory is not present; therefore, the memory select error is never generated on PCI accesses to system memory.

The memory select error can be controlled by means of registers in the primary register set. The mask is at register C0h bit 5. If an error is detected, the status bit at register C1h bit 5 is set. Register C7h bit 4 is cleared to indicate an error on a CPU cycle. The CPU address is saved in register C8h. The CPU control is saved in register C3h and the CPU number is saved in register C7h bit 5. This error can be reset by writing a 1 to register C1h bit 5.

This error cannot be controlled by means of the 650-compatible register set.

9.3.6 System Memory Parity Error

When memory is being operated in parity mode, this error is generated if a parity error is detected during a read from system memory. Memory parity is odd, which means that an odd number of bits including the parity bit are driven high.

MEM_CHECK[0] indicates the parity for MEM_DATA[7:0]. MEM_CHECK[1] indicates the parity for MEM_DATA[15:8] and so on.

The system memory parity error can be controlled by means of registers in the primary register set. The mask is at register C0h bit 2. If an error is detected, the status bit at register C1h bit 2 is set.

If the parity error occurred while the CPU was accessing memory, then register C7h bit 4 is cleared to indicate the error occurred during a CPU cycle. The CPU address is saved in register C8h. The CPU control is saved in register C3h and the CPU number is saved in register C7h bit 5.

If the parity error occurred while the PCI was accessing memory, then register C7h bit 4 is set to indicate the error occurred during a PCI cycle. The PCI address is saved in register C8h. The PCI control is saved in register C7h.

This error can be reset by writing a 1 to register C1h bit 2. Note that register locations listed above are used to indicate single-bit ECC errors if the memory is being operated in ECC mode.

This error can also be controlled by means of the register in the 650-compatible register set. The mask cannot be controlled by means of this register set. If an error is detected, the status bit at 8000 0840h bit 0 is cleared. The address is saved at BFFF EFF0h. This error can be reset by reading BFFF EFF0h.

9

9.3.7 System Memory Single-Bit ECC Error

When memory is being operated in ECC mode, single-bit errors are detected and corrected. Since single-bit errors are corrected, generally no error reporting is necessary. But when a single-bit error is detected, the single–bit error counter register (register B8h) is incremented and the system memory address is saved in the single-bit ECC error address register (register CCh). If the count in the single-bit error counter register exceeds the value in the single-bit error trigger level register (B9h), the 660 Bridge generates a single–bit ECC trigger exceeded error.

The trigger exceeded error can be controlled by means of registers in the primary register set. The mask is at register C0h bit 2. If an error is detected, the status bit at register C1h bit 2 is set.

If the error occurs while the CPU is accessing memory, register C7h bit 4 is cleared to indicate the error occurred during a CPU cycle. The CPU address is saved in register C8h. The CPU control is saved in register C3h and the CPU number is save in register C7h bit 5.

If the error occurs while the PCI is accessing memory, register C7h bit 4 is set to indicate the error occurred during a PCI cycle. The PCI address is saved in register C8h. The PCI control is saved in register C7h.

This error can be reset by writing a 1 to register C1h bit 2. Note that register locations listed above are used to indicate memory parity errors if the memory is being operated in parity mode.

This error cannot be controlled by means of the 650-compatible register set.

9.3.8 System Memory Multi-Bit ECC Error

When memory is being operated in ECC mode, this error is generated if a multi-bit ECC error (uncorrectable) is detected during a read from system memory.

The multi-bit ECC error can be controlled by means of registers in the primary register set. The mask is at register C0h bit 3. If an error is detected, the status bit at register C1h bit 3 is set.

If the error occurs while the CPU is accessing memory, then register C7h bit 4 is cleared to indicate the error occurred during a CPU cycle. The CPU bus address is saved in register C8h. The CPU control is saved in register C3h, and the CPU number is saved in bit 7 of register C7h.

If the error occurs while the PCI is accessing memory, then register C7h bit 4 is set to indicate the error occurred during a PCI cycle. The PCI address is saved in register C8h. The PCI control is saved in register C7h.

This error can be reset by writing a 1 to register C1h bit 3.

This error cannot be controlled by means of the 650-compatible register set.

9.3.9 L2 Cache Parity Error

This error is generated when a parity error is detected during a CPU read from the L2 cache. The parity is checked by the CPU which drives DPE# to the 660 Bridge. If parity is not supported on the L2 cache, this error should be masked.

This error can be controlled by means of registers in the primary register set. The mask is at register C4h bit 3. If an error is detected, the status bit at register C5h bit 3 is set. Register C7h bit 4 is cleared to indicate error on a CPU cycle. The CPU address is saved in register C8h. The CPU control is saved in register C3h and the CPU number is saved in register C7h bit 5. This error can be reset by writing a 1 to register C5h bit 3.

This error can also be controlled by means of a register in the 650-compatible register set. The mask cannot be controlled by means of this register set. If an error is detected, the status bits at 8000 0842h bit 0 and 8000 0843h bit 0 is cleared. The address is not saved in a 650-compatible register (register BFFF EFF0h is undefined). This error can be reset by reading 8000 0843h.

9.3.10 PCI Data Bus Parity Error While PCI Master

This error is generated when a data bus parity error is detected on the PCI bus during a cycle which is mastered by the 660 Bridge for CPU access to the PCI bus. The 660 Bridge checks parity during read cycles and samples PCI_PERR# during write cycles. The bridge asserts PCI_PERR# if a parity error is detected on a read cycle. The PCI bus uses even parity, which means that an even number of bits including the parity bit are driven high.

This error can be controlled by means of registers in the primary register set. The mask is at register 04h bit 6. If an error is detected, the status bit at register 06h bit 8 is set. Register C7h bit 4 is cleared to indicate an error on a CPU cycle. The CPU address is saved in register C8h. The CPU control is saved in register C3h and the CPU number is saved in register C7h bit 5. This error can be reset by writing a 1 to register 06h bit 8.

When this error is detected, the status bit at register 06h bit 15h is set, regardless of the state of the mask at register 04h bit 6. The status bit at register 06h bit 15 is set by all types of PCI bus parity errors. This bit is cleared by writing a 1 to register 06h bit 15.

This error cannot be controlled by means of the 650-compatible register set.

9.3.11 PCI Target Abort Received While PCI Master

This error is generated when a target abort is received on the PCI bus during a cycle which is mastered by the 660 Bridge for CPU access to the PCI bus.

This error can be controlled by means of registers in the primary register set. The mask is at register C0h bit 7. If an error is detected, the status bit at register 06h bit 12 is set. Register C7h bit 4 is cleared to indicate an error on a CPU cycle. The CPU address is saved in register C8h. The CPU control is saved in register C3h and the CPU number is save in register C7h bit 5. This error can be reset by writing a 1 to register 06h bit 12.

This error cannot be controlled by means of the 650-compatible register set.

9.3.12 PCI Master Abort Detected While PCI Master

This error is generated when a master abort is detected on the PCI bus during a cycle which is mastered by the 660 Bridge for CPU access to the PCI bus. Master aborts occur when no target claims a PCI memory or I/O cycle—PCI_DEVSEL# is never asserted.

Note that some operating systems intentionally access unused addresses to determine what devices are located on the PCI bus. These operating systems do not expect an error to be generated by these accesses. When using such an operating system it is necessary to leave this error masked.

9

This error can be controlled by means of registers in the primary register set. The mask is at register C4h bit 4. If an error is detected, the status bit at register 06h bit 13 is set. Register C7h bit 4 is cleared to indicate an error on a CPU cycle. The CPU address is saved in register C8h. The CPU control is saved in register C3h and the CPU number is saved in register C7h bit 5. This error can be reset by writing a 1 to register 06h bit 13.

This error cannot be controlled by means of the 650-compatible register set.

9.3.13 PCI Address Bus Parity Error While PCI Target

This error is generated when a parity error is detected during the address phase of a PCI access where the 660 Bridge is the PCI target of a PCI access to system memory.

This error can be controlled by means of registers in the primary register set. The mask is at register 04h bit 6. This error does not have an explicit status bit to indicate its occurrence. However, the following status bits are set:

1. Register 06h bit 14 is set to indicate PCI_SERR# has been asserted if the mask at 04h bit 8 is disabled (cleared). This bit is cleared by writing a 1 to register 06h bit 14.
2. Register 06h bit 11 is set to indicate signalled target abort if the cycle was target aborted. This bit is cleared by writing a 1 to register 06h bit 11.
3. Register 06h bit 15 is set to indicate a PCI bus parity error regardless of the state of the mask at register 04h bit 6. Note that this bit is set by all types of PCI bus parity errors. This bit is cleared by writing a 1 to register 06h bit 15.

Register C7h bit 4 is set to indicate an error on a PCI cycle. The PCI address is saved in register C8h. The PCI control is saved in register C7h.

This error cannot be controlled by means of the 650-compatible register set.

9.3.14 PCI Data Bus Parity Error While PCI Target

This error is generated when a data bus parity error is detected on the PCI bus during a cycle for which the 660 Bridge is the PCI target of a write cycle to system memory. The PCI bus uses even parity, which means that an even number of bits including the parity bit are driven high.

This error can be controlled by means of the registers in the primary register set. The mask is at register 04h bit 6. This error does not have an explicit status bit to indicate its occurrence. However, the following status bits are set:

1. Register 06h bit 11 is set to indicate signalled target abort if the cycle was target aborted. This bit is cleared by writing a 1 to register 06h bit 11.
2. Register 06h bit 15 is set to indicate a PCI bus parity error regardless of the state of the mask at register 04h bit 6. Note that this bit is set by all types of PCI bus parity errors. This bit is cleared by writing a 1 to register 06h bit 15.
3. Register 06h bit 14, which indicates PCI_SERR#, is set if the mask at register C0h bit 6 is disabled (cleared). Note that the mask at register C0h bit 6 allows the 660 Bridge to signal PCI_SERR# in addition to PCI_PERR# for this error. This bit is cleared by writing a 1 to register 06h bit 14.

Register C7h bit 4 is set to indicate an error on a PCI cycle. The PCI address is saved in register C8h. The PCI control is saved in register C7h.

This error cannot be controlled by means of the 650-compatible register set.

9

9.3.15 NMI Asserted

This error is generated when the NMI input is sampled asserted by the 660 Bridge. External logic (such as the Intel SIO) can assert this signal for any type of catastrophic error it detects. It should also assert this signal if it detects PCI_SERR# asserted. The external logic provides mask and status information. This Intel SIO provides the mask at Port 70h and the status at Port 61h in PCI I/O space.

Note: NMI is detected as a level-sensitive error; therefore, an error is generated if NMI is active when error detection is re-enabled by clearing a previous error.

Section 10
Additional Material

10.1 Resets

TBD

10.2 Endian Mode Switching

TBD

10.3 Power Management Support

The 660 Bridge supports a low-power mode. This mode allows the CPU_CLK to the 664 Controller and the 663 Buffer to be stopped while maintaining system DRAM refresh. The STOP_CLK_EN# input pin in the 664 Controller supports this function. (This pin is deasserted when low-power mode is not desired.)

If the 60X CPU bus and PCI bus are both idle, the 660 Bridge can be put into low power mode by asserting the STOP_CLK_EN# pin. Two CPU_CLKs after the assertion of STOP_CLK_EN#, the CPU_CLK can be stopped to both the 664 Controller and the 663 Buffer. Also, the PCI_CLK frequency can be reduced to as low as 1MHz anytime after the assertion of STOP_CLK_EN#. Entering the low power mode can be masked by means of the bridge chip set options 2 register (index BBh).

10

Assertion of the STOP_CLK_EN# pin tells the 664 Controller to begin generating refresh request based on the suspend refresh rate register. This register is programmed with the divisor for generating refresh request from PCI_CLK in suspend mode. The value of this register is programmed based on the PCI_CLK frequency (which can be reduced in suspend mode) and the desired refresh rate. If all the installed memory supports extended (slow) refresh, the suspend mode refresh rate can be slower than the normal mode refresh rate (15us).

STOP_CLK_EN# can be deasserted before, after, or coincident with the restart of CPU_CLK. PCI_CLK must return to its normal frequency before or coincident with the deassertion of STOP_CLK_EN#. While stopping and starting clocks, the minimum pulse width requirements for the CPU_CLK and PCI_CLK must be met (no glitches).

10.4 Configuration Strapping

There are two strapping options for 660 Bridge system configuration information required before the processor can execute (and which, therefore, cannot be programmed into the 660

Bridge). Configuration strapping is accomplished by attaching a pullup or pulldown resister to the specified 664 Controller output pin. During reset, the 664 Controller tri-states these outputs allowing them to assume the level to which they are strapped. When RESET# is deasserted, the 664 Controller reads in the value from these pins.

Table10-1 shows the strapping options and their associated pins. Pullup resistors should be 10K ohms to 20K ohms. Pull down resistors should be 500 ohms to 2K ohms.

Table10-1. Configuration Strapping Options

Function	Pull Up/Down	Pin
Location of ROM	Up = Remote Down = Direct	CRS_C2PWXS
603 in 1-to-1 mode	Down = 601, 604, or 603 not in 1:1 mode. Up = 603 in 1:1 mode.	MWS_P2MRXS

10

Appendix A
Timing Characteristics

A.1 Timing Conventions

Unless otherwise noted, all timing information is given for 660 Bridge operation within the envelope defined by the Recommended Operating Conditions.

Unless otherwise noted, all specifications in this section apply equally to the 663 Buffer and the 664 Controller.

A.1.1 Board Delays

Unless otherwise indicated, all timing specifications refer to events at the pins of the chip under discussion. In systems operating at speeds typical of the 60X family, propagation delays from point to point on a circuit board can be significant. The timing diagrams make no assumptions about board delays. No board or system propagation delays have been included in the timing diagrams or in the timing charts. Allow for delays between components while constructing timing diagrams for the design of an actual system.

A.1.2 Terms and Definitions

Signal range names used without range indicators refer to the entire group of signals. For example, CPU_DATA refers to the 653 signals CPU_DATA[0:63]. Ranges are expressed as [most-significant bit : least-significant bit].

A

Some sets of signals are referred to in a group in the timing diagrams. For example, CPU_ADDR generally refers to 60X address and address transfer attribute signals. Particular signals in the group may be shown separately for emphasis (TBST#, for example).

Valid refers to a voltage level above V_H(min) or below V_L(max). Valid does not imply logically true or false, asserted or negated.

A.1.3 Signal Switching Levels for Timing Analysis

Figure A-1 shows typical timing analysis signal switching levels, where V_H and V_L are the valid logic levels used for all input and output signals except CPU_CLK. Unless otherwise indicated, all input and output signal (not clock) switching specifications refer to the point in time at which the signal crosses one of these levels. These levels are used for both inputs and outputs for timing analysis only, and do not imply anything about the DC characteristics of the device.

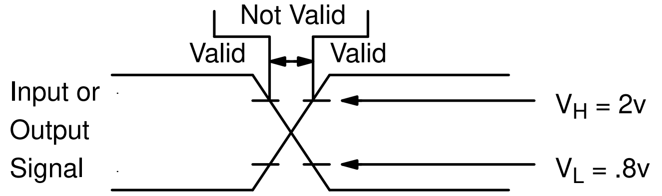


Figure A-1. Switching Levels

A.1.4 Input Setup Time

Input setup time is the amount of time that an input signal is required to be stable at a valid logic level immediately prior to an event. Input setup time (T_{IS} in Figure A-2) from a signal to the clock is measured from the point in time at which the input becomes valid to the the point in time at which the clock rising edge crosses the VM level. Input setup time from a signal to an input strobe is measured from the point in time at which the input becomes valid to the the point in time at which the strobe becomes active (its active edge crosses the valid logic level in the active going direction).

A.1.5 Input Hold Time

Input hold time is the amount of time that an input signal is required to remain stable at a valid logic level immediately following an event. Input hold time (T_{IH} in Figure A-2) from the clock to an input signal is measured from the point in time at which the clock rising edge crosses the VM level to the point in time at which the input goes invalid (crosses the valid logic level in the invalid going direction). Input hold time from an input strobe to an input signal is measured from the point in time at which the strobe becomes active (its active edge crosses the valid logic level in the active going direction) to the point in time at which the input goes invalid.

A.1.6 Output Hold Time

Output hold time is the amount of time that an output signal remains stable at a valid logic level immediately following an event which may cause the output to change state. Output hold time (T_{OH} in Figure A-2) from the clock is measured from the point in time at which the rising edge of the clock crosses the VM level to the point in time at which the output signal becomes invalid (crosses the valid logic level in the invalid going direction). Output hold time from an input strobe is measured from the point in time at which the strobe becomes active (its active edge crosses the valid logic level in the active going direction) to the point in time at which the output signal becomes invalid.

A

A.1.7 Output Delay Time

Output delay time is the amount of time required for an output signal to change to a stable valid state following an event. Output delay time (T_{OD} in Figure A-2) from the clock is measured from the point in time at which the rising edge of the clock crosses the VM level to the point in time at which the output signal becomes valid (crosses the valid logic level in the valid going direction). Output valid delay time from an input strobe is measured from the point in time at which the strobe becomes active (its active edge crosses the valid logic level in the active going direction) to the point in time at which the output signal becomes valid.

A.1.8 Output Enable Time

Output enable time is the amount of time required for an output signal driver to drive a tristated line to a valid logic level. This time does not take into account the effects of possible pullup resistors connected to the net. Such resistors may improve or degrade the actual time, depending on the situation. Output enable time from the clock is measured from the point in time at which the rising edge of the clock crosses the VM level to the point in time at which the output signal becomes valid (crosses the valid logic level in the valid going direction). There are no asynchronous output enables in the 660 Bridge.

A.1.9 Output Tristate Hold Time

Output tri-state hold time is the amount of time that an output signal remains driven to a valid logic level immediately following an event which may cause the output to tri-state (go to a high impedance state). Output tristate hold time (T_{3SH} in Figure A-2) from the clock is measured from the point in time at which the rising edge of the clock crosses the VM level to the point in time at which the output signal is no longer guaranteed to be actively driven to a valid logic level. Output hold time from an input strobe is measured from the point in time at which the strobe becomes active (its active edge crosses the valid logic level in the active going direction) to the point in time at which the output signal is no longer guaranteed to be actively driven to a valid logic level. Note that this specification deals with the time that the output driver remains active following an event which may turn it off, and is computed from the minimum output tristate delay time. The actual output signal may remain valid for some time after this, depending on other conditions.

A.1.10 Output Tristate Delay Time

Output tristate delay time is the amount of time required for an output signal driver to turn off (go to a high impedance state) following an event. Output tristate delay time (T_{3SD} in Figure A-2) from the clock is measured from the point in time at which the rising edge of the clock crosses the VM level to the point in time at which the output signal driver turns off (is no longer driving the output). Output valid delay time from an input strobe is measured from the point in time at which the strobe becomes active (its active edge crosses the valid logic level in the active going direction) to the point in time at which the output signal driver turns off. Note that this specification deals with the time that it takes the output driver to stop driving the output signal line following an event which may turn it off. The actual output signal may remain valid for some time after this, depending on other conditions.

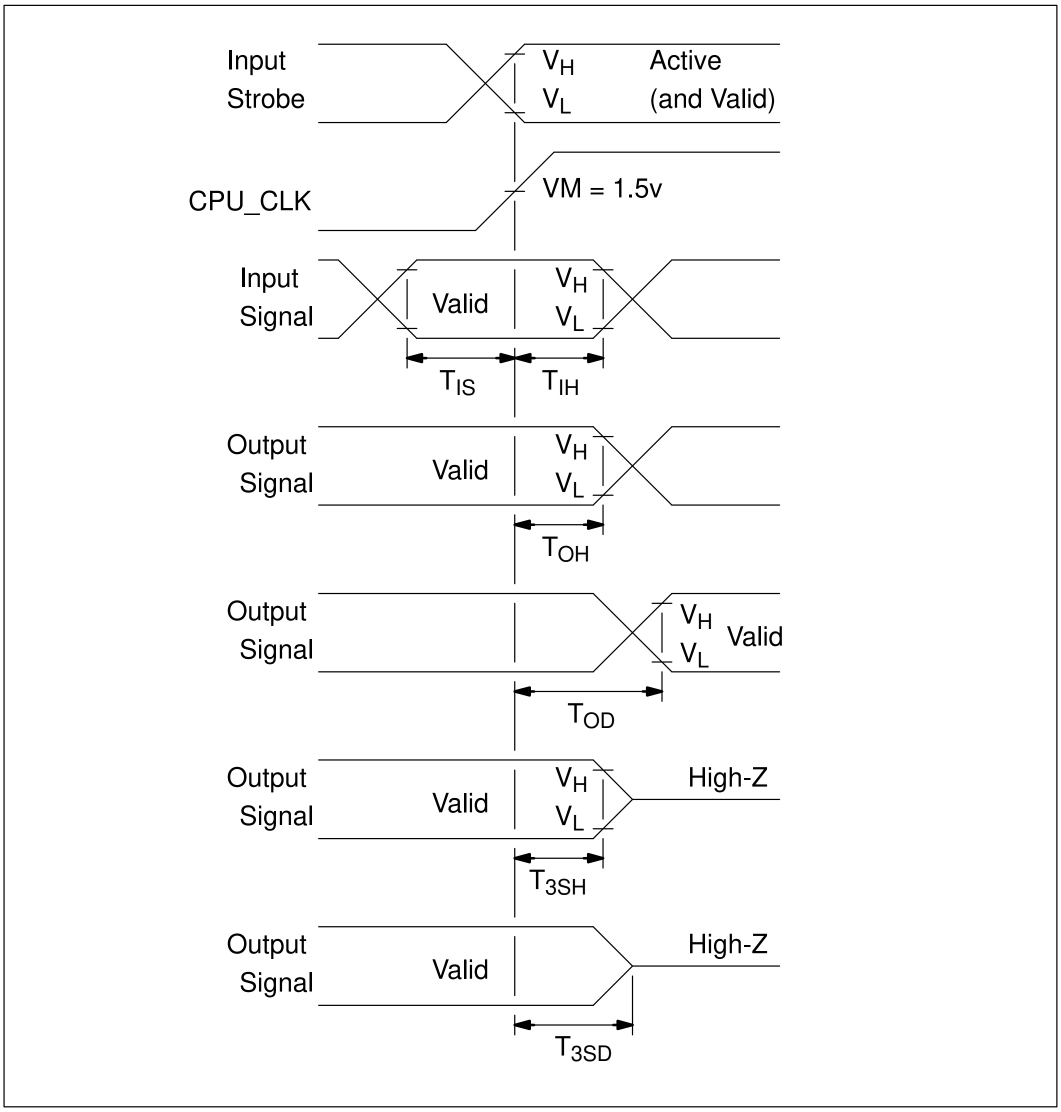


Figure A-2. Signal Timing Conventions

A.2 Clock Considerations

To maintain synchronization between the 660 Bridge and the CPU, certain constraints are placed on the CPU_CLK signal. Unless otherwise noted, all references to CPU_CLK timing refer to the point in time at which the CPU_CLK crosses the VM level, 1.5v.

In general, both the 663 Buffer and the 664 Controller are synchronous machines. Inputs are sampled on the rising edge of CPU_CLK, and outputs are updated on the rising edge of CPU_CLK.

A.2.1 660 Bridge CPU_CLK Skew to the Processor SYSCLK

As shown in Figure A-3, the allowed skew between the 664 Controller CPU_CLK and the processor SYSCLK is +/– 1ns. The allowed skew between the 663 Buffer CPU_CLK and the processor SYSCLK is +/– 1ns.

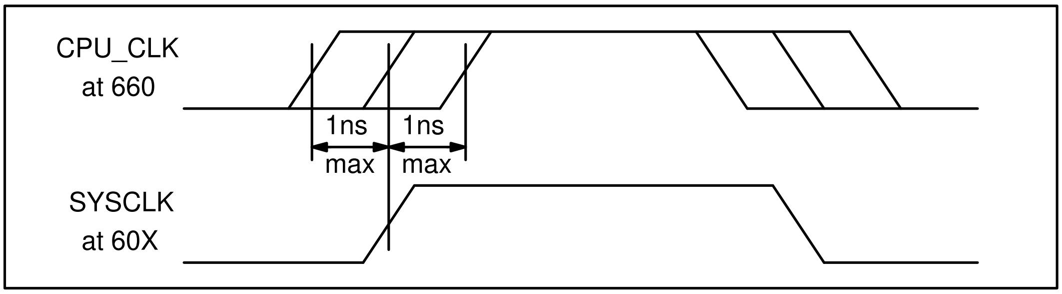


Figure A-3. CPU_CLK to SYSCLK Skew

A.2.2 663 Buffer CPU_CLK Skew to 664 Controller CPU_CLK

The allowed skew between the CPU_CLK at the input to the 664 and the CPU_CLK at the input to the 663 Buffer is also +/– 1ns. Thus the skew between any two of the three devices (663, 664, and 60x) is a maximum of +/– 1ns.

A

A.2.3 CPU_CLK Duty Cycle

The CPU_CLK is shown (at the CPU_CLK pin of the 664 Controller) in Figure A-4, where T_{CH} is the time that CPU_CLK is high, and T_{CL} is the time that CPU_CLK is low. The allowed duty cycle of CPU_CLK is from 35% to 65%. If the period of CPU_CLK is 15ns, then T_{CH} may range from 5.25ns to 9.75ns.

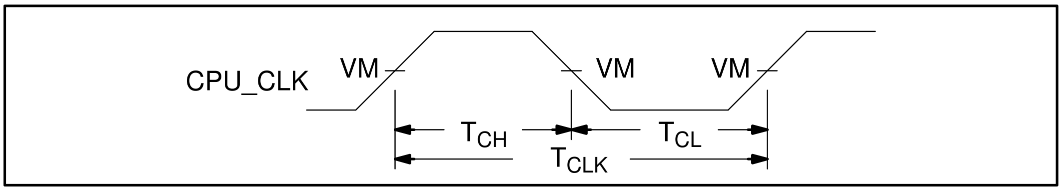


Figure A-4. CPU_CLK Duty Cycle

A.2.4 CPU_CLK to PCI_CLK Skew

The allowed skew of the PCI_CLK at any point in the system to the CPU_CLK at the 660 Bridge is +/– 2ns, as shown in Figure A-5.

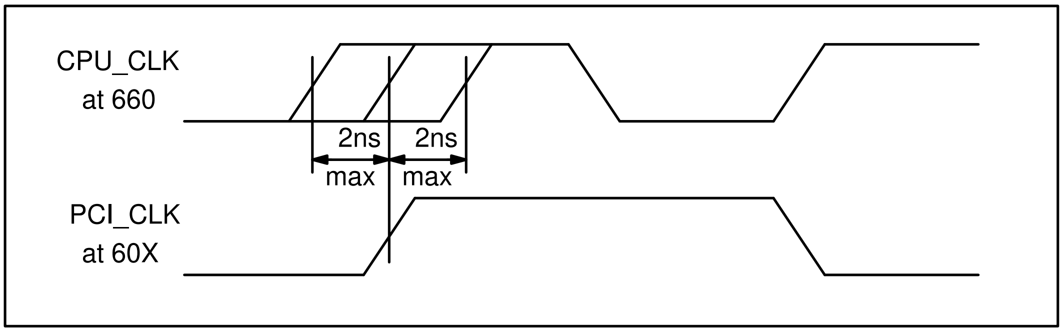


Figure A-5. CPU_CLK to PCI_CLK Skew

A.3 Asynchronous Paths

The 660 Bridge is in general a synchronous device. There are however some asynchronous paths which are only active to speed up the assertion of particulate signals. The paths speculatively assert certain signals. Once the clock edge latches in the correct input conditions and the synchronous logic verifies the speculative assertion, the asynchronous path is deactivated and no longer has any effect on the Bridge (until next time).

A.4 Power-On Considerations

The 660 Bridge is designed to impose no additional power-on-reset or power supply behavior constraints on a system that contains a 60X CPU and a PCI bus. The 660 works properly in a system designed to correctly support the 60X CPU and the PCI bus.

The 660 Bridge requires RESET# to be asserted at power-on for a minimum of 1us past power-good, and for a minimum of 10 CPU_CLK cycles past the point in time at which CPU_CLK is stable and within specification. The inputs to the Bridge are not required to be in any special state during the reset period, but the Bridge will start to respond to control inputs immediately following the deassertion of RESET#. This design fully supports a properly functioning 60X CPU, L2 cache, and PCI agents.

A

A.5 Detailed Timing Diagrams

The following timing diagrams in Figure A-6 through Figure A-22 illustrate some fundamental operations of the 660 Bridge.

In CPU to memory read burst-mode timing diagrams where asynchronous SRAMs are used with the internal L2 controller, an additional clock cycle must be added to the fourth beat of any CPU to memory read burst that causes a cache miss. For example after a read:

> A pipelined page hit, cache miss, burst read with EDO DRAM requires –3–3–3–3 CPU clocks when the L2 uses burst SRAMs and –3–3–3–4 CPU clocks when the L2 uses asynchronous SRAMs. The extra beat is caused by delaying the final TA# by one CPU_CLK to allow the asynchronous SRAM sufficient data hold time for the fourth beat.

A.5.1 CPU to Memory Write (Page DRAM) From Bus Idle

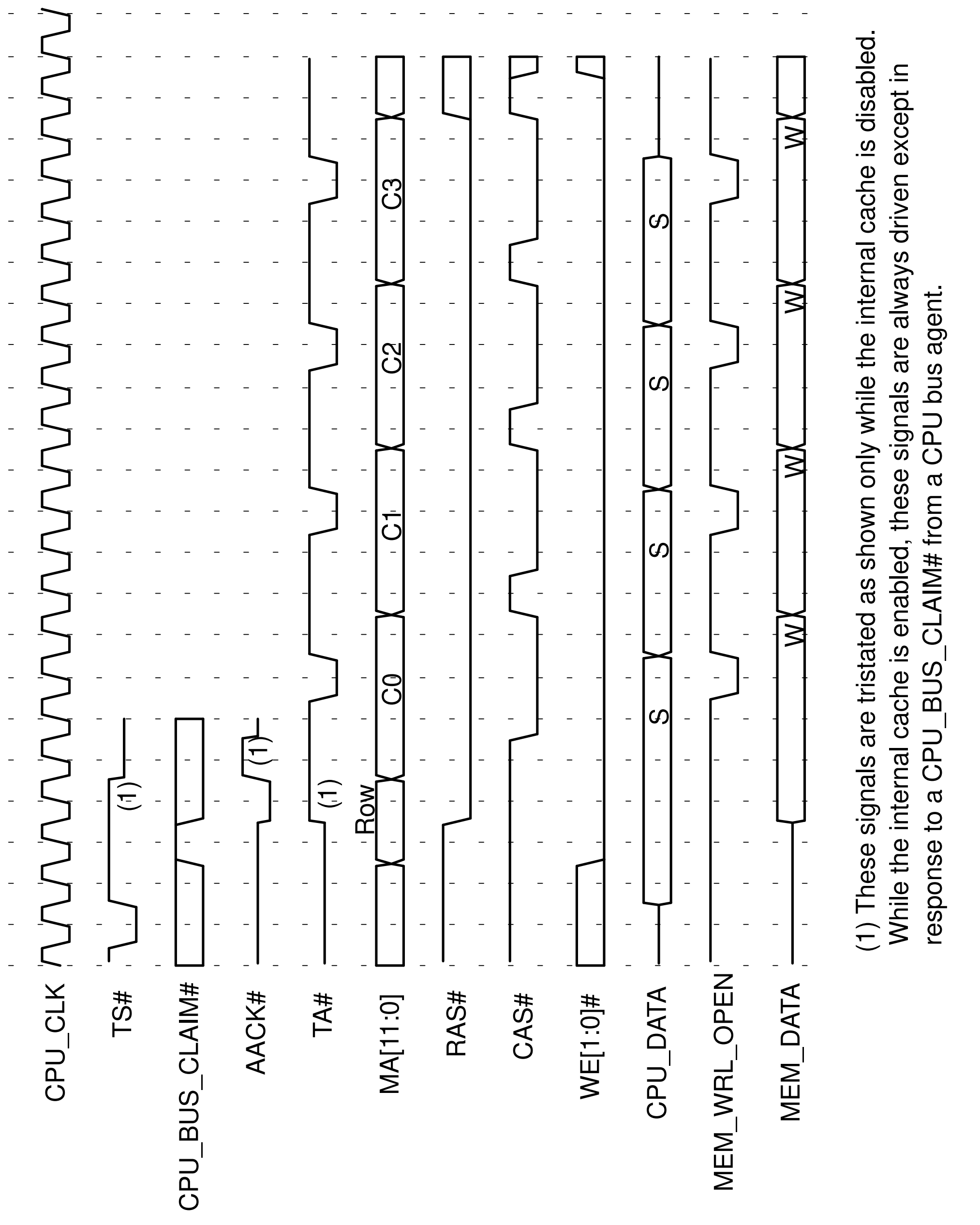


Figure A-6. CPU to Memory Write (Page DRAM) From Bus Idle

A.5.2 CPU to Memory Write (Page DRAM) Followed by Write Hit

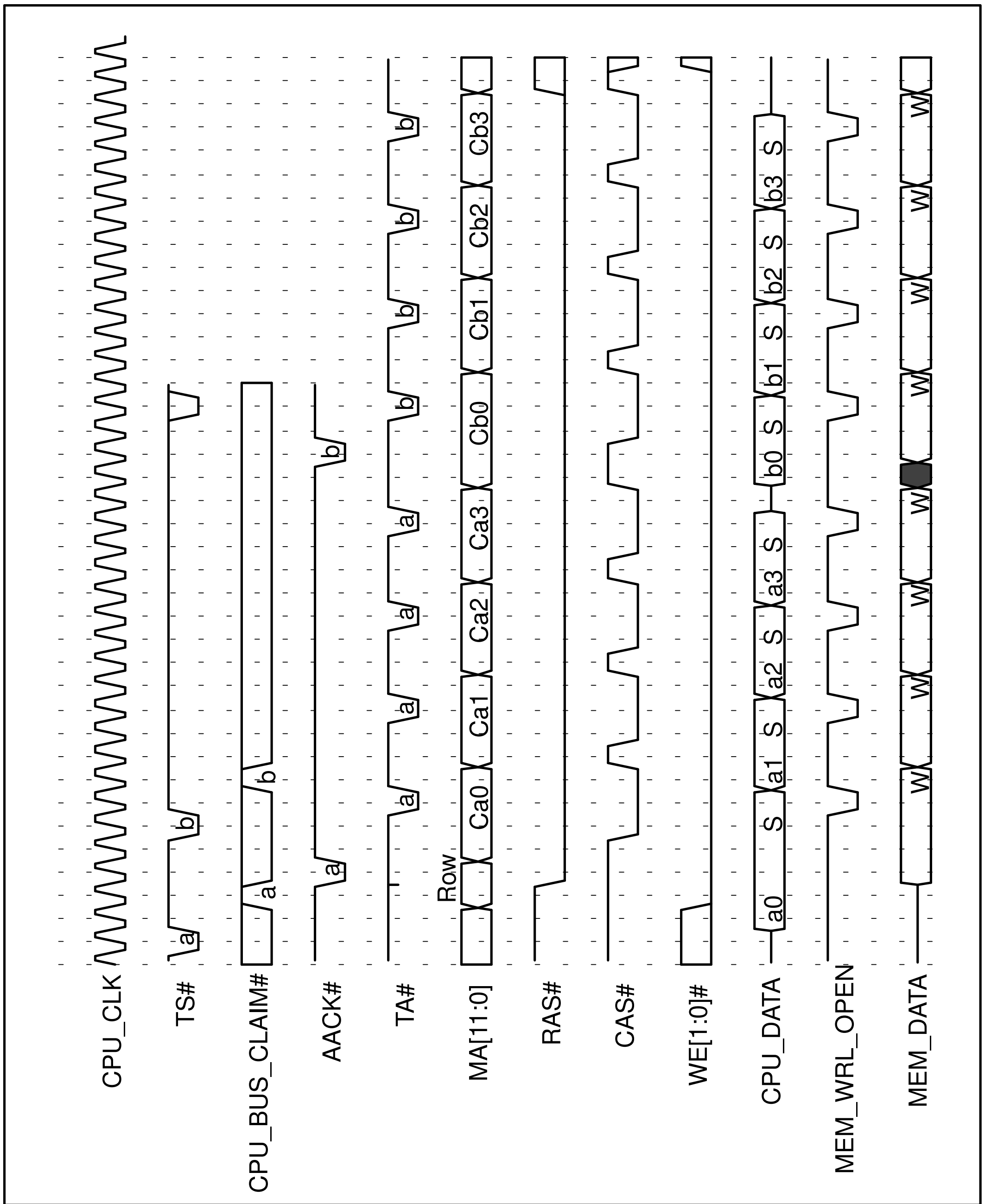


Figure A-7. CPU to Memory Write (Page DRAM) Followed by Write Hit

A

A.5.3 CPU to Memory Write (Page DRAM) Followed by Write Page Miss and Bank Miss

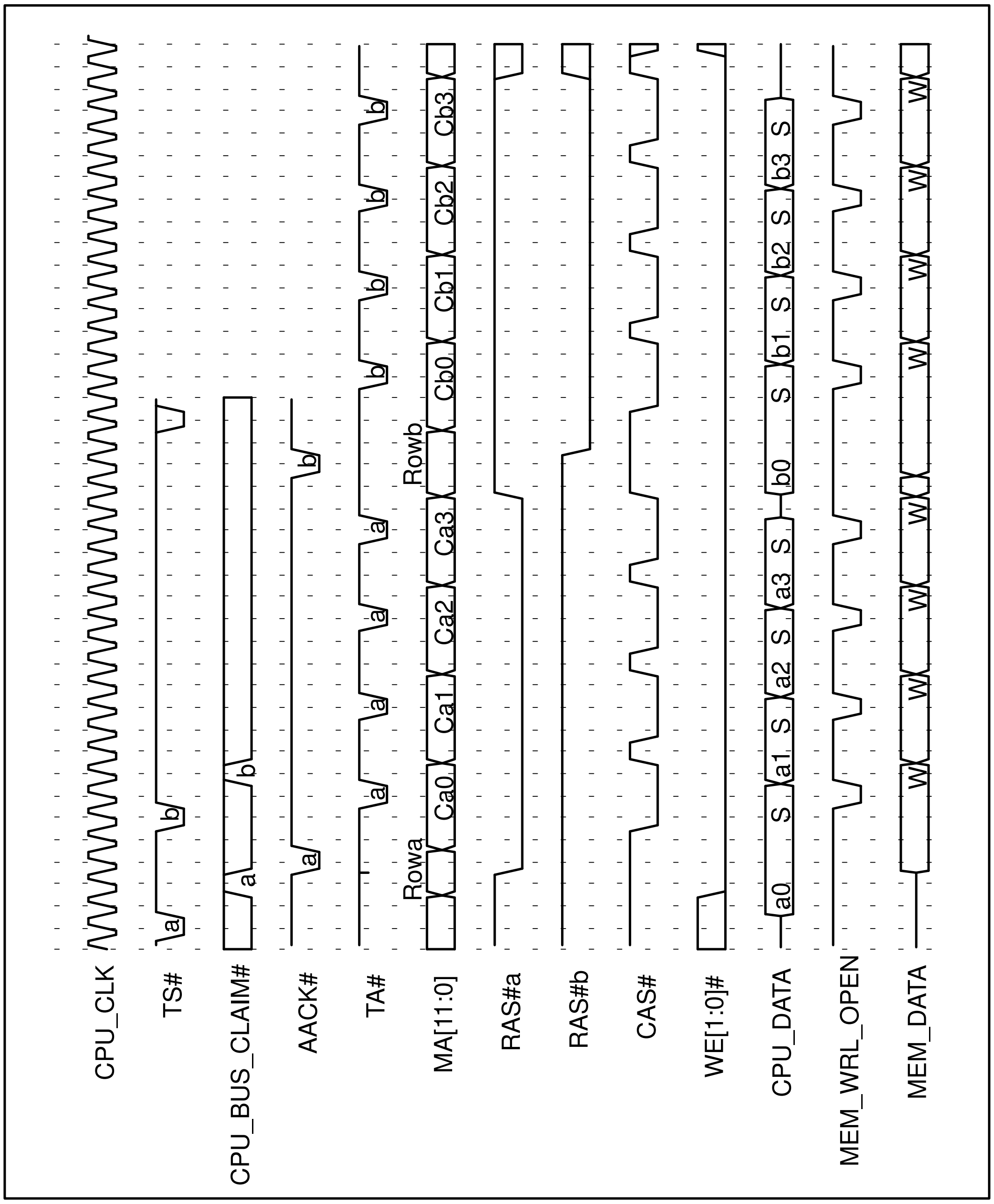


Figure A-8. CPU to Memory Write (Page DRAM) Followed by Write Page Miss and Bank Miss

A.5.4 CPU to Memory Write (Page DRAM) Followed by Read Hit

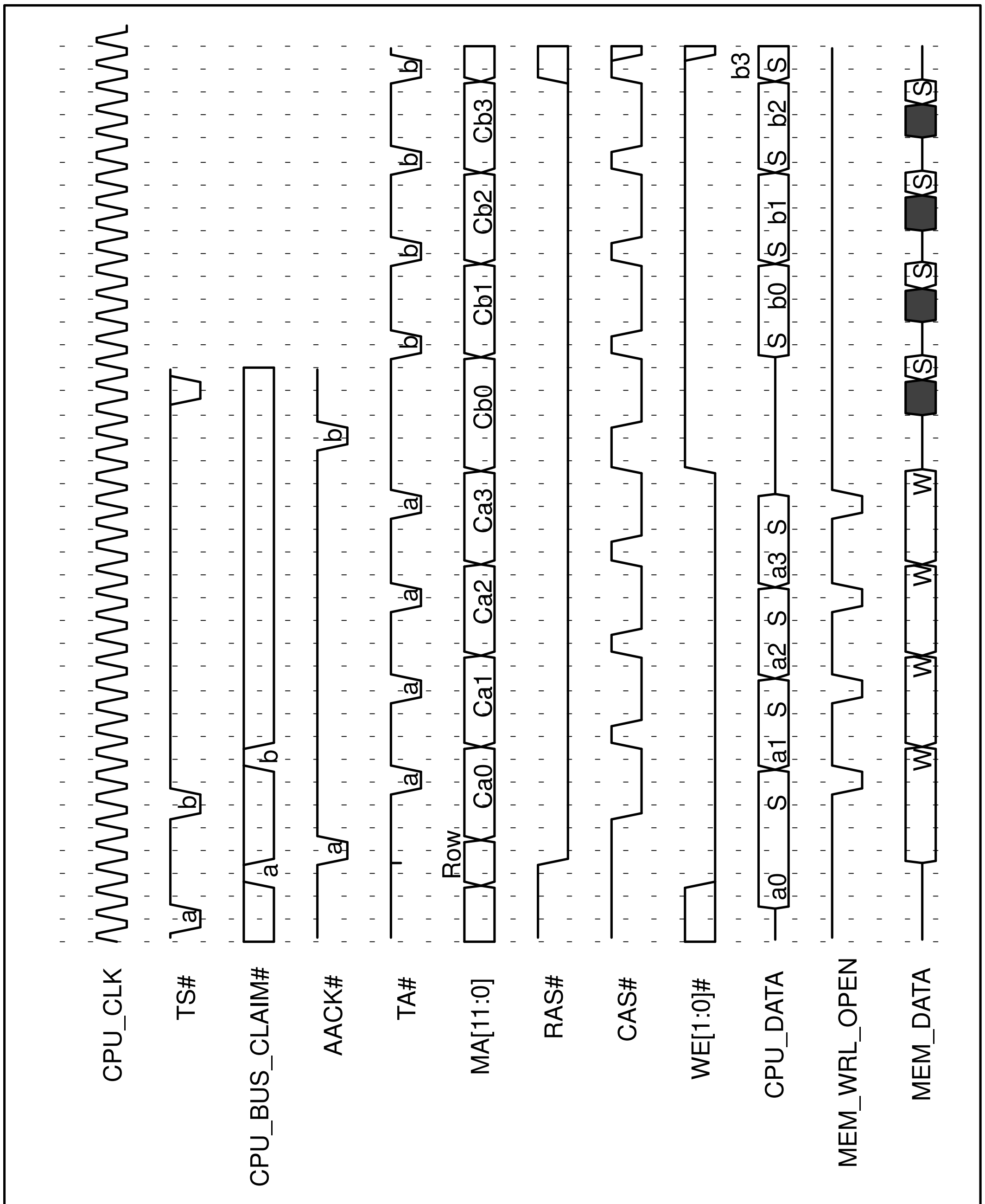


Figure A-9. CPU to Memory Write (Page DRAM) Followed by Read Hit

A

A.5.5 CPU to Memory Write (Page DRAM) Read/Modify/Write From Bus Idle

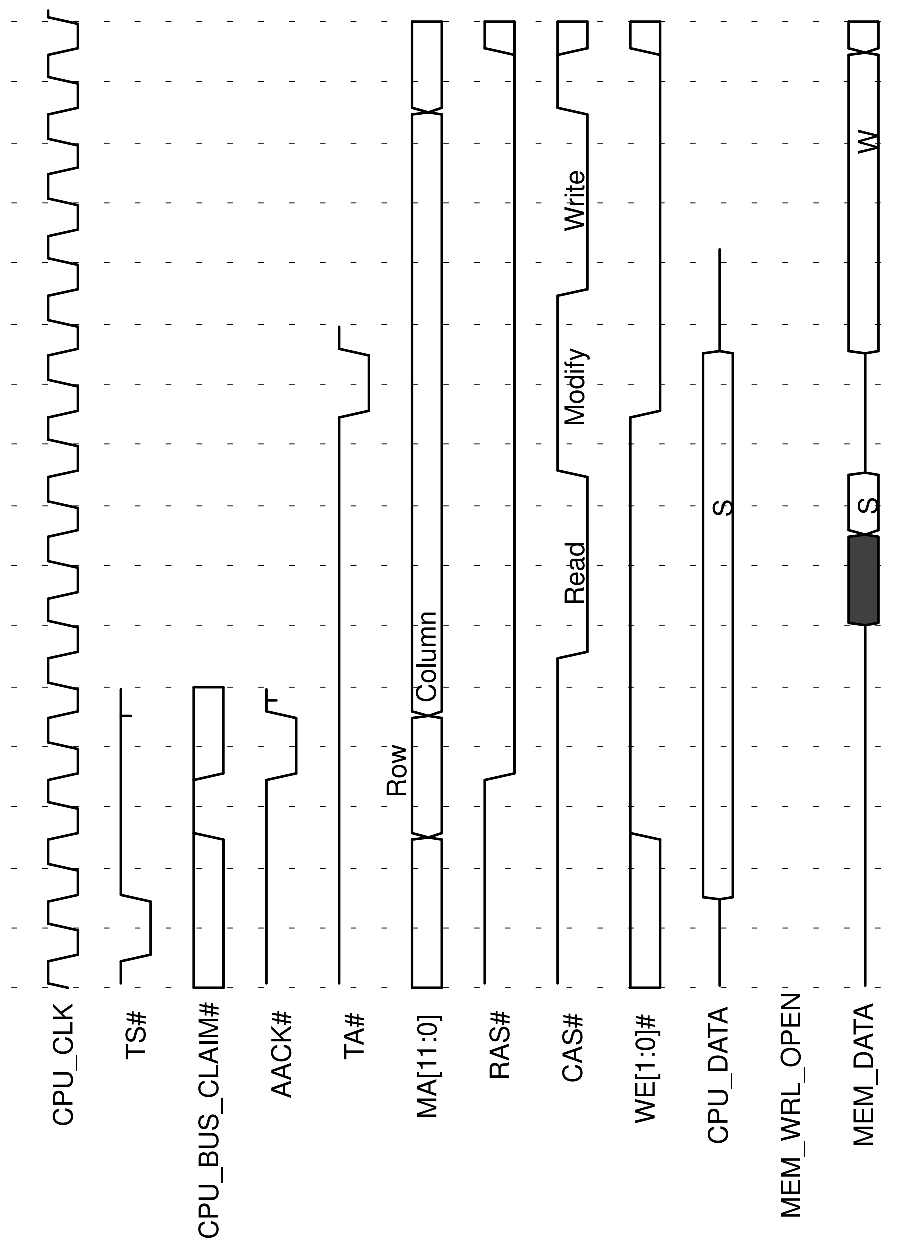
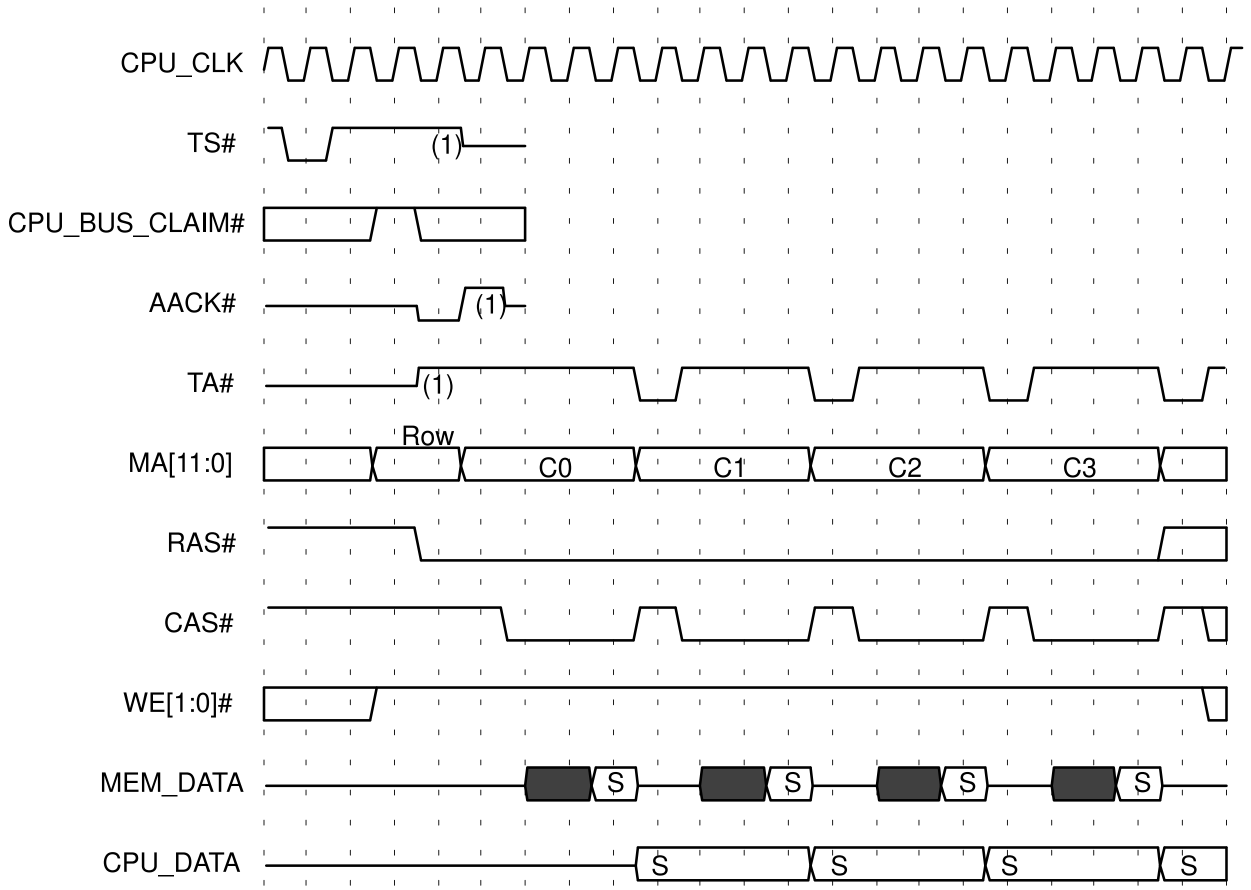


Figure A-10. CPU to Memory Write (Pg. DRAM) Read/Modify/Write From Bus Idle

A

A.5.6 CPU to Memory Read (Page DRAM) From Bus Idle

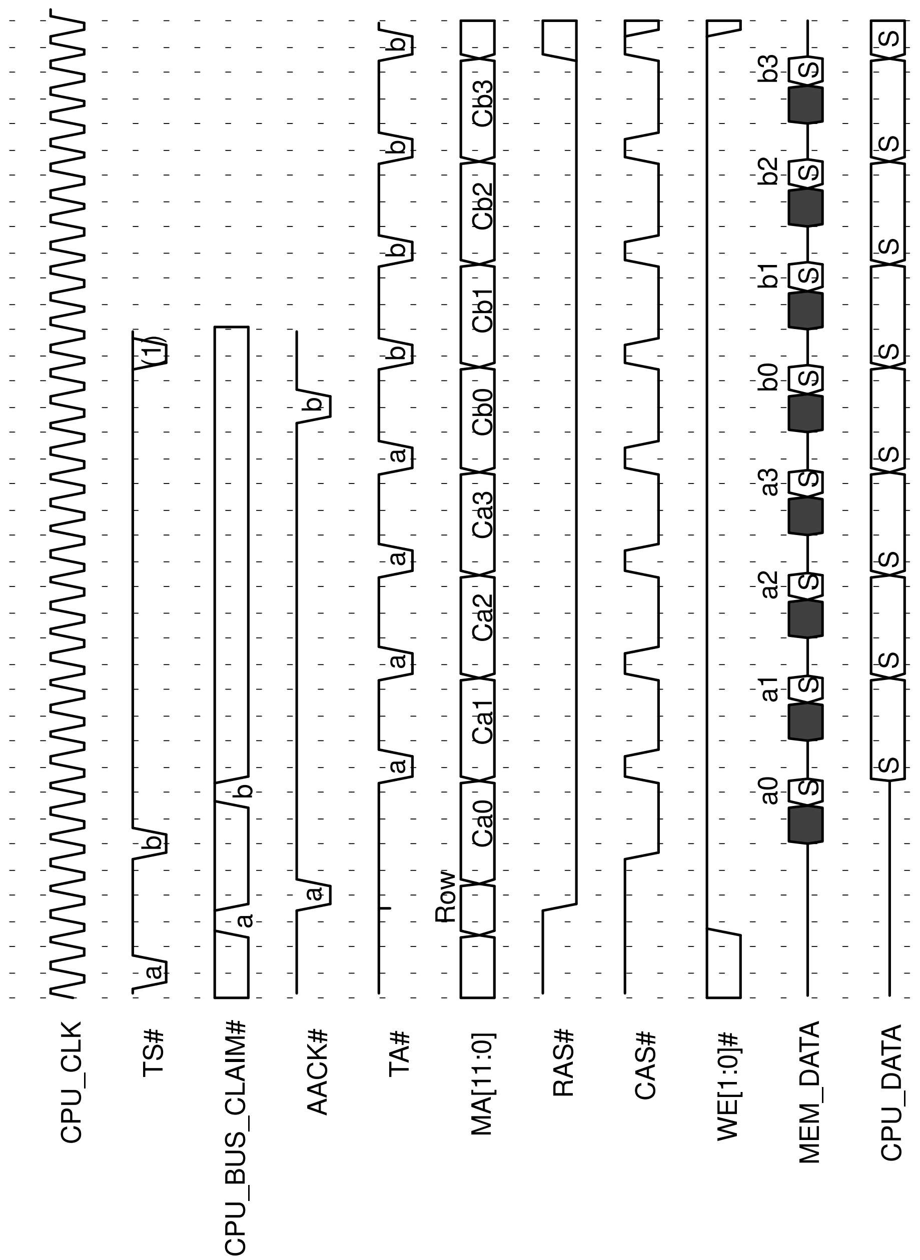


(1) These signals are tristated as shown only while the internal cache is disabled. While the internal cache is enabled, these signals are always driven except in response to a CPU_BUS_CLAIM# from a CPU bus agent.

Figure A-11. CPU to Memory Read (Page DRAM) From Bus Idle

A

A.5.7 CPU to Memory Read (Page DRAM) Followed by Read Hit



(1) Earliest that another TS# could be issued.

Figure A-12. CPU to Memory Read (Page DRAM) Followed by Read Hit

A.5.8 CPU to Memory Read (Page DRAM) Followed by Read Miss and Bank Miss

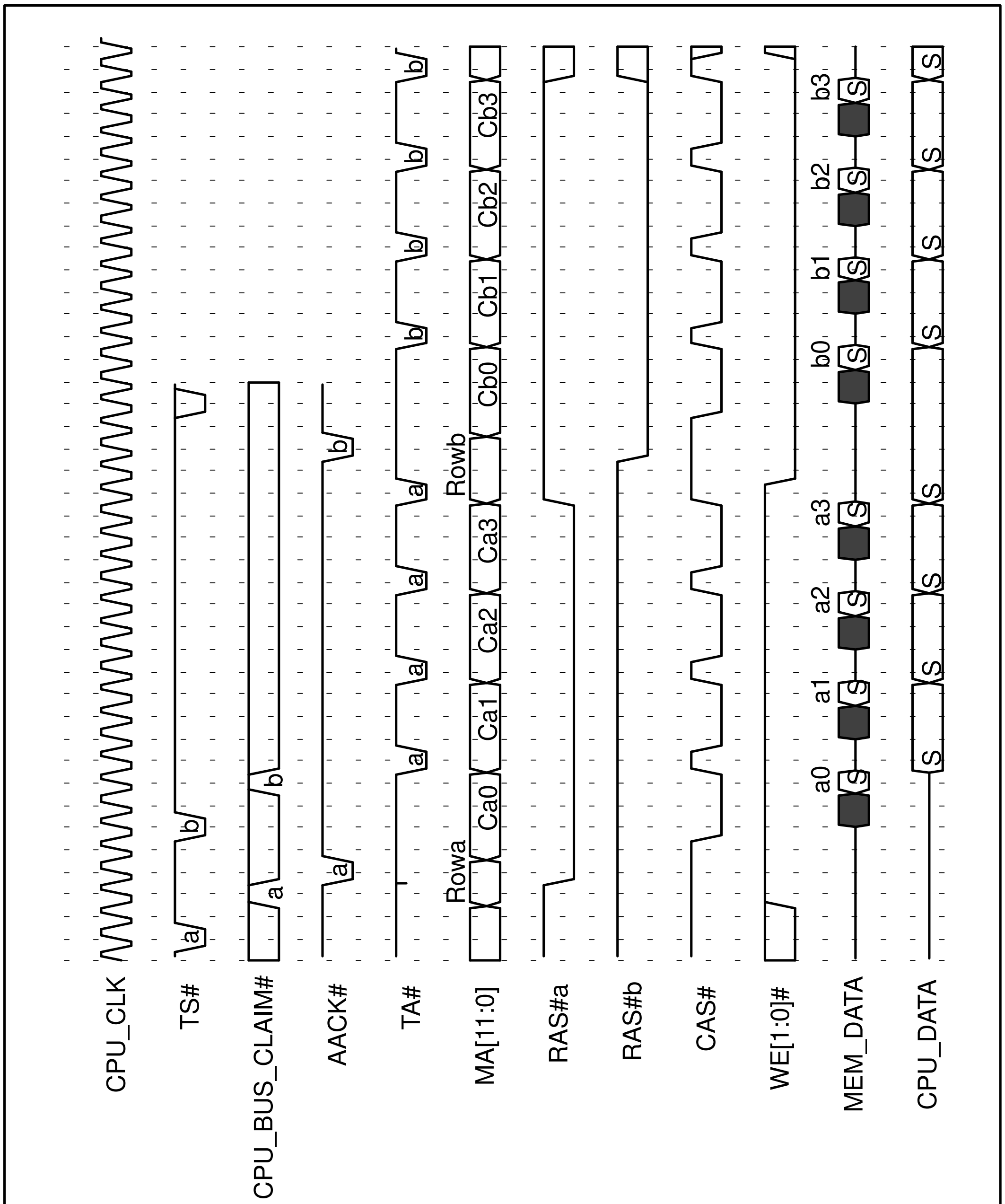


Figure A-13. CPU to Memory Read (Page DRAM) Followed by Read Miss and Bank Miss

A.5.9 CPU to Memory Read (Page DRAM) Followed by Write Hit

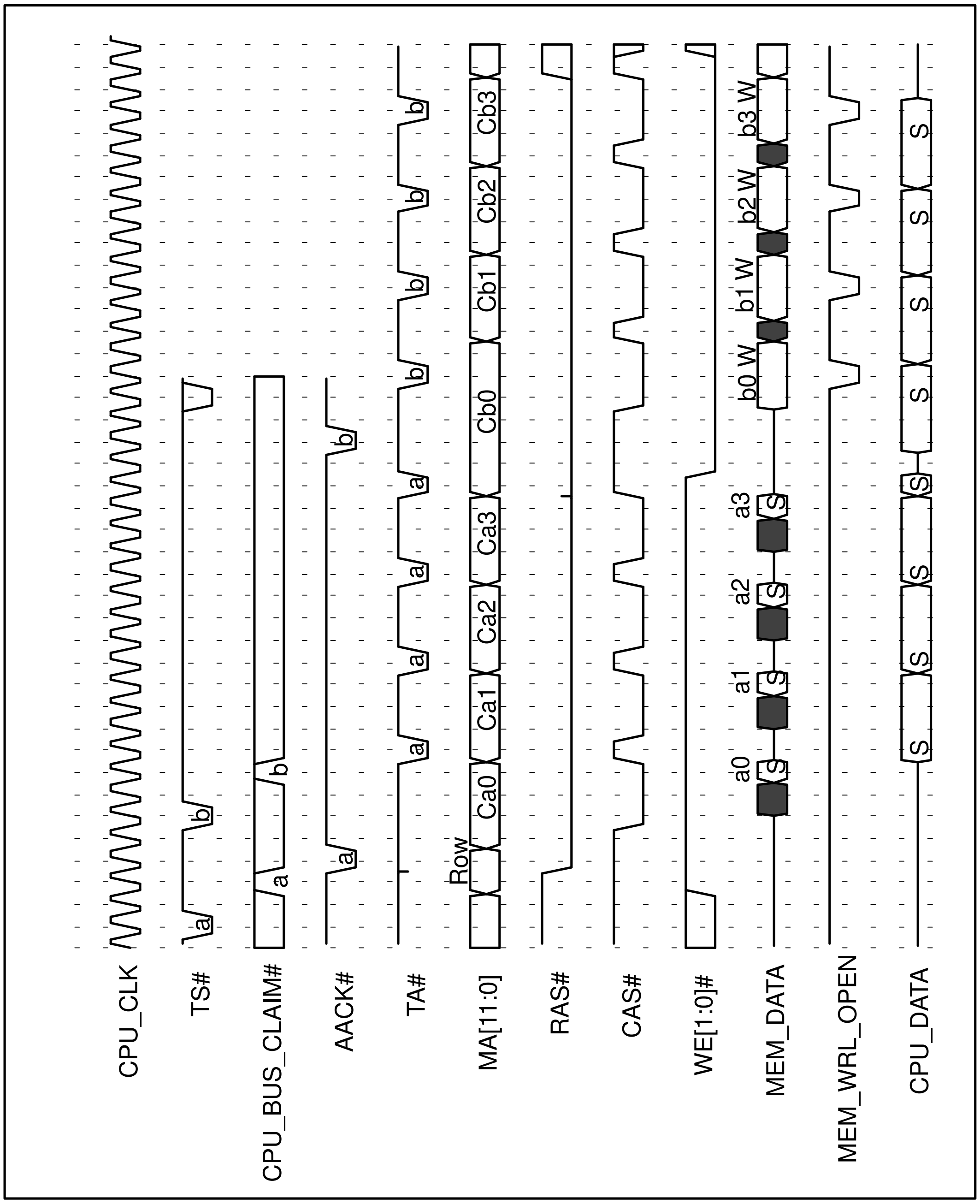
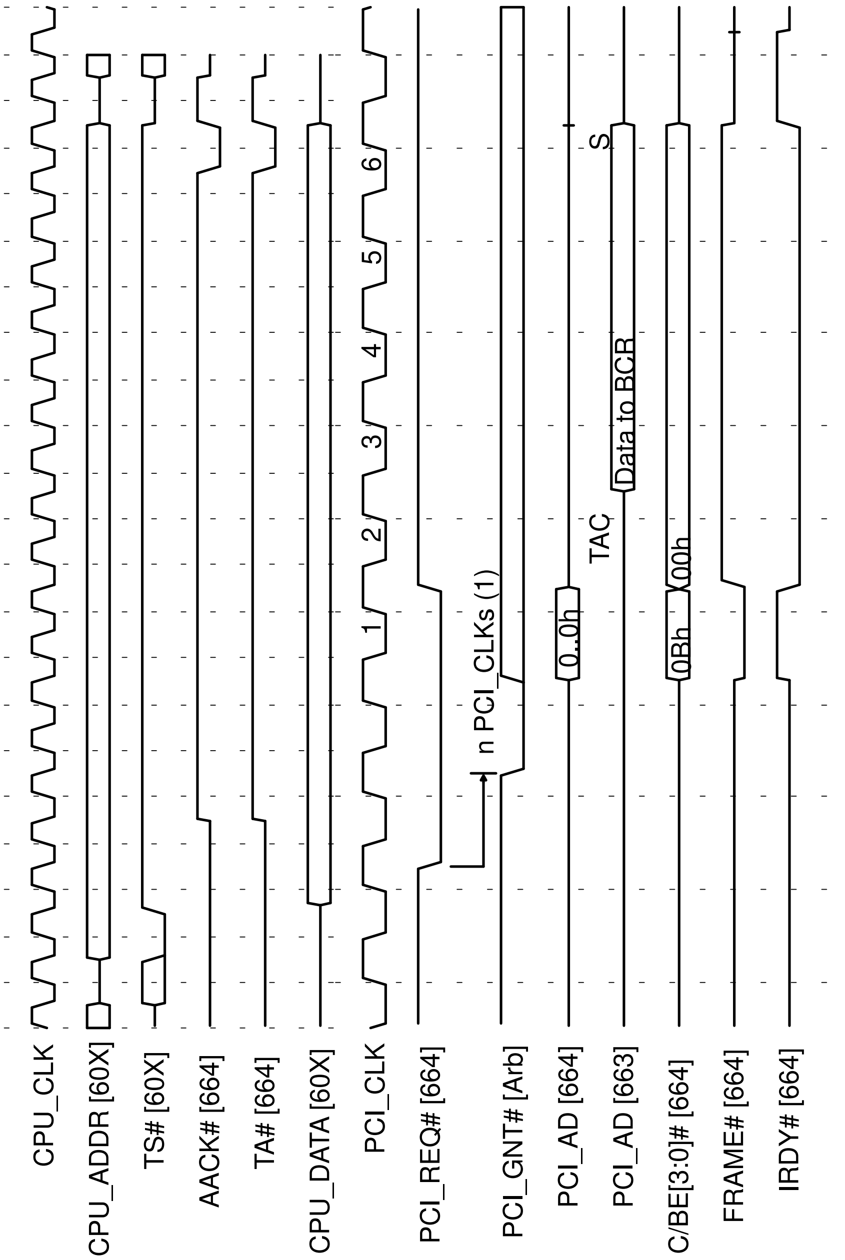


Figure A-14. CPU to Memory Read (Page DRAM) Followed by Write Hit

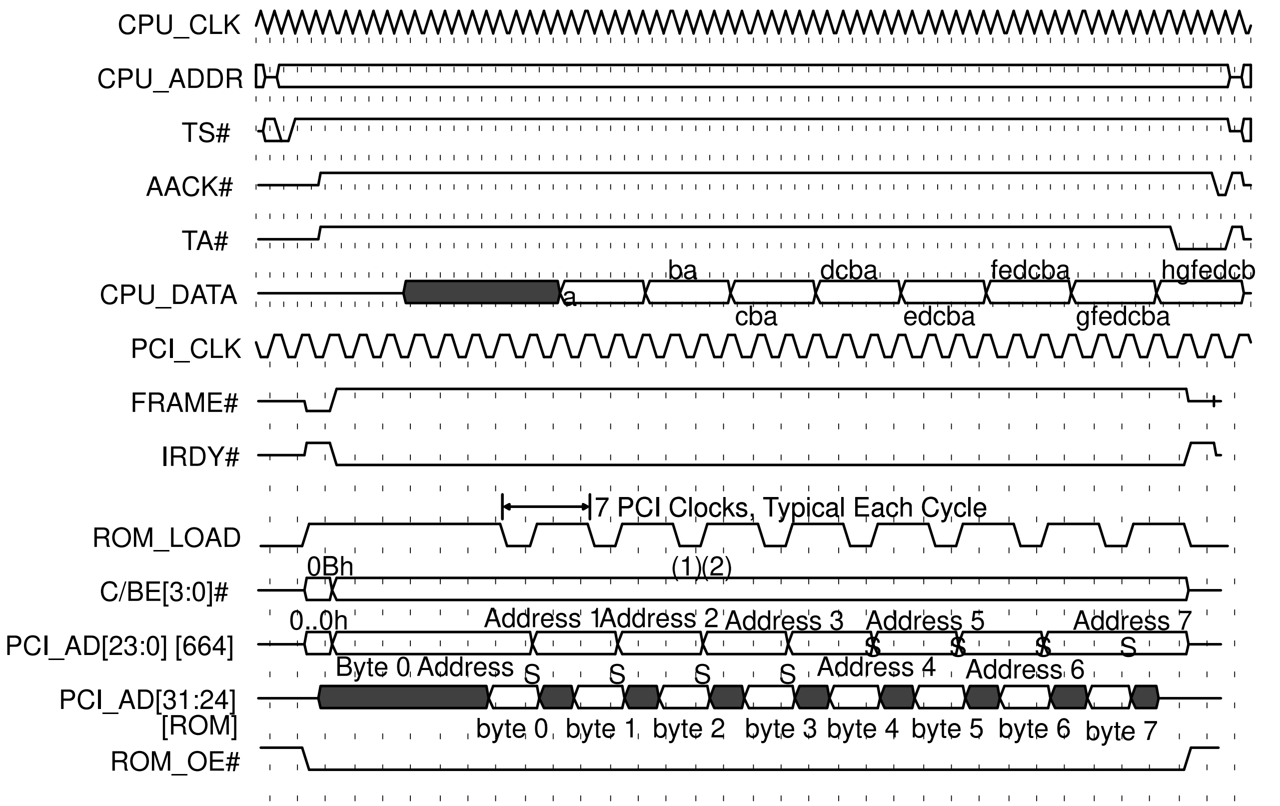
A.5.10 CPU to Bridge Write of Bridge Control Register



Note that: DEVSEL#, IDSEL#, TRDY#, and STOP# are not asserted during the transaction.
(1) This delay controlled by the system arbiter.

Figure A-15. CPU to Bridge Write of Bridge Control Register

A.5.11 CPU to Bridge Read of ROM



Note that: DEVSEL#, IDSEL#, TRDY#, STOP#, and ROM_WE# are not asserted during the transaction.

1 Diagram assumes that the PCI bus is parked on the CPU.

2 (1) shift (2) load

Figure A-16. CPU to Bridge Read of ROM

A.5.12 CPU to Bridge Read of Bridge Control Register

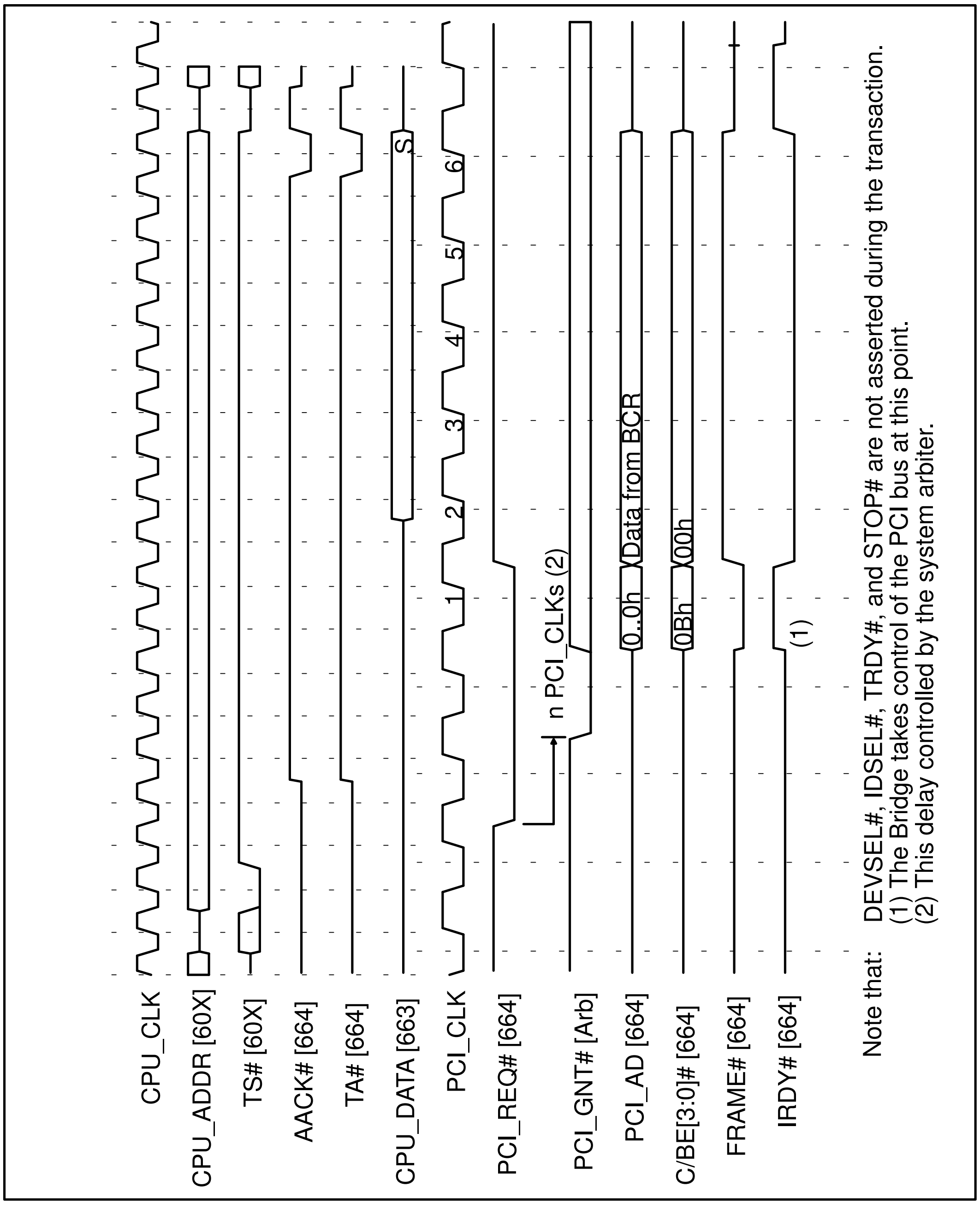
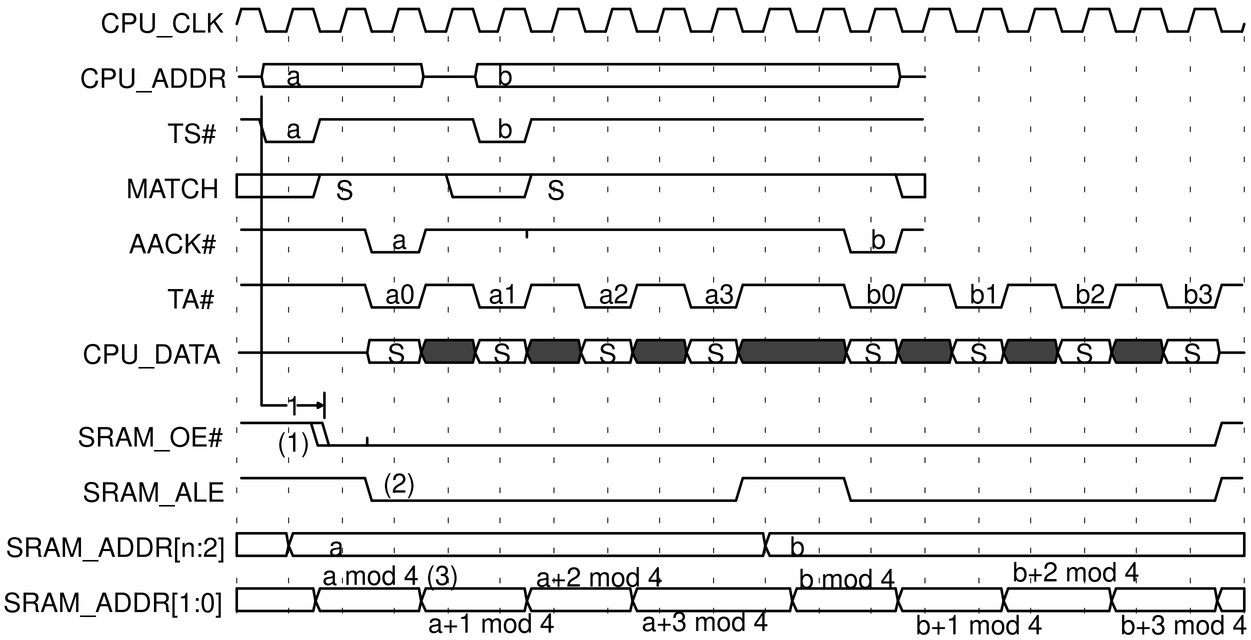


Figure A-17. CPU to Bridge Read of Bridge Control Register

A.5.13 CPU to Memory Read, L2 Cache w/Async SRAMS, Hit



(1) On cacheable CPU to memory reads, SRAM_OE# is speculatively asserted.

If MATCH is sampled deasserted, SRAM_OE# is deasserted at the tick mark.

(2) SRAM_ADDR[n:2] are held by a transparent external latch while ALE is low.

(3) SRAM_A[1:0] is incremented in a circular manner (00, 01, 10, 11, 00,...) and may begin in any sta

Figure A-18. CPU to Memory Read, L2 Cache w/Async SRAMS, Hit

A.5.14 CPU to Memory Read, L2 Cache w/Burst SRAMs, Hit

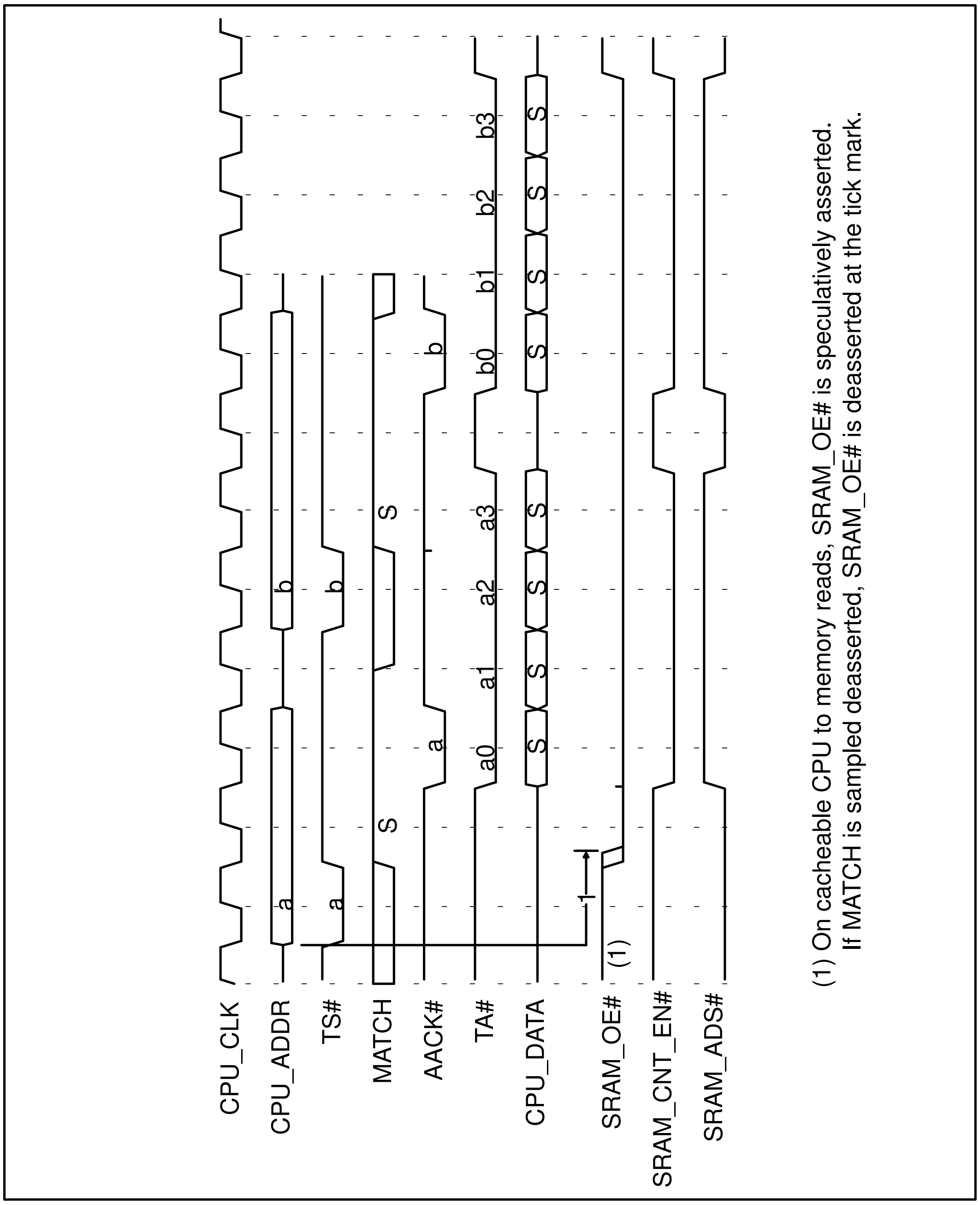


Figure A-19. CPU to Memory Read, L2 Cache w/Burst SRAMs, Hit

A.5.15 CPU to Memory Read (EDO DRAM) Cache Miss Followed by Read Hit Cache Miss w/Async SRAMs

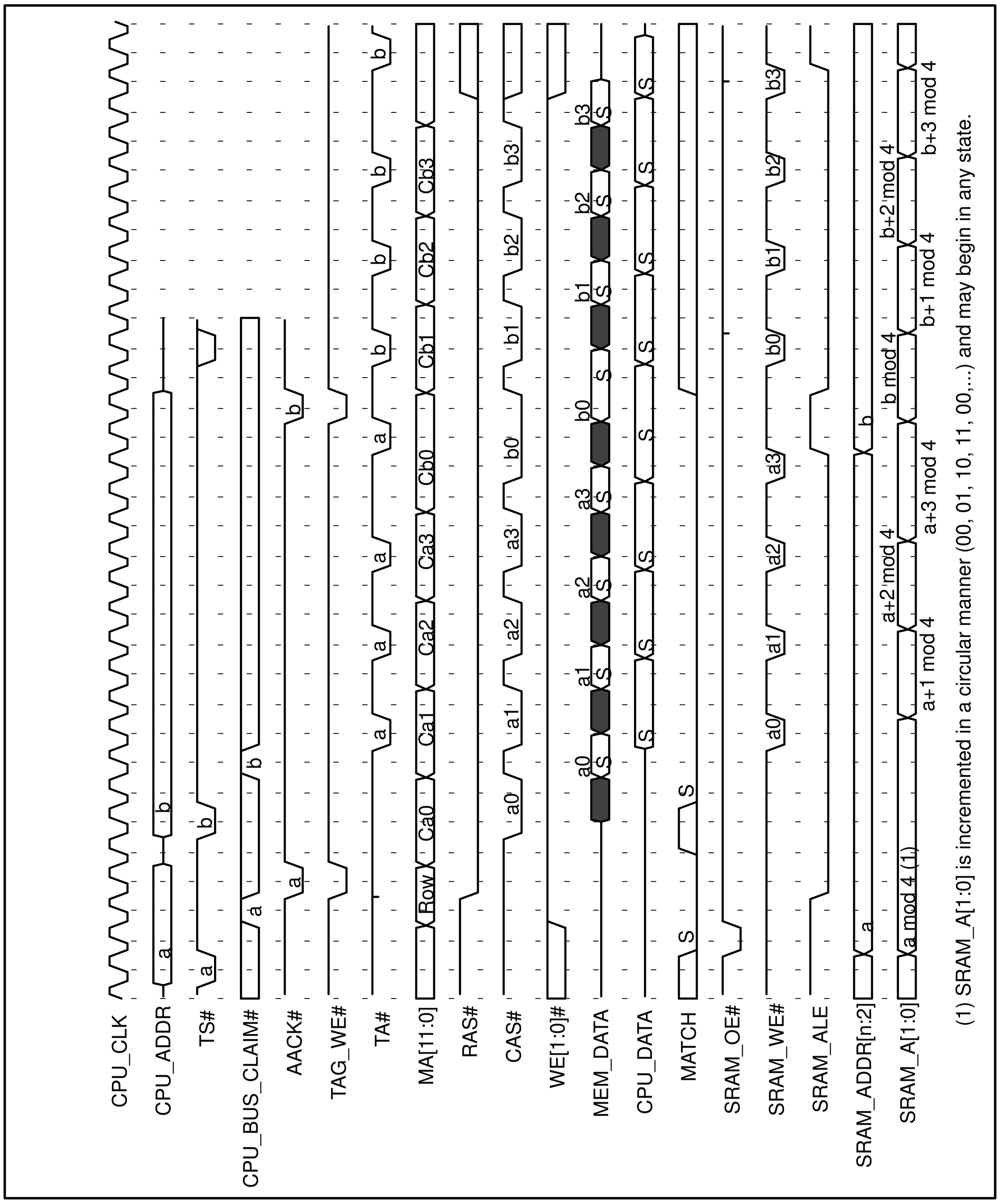


Figure A-20. CPU to Memory Read (EDO DRAM) Cache Miss Followed by Read Hit Cache Miss w/Async SRAMs

A.5.16 CPU to Memory Read (EDO DRAM) Cache Miss Followed by Read Hit Cache Miss w/Burst SRAMs

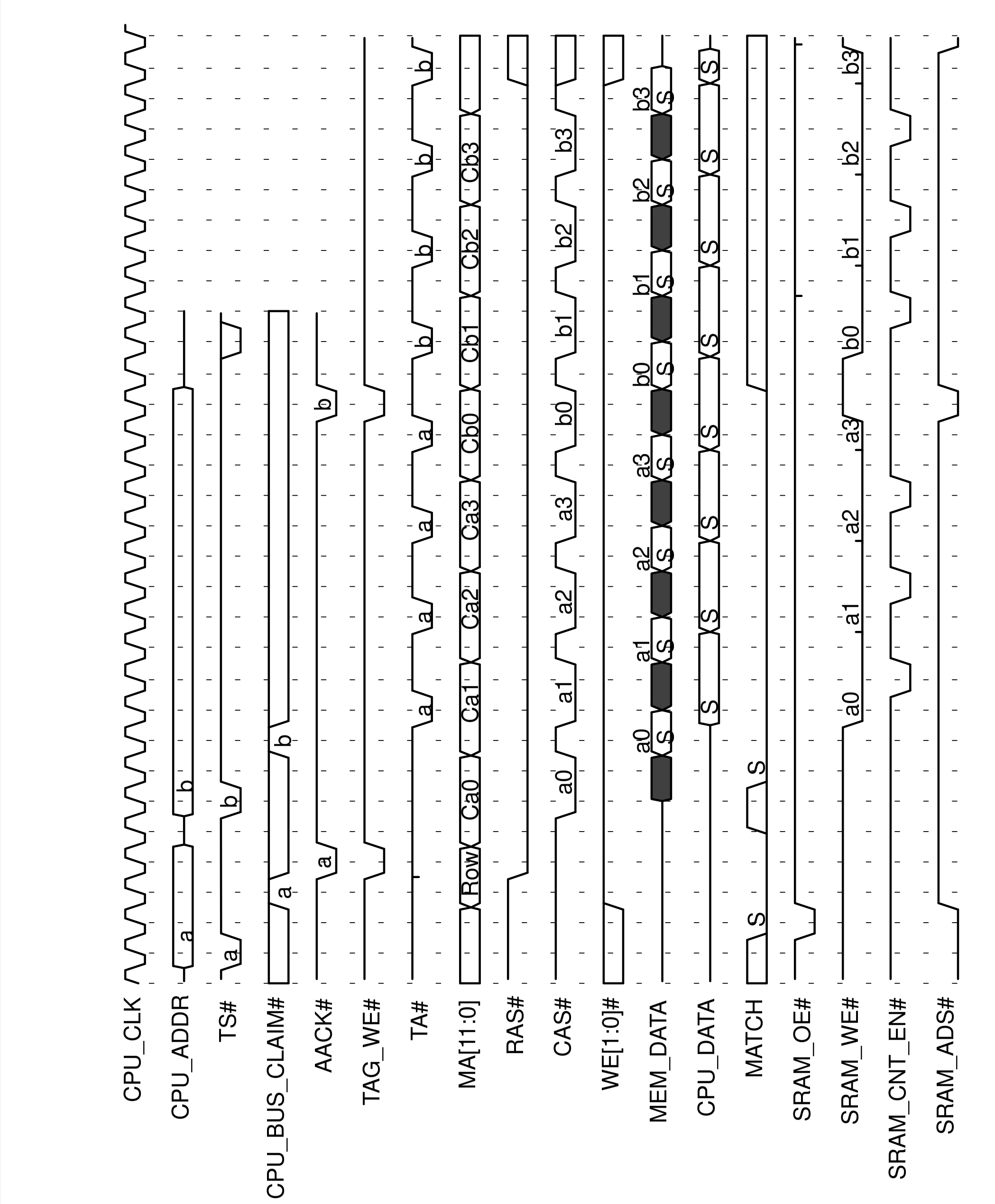


Figure A-21. CPU to Memory Read (EDO DRAM) Cache Miss Followed by Read Hit Cache Miss w/Burst SRAMs

A.5.17 CPU to Memory Read (EDO DRAM) Followed by Read Hit

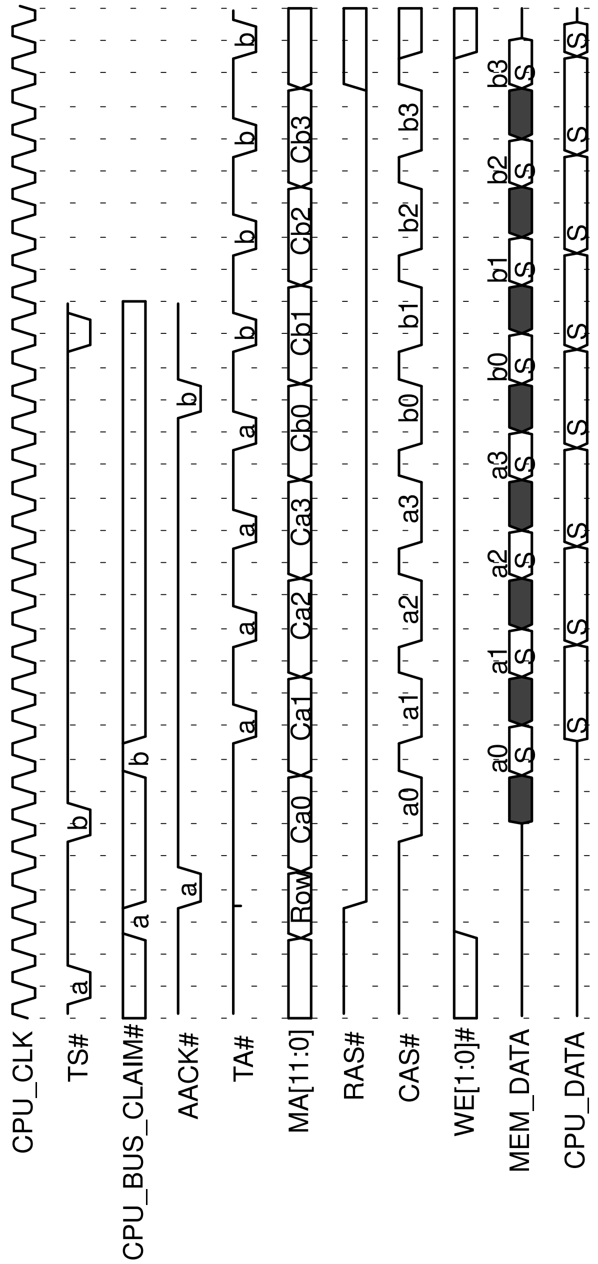


Figure A-22. CPU to Memory Read (EDO DRAM) Followed by Read Hit

A.6 663 Buffer Timing By Signal

Table D–6. 663 Buffer DC Characteristics By Signal (See Note 1)

Signal	Pin	I/O	Input (ns)		Output (ns)				
			Setup (min)	Hold (min)	Valid Delay (Max)	Enable Delay (Max)	Hold Time (min)	Hold (to 3S) (min)	Tristate Delay (Max)
AOS_RR_MMRS	176	I	*	*	—	—	—	—	—
C2P_WRL_OPEN	150	I	*	*	—	—	—	—	—
CPU_DATA[00:63]	—	I/O	5.7	0	8.7	10	3.3	3	8.5
CPU_DATA_OE#	138	I	*	*	—	—	—	—	—
CPU_DPAR[0:7]	126	I/O	4	0	8.7	10	3.3	3	8.5
CPU_PAR_ERR#	178	O	—	—	*	—	*	—	—
CPU_RDL_OPEN	141	I	*	*	—	—	—	—	—
CRS_C2PWXS	147	I	*	*	—	—	—	—	—
DUAL_CTRL_REF	175	I	*	*	—	—	—	—	—
ECC_LE_SEL	145	I	*	*	—	—	—	—	—
MEM_BE[3:0]	—	I	*	*	—	—	—	—	—
MEM_CHECK[0:7]	—	I/O	2	1	16.8	17.5	4.6	2.5	7.9
MEM_DATA[63:0]	—	I/O	5.2	1	16.8	17.5	4.6	2.5	7.9
MEM_DATA_OE#	137	I	*	*	—	—	—	—	—
MEM_ERR#	179	O	—	—	*	—	*	—	—
MEM_RD_SMPL	139	I	*	*	—	—	—	—	—
MEM_WRL_OPEN	146	I	*	*	—	—	—	—	—
MWS_P2MRXS	148	I	*	*	—	—	—	—	—
PCI_AD[31:0]	—	I/O	7	0	11	12	2	2	28
PCI_AD_OE#	136	I	*	*	—	—	—	—	—
PCI_EXT_SEL	149	I	*	*	—	—	—	—	—
PCI_IRDY#	177	I	7	0	—	—	—	—	—
PCI_OL_OPEN	151	I	*	*	—	—	—	—	—
PCI_OUT_SEL	153	I	*	*	—	—	—	—	—
ROM_LOAD	140	I	*	*	—	—	—	—	—
SBE#	180	O	—	—	*	—	*	—	—

* These signals interconnect the 663 and the 664. Keep these lines point to point and as short as possible to maintain minimum flight time between the 663 and the 664.

—These timing specifications are not applicable to the signal.

Where required for timing purposes, the 663 Buffer enables the PCI_AD lines one cycle before asserting them.

A

A.7 664 Controller Timing By Signal

Table D–7. 664 Controller DC Characteristics By Signal (See Note 1)

Signal	Pin	I/O	Input (ns)		Output (ns)				
			Setup (min)	Hold (min)	Valid Delay (Max)	Enable Delay (Max)	Hold Time (min)	Hold (to 3S) (min)	Tristate Delay (Max)
AACK#	109	I/O	2.0	0	8.5	9.3	2.3	2.9	8.6
AOS_RR_MMRS	69	O	—	—	*	—	*	—	—
ARTRY#	110	I/O	3.8	0	8.9	10.6	2.4	3.2	10.3
C2P_WRL_OPEN	61	O	—	—	*	—	*	—	—
CAS[7:0]#	—	O	—	—	12.9	—	2.9	—	—
CPU_ADDR[0:31]	—	I/O	1.5	0	12.6	15.2	3.4	2.6	13.8
CPU_BUS_CLAIM#	132	I	4.1	1.0	—	—	—	—	—
CPU_DATA_OE#	197	O	—	—	*	—	*	—	—
CPU_GNT1#	134	O	—	—	9.0	—	2.4	—	—
CPU_GNT2#	135	O	—	—	9.0	—	2.4	—	—
CPU_PAR_ERR#	192	I	*	*	—	—	—	—	—
CPU_RDL_OPEN	50	O	—	—	*	—	*	—	—
CPU_REQ1#	127	I	2.1	0	—	—	—	—	—
CPU_REQ2#	128	I	2.1	0	—	—	—	—	—
CRS_C2PWXS	65	O	—	—	*	—	*	—	—
DPE#	133	I	1.2	0	—	—	—	—	—
DUAL_CTRL_REF	205	O	—	—	*	—	*	—	—
ECC_LE_SEL	2	O	—	—	*	—	*	—	—
GBL#	120	O	—	—	11.9	11.9	2.9	2.9	13.4
IGN_PCI_AD31	57	I	1.4	0	—	—	—	—	—
INT_CPU#	139	O	—	—	14.1	—	3.5	—	—
INT_REQ	55	I	1.0	1.0	—	—	—	—	—
MA[11:0]	—	O	—	—	13.7	—	2.9	—	—
MCP#	138	O	—	—	—	14.9	—	4.0	14.9
MEM_BE[3:0]	206	O	—	—	*	—	*	—	—
MEM_DATA_OE#	196	O	—	—	*	—	*	—	—
MEM_ERR#	194	I	*	*	—	—	—	—	—
MEM_RD_SMPL	49	O	—	—	*	—	*	—	—
MEM_WRL_OPEN	51	O	—	—	*	—	*	—	—
MWS_P2MRXS	66	O	—	—	*	—	*	—	—
NMI_REQ	56	I	1.5	.5	—	—	—	—	—
PCI_AD[31:0]		I/O	7.0	0	11.0	11.0	2.0	2.0	28.0
PCI_AD_OE#	195	O	—	—	*	—	*	—	—
PCI_C/BE[3:0]#	—	I/O	7.0	0	11.0	11.0	2.0	2.0	28.0
PCI_DEVSEL#	204	I/O	7.0	0	11.0	11.0	2.0	2.0	28.0
PCI_EXT_SEL	67	O	—	—	*	—	*	—	—
PCI_FRAME#	200	I/O	7.0	0	11.0	11.0	2.0	2.0	28.0

A

Table D–7. 664 Controller DC Characteristics By Signal (See Note 1) (Continued)

Signal	Pin	I/O	Input (ns)		Output (ns)				
			Setup (min)	Hold (min)	Valid Delay (Max)	Enable Delay (Max)	Hold Time (min)	Hold (to 3S) (min)	Tristate Delay (Max)
PCI_GNT#	54	I	7.0	0	—	—	—	—	—
PCI_IRDY#	201	I/O	7.0	0	11.0	11.0	2.0	2.0	28.0
PCI_LOCK#	53	I	7.0	0	—	—	—	—	—
PCI_OL_OPEN	64	O	—	—	*	—	*	—	—
PCI_OUT_SEL	68	O	—	—	*	—	*	—	—
PCI_PAR	7	I/O	7.0	0	11.0	11.0	2.0	2.0	28.0
PCI_PERR#	10	I/O	7.0	0	11.0	11.0	2.0	2.0	28.0
PCI_REQ#	58	O	—	—	12.0	—	2.0	—	—
PCI_SERR#	71	O	—	—	11.0	11.0	2.0	2.0	28.0
PCI_STOP#	203	I/O	7.0	0	11.0	11.0	2.0	2.0	28.0
PCI_TRDY#	202	I/O	7.0	0	11.0	11.0	2.0	2.0	28.0
RAS[7:0]#	—	O	—	—	12.8	—	2.8	—	—
RESET#	156	I	2.0	0	—	—	—	—	—
ROM_LOAD	70	O	—	—	*	—	*	—	—
ROM_OE#	47	O	—	—	16.0	—	3.9	—	—
ROM_WE#	60	O	—	—	16.0	—	3.8	—	—
SBE#	193	I	*	*	—	—	—	—	—
SHD#	141	O	—	—	9.6	9.6	—	2.2	10.6
SRAM_ADS#/ADDR0	124	O	—	—	10.8	—	3.0	—	—
SRAM_ALE	119	O	—	—	11.9	—	3.0	—	—
SRAM_CNT_EN#/ ADDR1	125	O	—	—	10.8	—	3.0	—	—
SRAM_OE#	117	O	—	—	15.7	—	3.4	—	—
SRAM_WE#	118	O	—	—	11.0	—	3.2	—	—
STOP_CLK_EN#	151	I	2.1	0	—	—	—	—	—
TA#	111	I/O	4.5	0	8.6	16.7	2.2	3.3	15.7
TAG_CLR#	116	O	—	—	14.2	—	4.0	—	—
TAG_MATCH	142	I	3.0	1.0	—	—	—	—	—
TAG_VALID	115	O	—	—	12.6	—	3.4	—	—
TAG_WE#	114	O	—	—	13.0	—	3.6	—	—
TBST#	144	I	1.0	0	—	13.7	—	2.5	11.9
TEA#	137	O	—	—	8.3	—	2.2	—	—
TS#	143	I/O	2.0	0	8.3	10.8	2.2	3.3	10.2
TSIZE[0:2]	—	I/O	0	0	13.8	13.8	2.5	2.5	11.9
TT[0:4]	—	I/O		0	14.5	14.5	2.5	2.5	12.9
WE[1:0]#	—	O	—	—	11.8	—	2.7	—	—
XATS#	129	I	1.0	0	—	—	—	—	—

* These signals interconnect the 663 and the 664. Keep these lines point to point and as short as possible to maintain minimum flight time between the 663 and the 664.

—These timing specifications are not applicable to the signal.

Typically asynchronous signals, such as RESET#, INT_REQ, and NMI_REQ are metastable-hardened. If their assertion timing violates the input setup or hold times for a given clock cycle, these signals will be recognized on a subsequent clock.

A

Appendix B
Example Implementation

This section contains schematics for an example system. The schematics illustrate the implementation of a 660 Bridge chip set with a PowerPC 603 or 604 microprocessor. The example system design includes provision for an external L2 cache or an upgrade microprocessor. Four SIMM slots allow up to 512M of system memory to be installed in any configuration in the four slots.

The example system uses the IBM27-82376 I/O bridge chip or comparable device for ISA bus support. The I/O bridge also provides refresh timing and an X-bus interface for various system support functions.

The example schematics are available in a separate package.

Appendix C
The 663 Buffer

This section is currently undergoing revision.

C

C

Appendix D
Electrical and Mechanical Characteristics

Unless otherwise noted, all specifications in this section apply to both the 663 Buffer and to the 664 Controller.

D.1 Absolute Maximum Ratings

Stresses in excess of those listed in Table D-1 may damage and/or decrease the reliability of the 660 Bridge. Additionally, stressing the 660 Bridge in excess of the conditions listed as *Recommended Operating Conditions* is neither intended nor supported. All voltages are referenced to ground (V_{SS}).

Table D-1. Absolute Maximum Ratings, 660 Bridge

Symbol	Parameter	Min	Max	Units
Tjst	Junction Temperature, Storage	-40	125	deg C
Tjp	Junction Temperature, Power Applied	-25	100	deg C
V_{DD}	Supply Voltage	2.7	3.9	V
Vi	DC Voltage Applied to Any Input	–.5	5.5	V
Vo	DC Voltage Applied to Any Output (Output Tri-stated)	–.5	5.5	V
	ESD Withstand	2.2	—	kV
	Latchup current	100	—	mA

D

D.2 Recommended Operating Conditions

Table D-2 lists the conditions under which the 660 Bridge is intended to operate.

Table D-2. Recommended Operating Conditions, 660 Bridge

Symbol	Parameter	Min	Max	Units	Notes
V_{DD}	Supply Voltage	3.0	3.8	v	
V_I	DC Voltage Applied to Any Input Pin	–.5	5.5	v	(1)
V_O	DC Voltage Applied to Any Output Pin	–.5	5.5	v	(1)
Top	Junction Temperature, Operating	10	85	deg C	

Notes ForTable D-2 :

1) Allowed range of DC voltage applied to any I/O pin in input mode or to any input pin. The pins shown as Type = PCI may conduct excess current if forced above V_{DD} + 1.5v.

D.3 Power Dissipation and Thermal Characteristics

D.3.1 Power Dissipation

Neither the 653 Buffer nor the 654 Controller are expected to require a heat sink when used in the manner described in this manual. However, thermal management is a complex discipline, and the application is the responsibility of the designer. Final figures are not yet available, so Table D-3 contains conservative calculations based on laboratory testing. Using T_J(max) = 85 deg C, T_A = 50 deg C, and

$$\Theta j\text{–}a\ (\max) = \frac{T_J - T_A}{Pd}$$

we see Table D-3.

Table D-3. Estimated Power Dissipation and Required Θj–a at 50°C Ambient

Parameter	663 Buffer	664 Controller
Maximum Power Dissipation	600mW	400mW
Required Θj–a	58.3 °C/W	87.5 °C/W

D.3.2 Thermal Characteristics

Table D-4 shows typical thermal resistances associated with the 663 Buffer and the 664 Controller. Each row shows data for a given air flow condition at the chip package. The row titled *Convection* shows data for the chip package in free air with no forced air cooling, with the package mounted horizontally on the upper surface of a PCB. The other rows show data for a variety of forced air flow conditions. The values shown do not include a significant amount of heat flow through the pins of the chip, either to or from the PCB.

Examination of Table D-4 shows that neither the 663 Buffer or the 664 Controller require a heat sink, even with worst case estimated power dissipation, to maintain a junction temperature of less than 85 deg C in free air at 50 deg C ambient.

Table D-4. Typical Thermal Resistance, Junction to Ambient, No Heat Sink

Airflow	Θj–a, 663 Buffer	Θj–a, 664 Controller	Units
Convection	43	52	deg C/W
.25 M/s (50fpm)	37	45	deg C/W
.5 M/s (100fpm)	34	42	deg C/W
1 M/s (200fpm)	31	38	deg C/W

D.4 Common Characteristics

The specifications shown in Table D-5 are common to both the 663 Buffer and the 664 Controller, while within the recommended operating conditions envelope.

Table D-5. Common Characteristics

Symbol	Parameter	Type	Min	Max	Units	Notes
V_{IL}	Input Low Voltage	All	—	.8	v	(2)
V_{IH}	Input High Voltage	All	2.0	—	v	(2)
I_{IL}	Input Leakage Current	All	—	1	uA	(2)
V_{OL}	Output Low Voltage	TTL	—	.40	v	(3)
		PCI	—	.55	v	(3)
V_{OH}	Output High Voltage	TTL	2.4	—	v	(3)
		PCI	2.4	—	v	(3)
I_{O3S}	Output Tri-state Leakage Current	TTL	—	10	uA	(3)
		PCI	—	70	uA	(3)

Notes for Table D-5:
1) Over Recommended Operating Conditions.
2) Values apply to each I/O pin in input mode and to each input pin.
3) Values apply to each output pin and to each I/O pin in output mode.

D

D.5 Package and Pin Electrical Characteristics Model

The electrical model of the effects of package and pin parasitic effects on the 660 Bridge is shown in Figure D-1. The corresponding ranges of values shown in Table D-6 are nominal only, are not guaranteed, and vary somewhat from pin to pin. The lower values are typical of pins that are located on the side of the package and which are closest to the chip. The higher values are typical of corner pins. Note that the C1 capacitance is the value shown for DPC (Die Pad Capacitance) in Figure D-1 (which is due to the I/O book). C2 represents a distributed capacitance, and L1 represents a lumped loop inductance which includes the effects of inductance from the driver book to the power supply pins.

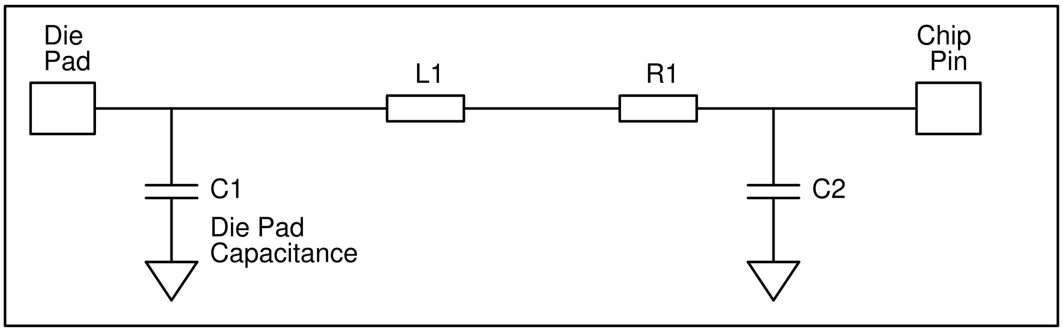


Figure D-1. 653 Package/Pin Electrical Model

Table D-6. Common Characteristics

Symbol	Condition	663	664	Unit
Inductance (L1)	Shortest lead	12	11	nH
	Longest lead	17	14	nH
Resistance (R1)	Shortest lead	.17	.13	Ω
	Longest lead	.21	.17	Ω
Capacitance (C2)	Shortest lead	1.4	1.2	pf
	Longest lead	2.4	1.9	pf

D

D.6 663 Buffer DC Characteristics By Signal

Table D-7. 663 Buffer DC Characteristics By Signal (See Note 1)

Signal	Pin	I/O	Book, Pad (2)	Type (3)	P/L (2)	I_{OL} (mA)	I_{OH} (mA)	DPC (4)	
								Min	Max
AOS_RR_MMRS	176	I	CBJE, B0	TTL	F	—	—	.8	1.0
C2P_WRL_OPEN	150	I	CBJE, B0	TTL	F	—	—	.8	1.0
CPU_CLK	144	I	CBB4, B0	TTL	A	—	—	.8	1.0
CPU_DATA[00:63]	—	I/O	CBUM, 10	PCI	B	6	6	3.1	3.7
CPU_DATA_OE#	138	I	CBJE, B0	TTL	F	—	—	.8	1.0
CPU_DPAR[0:7]	126	I/O	CBUM, 10	PCI	B	6	6	3.1	3.7
CPU_PAR_ERR#	178	O	CBNX, 16	TTL	C	14	12	2.0	2.2
CPU_RDL_OPEN	141	I	CBUM, 10	PCI	B	—	—	3.1	3.7
CRS_C2PWXS	147	I	CBUM, 10	PCI	B	—	—	3.1	3.7
DUAL_CTRL_REF	175	I	CBJE, B0	TTL	F	—	—	.8	1.0
ECC_LE_SEL	145	I	CBJE, BO	TTL	F	—	—	.8	1.0
MEM_BE[0:1]	—	I	CBSX, B0	TTL	A	—	—	.8	1.0
MEM_BE[2:3]	—	I	CBJE, B0	TTL	F	—	—	.8	1.0
MEM_CHECK[0:7]	—	I/O	CBNU, 10	TTL	B	12	8	3.7	4.3
MEM_DATA[63:0]	—	I/O	CBNU, 10	TTL	B	12	8	3.7	4.3
MEM_DATA_OE#	137	I	CBJE, B0	TTL	F	—	—	.8	1.0
MEM_ERR#	179	O	CBNX, 16	TTL	C	14	12	2.0	2.2
MEM_RD_SMPL	139	I	CBUM, 10	PCI	B	—	—	3.1	3.7
MEM_WRL_OPEN	146	I	CBUM, 10	PCI	B	—	—	3.1	3.7
MIO_TEST	70	I	CBJE, B0	TTL	F	—	—	.8	1.0
MWS_P2MRXS	148	I	CBUM, 10	PCI	B	—	—	3.1	3.7
PCI_AD[31:0]	—	I/O	CBUM, 10	PCI	A	6	6	3.1	3.7
PCI_AD_OE#	136	I	CBJE, B0	TTL	F	—	—	.8	1.0
PCI_EXT_SEL	149	I	CBUM, 10	PCI	B	—	—	3.1	3.7
PCI_IRDY#	177	I	CBUM, 10	PCI	B	—	—	3.1	3.7
PCI_OL_OPEN	151	I	CBJE, B0	TTL	F	—	—	.8	1.0
PCI_OUT_SEL	153	I	CBJE, BO	TTL	F	—	—	.8	1.0
RESERVED1	111	—	—	—	—	—	—	—	—
RESERVED2	231	—	CBJE, B0	TTL	F	—	—	.8	1.0
ROM_LOAD	140	I	CBUM, 10	PCI	B	—	—	3.1	3.7
SBE#	180	O	CBNX, 16	TTL	C	14	12	2.0	2.2
TEST#	152	I	CBSW, B3	TTL	A	—	—	.8	1.0

D

Notes for Table D–6:

1) Values apply over recommended operating conditions.

2) The Book, Pad, and P/L (performance level) define the I/O pin driver/receiver type, characteristics, and speed. More information on these items is contained in the *IBM CMOS4LP Logic Products Databook (8/93)*, Document Number ADCC4LDBU–01.

3) See Section D.4, Common Characteristics.

4) Die Pad Capacitance. The equivalent capacitance to ground of the die pad attachment as a function of the I/O book circuitry. To model the electrical path from the I/O book to the circuit board pad, see Section D.5.

D.7 664 Controller DC Characteristics By Signal

Table D-8. 664 Controller DC Characteristics By Signal (See Note 1)

Signal	Pin	I/O	Book, Pad (2)	Type (3)	P/L (2)	I_{OL} (mA)	I_{OH} (mA)	DPC (4) Min	DPC (4) Max
AACK#	109	I/O	CBUM, 10	PCI	C	6	6	3.1	3.7
AOS_RR_MMRS	69	O	CBNX, 16	TTL	C	14	12	2.0	2.2
ARTRY#	110	I/O	CBUM, 10	PCI	C	6	6	3.1	3.7
C2P_WRL_OPEN	61	O	CBNX, 16	TTL	C	14	12	2.0	2.2
CAS[7:0]#	—	O	CBNQ, 13	TTL	C	12	8	3.5	4.2
CPU_ADDR[0:31]	—	I/O	CBNZ, 11	TTL	B	14	12	2.0	2.2
CPU_BUS_CLAIM#	132	I	CBUM, 10	PCI	B	—	—	3.5	3.7
CPU_CLK	121	I	CBB4, B0	TTL	A	—	—	.8	1.0
CPU_DATA_OE#	197	O	CBNX, 16	TTL	C	14	12	2.0	2.2
CPU_GNT1#	134	O	CBUI, 13	PCI	C	6	6	2.8	3.5
CPU_GNT2#	135	O	CBUI, 13	PCI	C	6	6	2.8	3.5
CPU_PAR_ERR#	192	I	CBJE, BO	TTL	F	—	—	.8	1.0
CPU_RDL_OPEN	50	O	CBNX, 16	TTL	C	14	12	2.0	2.2
CPU_REQ1#	127	I	CBJE, B0	TTL	F	—	—	.8	1.0
CPU_REQ2#	128	I	CBJE, B0	TTL	F	—	—	.8	1.0
CRS_C2PWXS	65	O	CBNZ, 11	TTL	C	14	12	2.0	2.2
DPE#	133	I	CBJE, B0	TTL	F	—	—	.8	1.0
DUAL_CTRL_REF	205	O	CBNX, 16	TTL	B	14	12	2.0	2.2
ECC_LE_SEL	2	O	CBNX, 16	TTL	C	14	12	2.0	2.2
GBL#	120	O	CBNX, 11	TTL	B	14	12	2.0	2.2
IGN_PCI_AD31	57	I	CBJE, B0	TTL	F	—	—	.8	1.0
INT_CPU#	139	O	CBNQ, 15	TTL	C	4	4	2.9	3.5
INT_REQ	55	I	CBUM, 10	PCI	B	—	—	3.1	3.7
MA[11:0]	—	O	CBNQ, 13	TTL	C	12	8	3.5	4.2
MCP#	138	O	CBNX, 16	TTL	C	14	12	2.0	2.2
MEM_BE[3:0]	206	O	CBNX, 16	TTL	C	14	12	2.0	2.2
MEM_DATA_OE#	196	O	CBNX, 16	TTL	C	14	12	2.0	2.2
MEM_ERR#	194	I	CBJE, B0	TTL	F	—	—	.8	1.0
MEM_RD_SMPL	49	O	CBNX, 16	TTL	C	14	12	2.0	2.2
MEM_WRL_OPEN	51	O	CBNX, 16	TTL	C	14	12	2.0	2.2
MIO_TEST	154	I	CBJE, B0	TTL	F	—	—	.8	1.0
MWS_P2MRXS	66	O	CBNZ, 11	TTL	C	14	12	2.0	2.2
NMI_REQ	56	I	CBSX, B0	TTL	A	—	—	.8	1.0
PCI_AD[31:0]		I/O	CBUM, 10	PCI	B	6	6	3.1	3.7
PCI_AD_OE#	195	O	CBNX, 16	TTL	C	14	12	2.0	2.2
PCI_C/BE[3:0]#	—	I/O	CBUM, 10	PCI	B	6	6	3.1	3.7
PCI_CLK	123	I	CBJE, B0	PCI	F	—	—	.8	1.0
PCI_DEVSEL#	204	I/O	CBUM, 10	PCI	B	6	6	3.1	3.7

D

Table D-8. 664 Controller DC Characteristics By Signal (See Note 1) (Continued)

Signal	Pin	I/O	Book, Pad (2)	Type (3)	P/L (2)	I_{OL} (mA)	I_{OH} (mA)	DPC (4) Min	DPC (4) Max
PCI_EXT_SEL	67	O	CBNX, 16	TTL	C	14	12	2.0	2.2
PCI_FRAME#	200	I/O	CBUM, 10	PCI	B	6	6	3.1	3.7
PCI_GNT#	54	I	CBUM, 10	PCI	B	—	—	3.1	3.7
PCI_IRDY#	201	I/O	CBUM, 10	PCI	B	6	6	3.1	3.7
PCI_LOCK#	53	I	CBUM, 10	PCI	B	—	—	3.1	3.7
PCI_OL_OPEN	64	O	CBNX, 16	TTL	C	14	12	2.0	2.2
PCI_OUT_SEL	68	O	CBNX, 16	TTL	C	14	12	2.0	2.2
PCI_PAR	7	I/O	CBUM, 10	PCI	B	6	6	3.1	3.7
PCI_PERR#	10	I/O	CBUM, 10	PCI	B	6	6	3.1	3.7
PCI_REQ#	58	O	CBUI, 13	PCI	B	6	6	2.8	3.5
PCI_SERR#	71	O	CBUM, 10	PCI	B	6	6	3.1	3.7
PCI_STOP#	203	I/O	CBUM, 10	PCI	B	6	6	3.1	3.7
PCI_TRDY#	202	I/O	CBUM, 10	PCI	B	6	6	3.1	3.7
RAS[7:0]#	—	O	CBNQ, 13	TTL	C	12	8	3.5	4.1
RESERVED	140	I	CBJE, B0	TTL	F	—	—	.8	1.0
RESET#	156	I	CBJE, B0	TTL	F	—	—	.8	1.0
ROM_LOAD	70	O	CBNX, 16	TTL	C	14	12	2.0	2.2
ROM_OE#	47	O	CBNQ, 15	TTL	C	4	4	2.9	3.5
ROM_WE#	60	O	CBNQ, 15	TTL	C	4	4	2.9	3.5
SBE#	193	I	CBJE, B0	TTL	F	—	—	.8	1.0
SHD#	141	O	CBNX, 11	TTL	B	14	12	2.0	2.2
SRAM_ADS#/ADDR0	124	O	CBNX, 16	TTL	C	14	12	2.0	2.2
SRAM_ALE	119	O	CBNX, 16	TTL	C	14	12	2.0	2.2
SRAM_CNT_EN#/ADDR1	125	O	CBNX, 16	TTL	C	14	12	2.0	2.2
SRAM_OE#	117	O	CBNX, 16	TTL	C	14	12	2.0	2.2
SRAM_WE#	118	O	CBNX, 16	TTL	C	14	12	2.0	2.2
STOP_CLK_EN#	151	I	CBSZ, BO	TTL		—	—	.8	1.0
TA#	111	I/O	CBUM, 10	PCI	C	6	6	3.1	3.7
TAG_CLR#	116	O	CBNX, 17	TTL	C	12	10	1.6	1.8
TAG_MATCH	142	I	CBUM, 10	PCI	B	—	—	3.1	3.7
TAG_VALID	115	O	CBNX, 17	TTL	C	12	10	1.6	1.8
TAG_WE#	114	O	CBNX, 17	TTL	C	12	10	1.6	1.8
TBST#	144	I/O	CBNZ, 11	TTL	B	14	12	2.0	2.2
TEA#	137	O	CBUM, 10	PCI	C	6	6	3.1	3.7
TEST#	155	I	CBSW, B3	TTL	A	—	—	.8	1.0
TS#	143	I/O	CBUM, 10	PCI	C	6	6	3.1	3.7
TSIZE[0:2]	—	I/O	CBNZ, 11	TTL	B	14	12	2.0	2.2
TT[0:4]	—	I/O	CBNZ, 11	TTL	B	14	12	2.0	2.2

D

Table D-8. 664 Controller DC Characteristics By Signal (See Note 1) (Continued)

Signal	Pin	I/O	Book, Pad (2)	Type (3)	P/L (2)	I_{OL} (mA)	I_{OH} (mA)	DPC (4) Min	DPC (4) Max
WE[1:0]#	—	O	CBNQ, 13	TTL	C	12	8	3.5	4.1
XATS#	129	I	CBJE, B0	TTL	F	—	—	.8	1.0

Notes:

1) Values apply over recommended operating conditions.

2) The Book, Pad, and P/L (performance level) define the I/O pin driver/receiver type, characteristics, and speed. More information on these items is contained in the *IBM CMOS4LP Logic Products Databook (8/93)*, Document Number ADCC4LDBU–01.

3) See Section D.4, Common Characteristics.

4) Die Pad Capacitance. The equivalent capacitance to ground of the die pad attachment as a function of the I/O book circuitry. To model the electrical path from the I/O book to the circuit board pad, see Section D.5.

D

D

D.8 Output VI Curves

D.8.1 PCI Local Bus Compatible Drivers

Inputs and outputs (drives) in the PCI group have input/output characteristics that comply with the DC specifications of both the 3.3v and 5v signalling environments defined for PCI Local Bus components in the *PCI Local Bus Specification, Revision 2.0*. These signals are identified in the DC Characteristics Tables as type PCI.

The following tables[1] show V–I curves for the PCI bus output drivers contained in the 650 Bridge chipset. The curves labeled WC show driver characteristics of a worst case process variation device operating at 85 deg C with V_{DD} = 3v. The curves labeled BC show driver characteristics of a best case process variation device operating at 10 deg C with V_{DD} = 3.6v. The curves labeled NOM show driver characteristics of a nominal process device operating at 25 deg C with V_{DD} = 3.3v.

[1]**Note:** Due to publication problems, the VI curves, along with the example system implementation, have been moved to a separate document attached to the end of this one (660_bridge_VI_curv).

D.9 Package Drawings

The following package drawings of the 663 Buffer chip and the 664 Controller chip (Figure D-2 through Figure D-5) show the packages that are used for the initial samples of the chips and the packages that are used for production chips. The differences between the initial sample and production chips lies in the area labelled as *Detail A* within the drawings.

D.9.1 663 Buffer Initial Sample Package Drawing

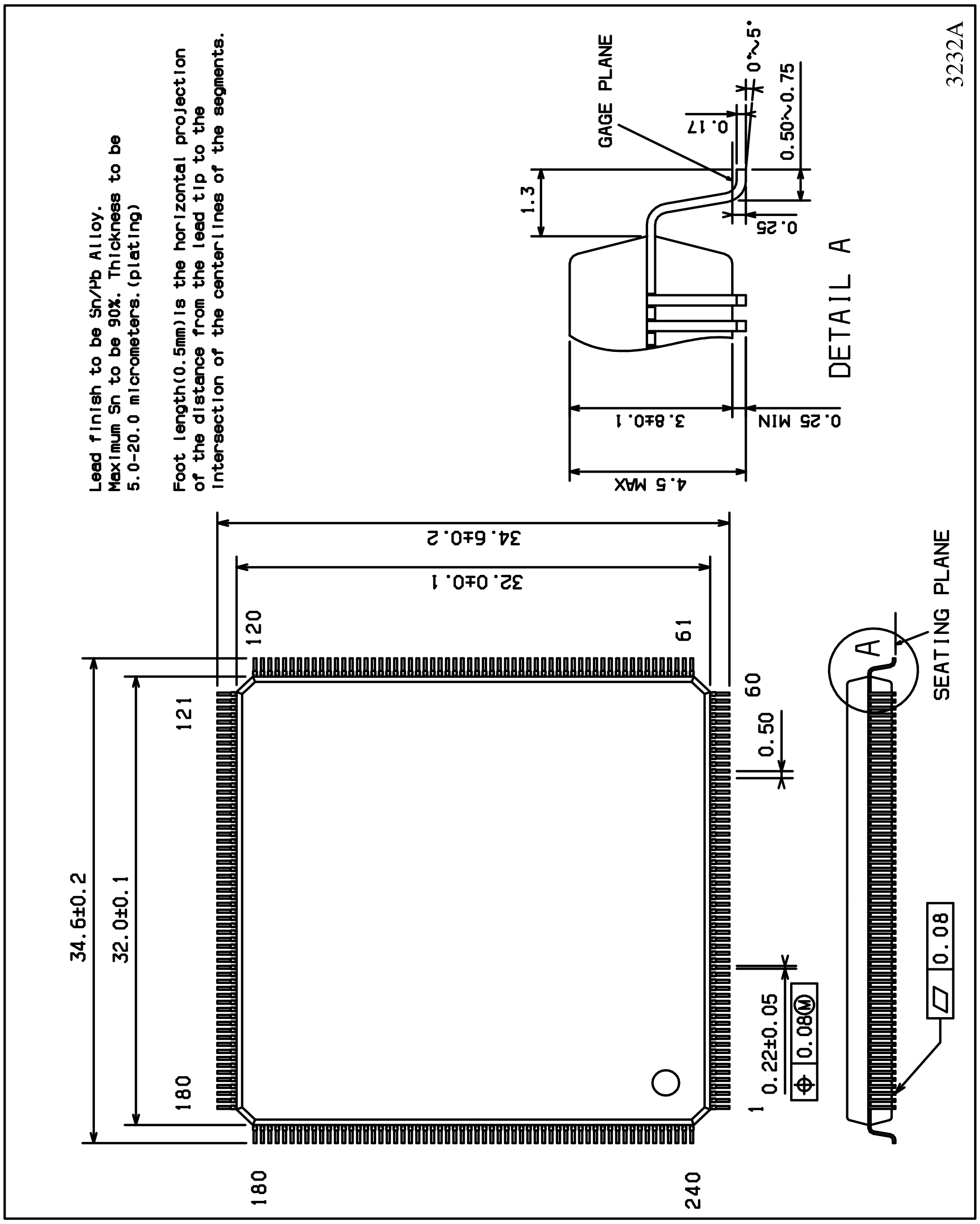


Figure D-2. 663 Buffer Initial Sample Package Drawing

D.9.2 663 Buffer Production Package Drawing

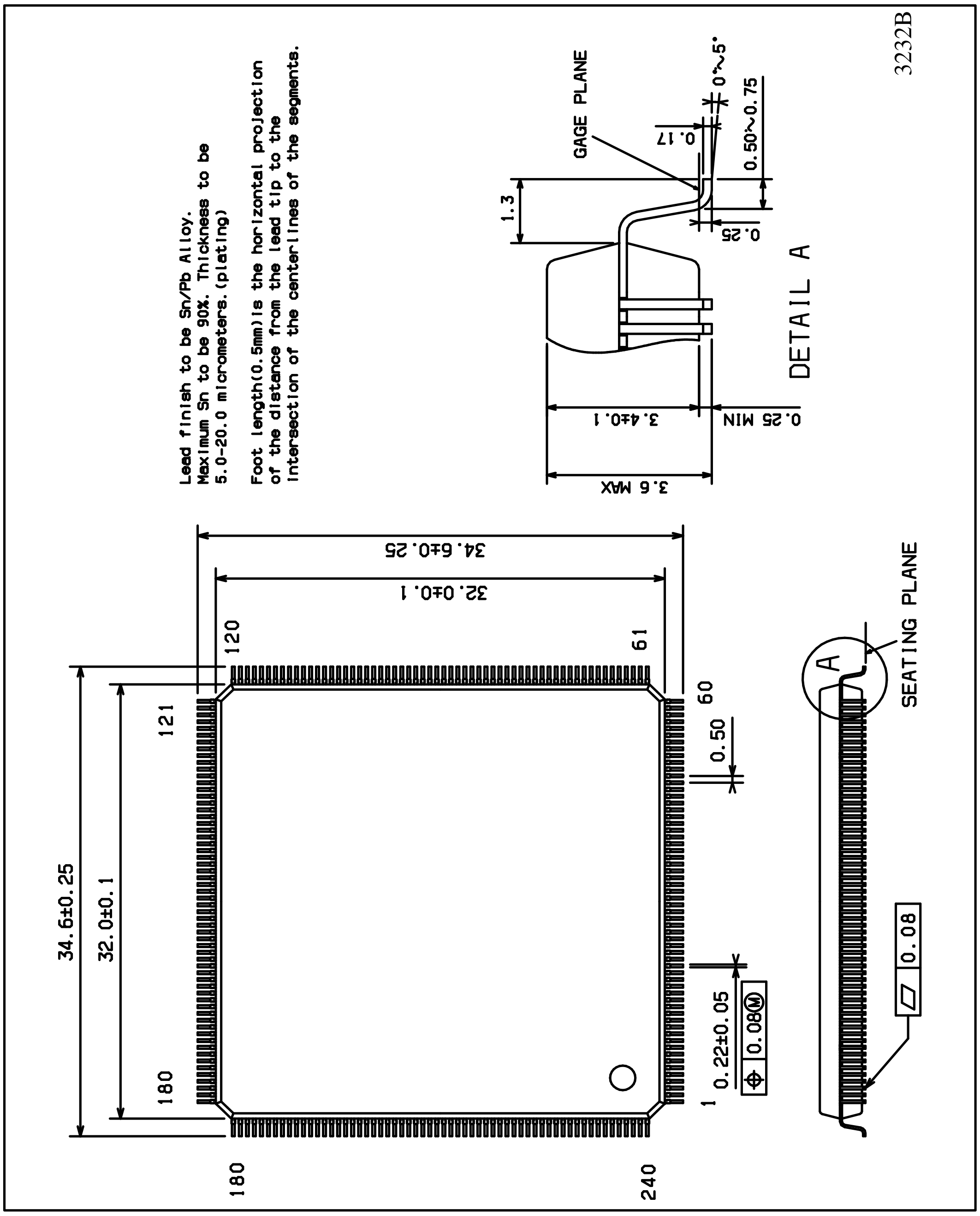


Figure D-3. 663 Buffer Production Package Drawing

D.9.3 664 Controller Initial Sample Package Drawing

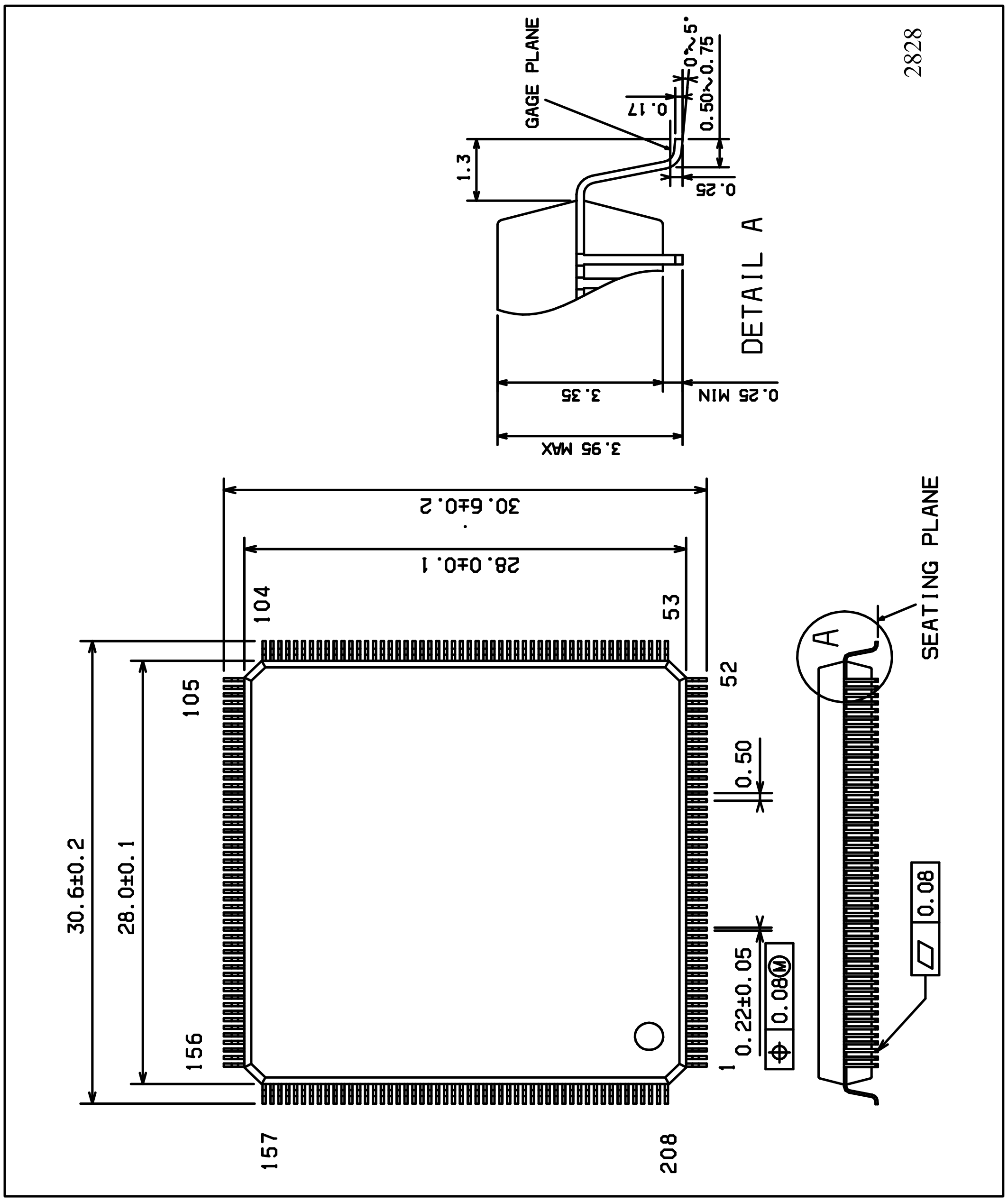


Figure D-4. 664 Controller Initial Sample Package Drawing

D.9.4 664 Controller Production Package Drawing

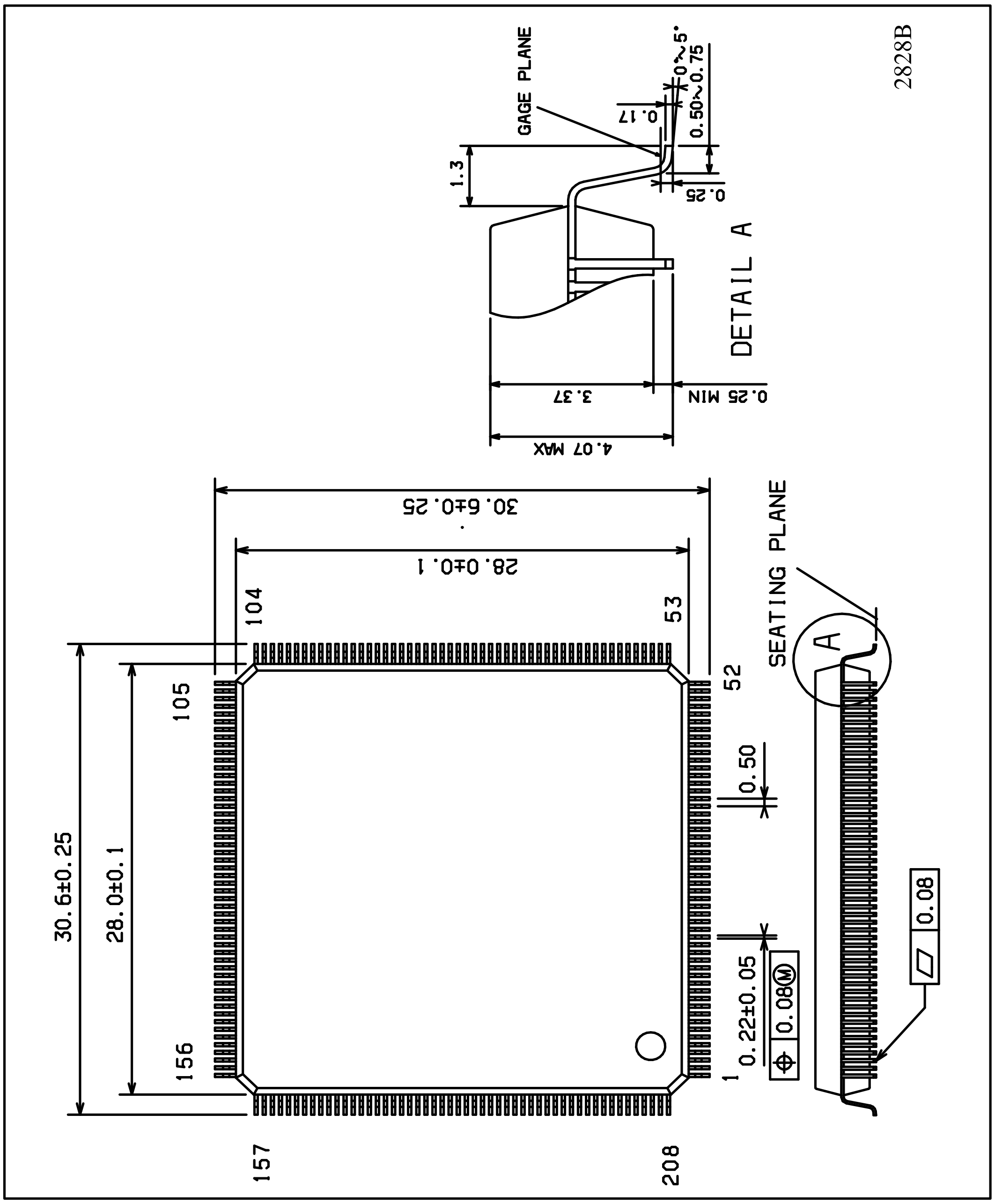


Figure D-5. 664 Controller Production Package Drawing

Appendix E
Alphabetic and Numeric Pin Lists

Appendix E contains alphabetic pin lists and numeric pin lists for the 663 Buffer and the 664 Controller.

E.1 663 Buffer Pin Lists

E.1.1 663 Buffer Alphabetic Pin List

Table E-1. 663 Pin List

663 Signal Name	Pin #
AOS_RR_MMRS	176
C2P_WRL_OPEN	150
CPU_CLK	144
CPU_DATA[00]	49
CPU_DATA[01]	50
CPU_DATA[02]	51
CPU_DATA[03]	54
CPU_DATA[04]	55
CPU_DATA[05]	56
CPU_DATA[06]	57
CPU_DATA[07]	58
CPU_DATA[08]	59
CPU_DATA[09]	60
CPU_DATA[10]	94
CPU_DATA[11]	95
CPU_DATA[12]	98
CPU_DATA[13]	99
CPU_DATA[14]	100
CPU_DATA[15]	101
CPU_DATA[16]	102
CPU_DATA[17]	103
CPU_DATA[18]	104
CPU_DATA[19]	105
CPU_DATA[20]	106
CPU_DATA[21]	107
CPU_DATA[22]	108
CPU_DATA[23]	109
CPU_DATA[24]	112
CPU_DATA[25]	113
CPU_DATA[26]	114
CPU_DATA[27]	115
CPU_DATA[28]	116
CPU_DATA[29]	117
CPU_DATA[30]	118
CPU_DATA[31]	119
CPU_DATA[32]	62

E

663 Signal Name	Pin #
CPU_DATA[33]	63
CPU_DATA[34]	64
CPU_DATA[35]	65
CPU_DATA[36]	66
CPU_DATA[37]	67
CPU_DATA[38]	68
CPU_DATA[39]	69
CPU_DATA[40]	72
CPU_DATA[41]	73
CPU_DATA[42]	74
CPU_DATA[43]	75
CPU_DATA[44]	76
CPU_DATA[45]	77
CPU_DATA[46]	78
CPU_DATA[47]	79
CPU_DATA[48]	80
CPU_DATA[49]	81
CPU_DATA[50]	82
CPU_DATA[51]	83
CPU_DATA[52]	86
CPU_DATA[53]	87
CPU_DATA[54]	88
CPU_DATA[55]	89
CPU_DATA[56]	91
CPU_DATA[57]	92
CPU_DATA[58]	93
CPU_DATA[59]	121
CPU_DATA[60]	122
CPU_DATA[61]	123
CPU_DATA[62]	124
CPU_DATA[63]	125
CPU_DATA_OE#	138

663 Signal Name	Pin #
CPU_DPAR[0]	126
CPU_DPAR[1]	127
CPU_DPAR[2]	130
CPU_DPAR[3]	131
CPU_DPAR[4]	132
CPU_DPAR[5]	133
CPU_DPAR[6]	134
CPU_DPAR[7]	135
CPU_PAR_ERR#	178
CPU_RDL_OPEN	141
CRS_C2PWXS	147
DUAL_CTRL_REF	175
ECC_LE_SEL	145
GND[01]	9
GND[02]	23
GND[03]	39
GND[04]	53
GND[05]	71
GND[06]	85
GND[07]	90
GND[08]	97
GND[09]	120
GND[10]	129
GND[11]	143
GND[12]	159
GND[13]	173
GND[14]	190
GND[15]	191
GND[16]	205
GND[17]	217
GND[18]	240
MEM_BE[0]	154

E

663 Signal Name	Pin #
MEM_BE[1]	155
MEM_BE[2]	156
MEM_BE[3]	157
MEM_CHECK[0]	165
MEM_CHECK[1]	166
MEM_CHECK[2]	167
MEM_CHECK[3]	168
MEM_CHECK[4]	169
MEM_CHECK[5]	170
MEM_CHECK[6]	171
MEM_CHECK[7]	174
MEM_DATA[00]	1
MEM_DATA[01]	2
MEM_DATA[02]	3
MEM_DATA[03]	4
MEM_DATA[04]	5
MEM_DATA[05]	6
MEM_DATA[06]	7
MEM_DATA[07]	10
MEM_DATA[08]	11
MEM_DATA[09]	12
MEM_DATA[10]	182
MEM_DATA[11]	183
MEM_DATA[12]	184
MEM_DATA[13]	185
MEM_DATA[14]	186
MEM_DATA[15]	187
MEM_DATA[16]	188
MEM_DATA[17]	189
MEM_DATA[18]	192
MEM_DATA[19]	193
MEM_DATA[20]	194

663 Signal Name	Pin #
MEM_DATA[21]	195
MEM_DATA[22]	196
MEM_DATA[23]	197
MEM_DATA[24]	198
MEM_DATA[25]	199
MEM_DATA[26]	200
MEM_DATA[27]	201
MEM_DATA[28]	202
MEM_DATA[29]	203
MEM_DATA[30]	206
MEM_DATA[31]	207
MEM_DATA[32]	208
MEM_DATA[33]	209
MEM_DATA[34]	211
MEM_DATA[35]	212
MEM_DATA[36]	213
MEM_DATA[37]	214
MEM_DATA[38]	215
MEM_DATA[39]	218
MEM_DATA[40]	219
MEM_DATA[41]	220
MEM_DATA[42]	221
MEM_DATA[43]	222
MEM_DATA[44]	223
MEM_DATA[45]	224
MEM_DATA[46]	225
MEM_DATA[47]	226
MEM_DATA[48]	227
MEM_DATA[49]	228
MEM_DATA[50]	229
MEM_DATA[51]	232
MEM_DATA[52]	233

663 Signal Name	Pin #
MEM_DATA[53]	234
MEM_DATA[54]	235
MEM_DATA[55]	236
MEM_DATA[56]	237
MEM_DATA[57]	238
MEM_DATA[58]	239
MEM_DATA[59]	160
MEM_DATA[60]	161
MEM_DATA[61]	162
MEM_DATA[62]	163
MEM_DATA[63]	164
MEM_DATA_OE#	137
MEM_ERR#	179
MEM_RD_SMPL	139
MEM_WRL_OPEN	146
MIO_TEST	70
MWS_P2MRXS	148
PCI_AD[00]	13
PCI_AD[01]	14
PCI_AD[02]	15
PCI_AD[03]	16
PCI_AD[04]	17
PCI_AD[05]	18
PCI_AD[06]	19
PCI_AD[07]	20
PCI_AD[08]	21
PCI_AD[09]	24
PCI_AD[10]	25
PCI_AD[11]	26
PCI_AD[12]	27
PCI_AD[13]	28
PCI_AD[14]	29

663 Signal Name	Pin #
PCI_AD[15]	30
PCI_AD[16]	31
PCI_AD[17]	32
PCI_AD[18]	33
PCI_AD[19]	34
PCI_AD[20]	35
PCI_AD[21]	36
PCI_AD[22]	37
PCI_AD[23]	40
PCI_AD[24]	41
PCI_AD[25]	42
PCI_AD[26]	43
PCI_AD[27]	44
PCI_AD[28]	45
PCI_AD[29]	46
PCI_AD[30]	47
PCI_AD[31]	48
PCI_AD_OE#	136
PCI_EXT_SEL	149
PCI_IRDY#	177
PCI_OL_OPEN	151
PCI_OUT_SEL	153
RESERVED1	111
RESERVED2	231
ROM_LOAD	140
SBE#	180
TEST#	152
VDD[01]	8
VDD[02]	22
VDD[03]	38
VDD[04]	52
VDD[05]	61

663 Signal Name	Pin #
VDD[06]	84
VDD[07]	96
VDD[08]	110
VDD[09]	128
VDD[10]	142
VDD[11]	158
VDD[12]	172
VDD[13]	181
VDD[14]	204
VDD[15]	210
VDD[16]	216
VDD[17]	230

E.1.2 663 Buffer Numeric Pin List

Table E-2. 663 Pin List

Pin #	663 Signal Name
1	MEM_DATA[00]
2	MEM_DATA[01]
3	MEM_DATA[02]
4	MEM_DATA[03]
5	MEM_DATA[04]
6	MEM_DATA[05]
7	MEM_DATA[06]
8	VDD[01]
9	GND[01]
10	MEM_DATA[07]
11	MEM_DATA[08]
12	MEM_DATA[09]
13	PCI_AD[00]
14	PCI_AD[01]
15	PCI_AD[02]
16	PCI_AD[03]
17	PCI_AD[04]
18	PCI_AD[05]
19	PCI_AD[06]
20	PCI_AD[07]
21	PCI_AD[08]
22	VDD[02]
23	GND[02]
24	PCI_AD[09]
25	PCI_AD[10]
26	PCI_AD[11]
27	PCI_AD[12]
28	PCI_AD[13]
29	PCI_AD[14]
30	PCI_AD[15]
31	PCI_AD[16]
32	PCI_AD[17]
33	PCI_AD[18]
34	PCI_AD[19]
35	PCI_AD[20]
36	PCI_AD[21]
37	PCI_AD[22]
38	VDD[03]
39	GND[03]
40	PCI_AD[23]
41	PCI_AD[24]
42	PCI_AD[25]
43	PCI_AD[26]
44	PCI_AD[27]
45	PCI_AD[28]
46	PCI_AD[29]
47	PCI_AD[30]
48	PCI_AD[31]

Pin #	663 Signal Name
49	CPU_DATA[00]
50	CPU_DATA[01]
51	CPU_DATA[02]
52	VDD[04]
53	GND[04]
54	CPU_DATA[03]
55	CPU_DATA[04]
56	CPU_DATA[05]
57	CPU_DATA[06]
58	CPU_DATA[07]
59	CPU_DATA[08]
60	CPU_DATA[09]
61	VDD[05]
62	CPU_DATA[32]
63	CPU_DATA[33]
64	CPU_DATA[34]
65	CPU_DATA[35]
66	CPU_DATA[36]
67	CPU_DATA[37]
68	CPU_DATA[38]
69	CPU_DATA[39]
70	MIO_TEST
71	GND[05]
72	CPU_DATA[40]
73	CPU_DATA[41]
74	CPU_DATA[42]
75	CPU_DATA[43]
76	CPU_DATA[44]
77	CPU_DATA[45]
78	CPU_DATA[46]
79	CPU_DATA[47]
80	CPU_DATA[48]

Pin #	663 Signal Name
81	CPU_DATA[49]
82	CPU_DATA[50]
83	CPU_DATA[51]
84	VDD[06]
85	GND[06]
86	CPU_DATA[52]
87	CPU_DATA[53]
88	CPU_DATA[54]
89	CPU_DATA[55]
90	GND[07]
91	CPU_DATA[56]
92	CPU_DATA[57]
93	CPU_DATA[58]
94	CPU_DATA[10]
95	CPU_DATA[11]
96	VDD[07]
97	GND[08]
98	CPU_DATA[12]
99	CPU_DATA[13]
100	CPU_DATA[14]
101	CPU_DATA[15]
102	CPU_DATA[16]
103	CPU_DATA[17]
104	CPU_DATA[18]
105	CPU_DATA[19]
106	CPU_DATA[20]
107	CPU_DATA[21]
108	CPU_DATA[22]
109	CPU_DATA[23]
110	VDD[08]
111	RESERVED1
112	CPU_DATA[24]

Pin #	663 Signal Name
113	CPU_DATA[25]
114	CPU_DATA[26]
115	CPU_DATA[27]
116	CPU_DATA[28]
117	CPU_DATA[29]
118	CPU_DATA[30]
119	CPU_DATA[31]
120	GND[09]
121	CPU_DATA[59]
122	CPU_DATA[60]
123	CPU_DATA[61]
124	CPU_DATA[62]
125	CPU_DATA[63]
126	CPU_DPAR[0]
127	CPU_DPAR[1]
128	VDD[09]
129	GND[10]
130	CPU_DPAR[2]
131	CPU_DPAR[3]
132	CPU_DPAR[4]
133	CPU_DPAR[5]
134	CPU_DPAR[6]
135	CPU_DPAR[7]
136	PCI_AD_OE#
137	MEM_DATA_OE#
138	CPU_DATA_OE#
139	MEM_RD_SMPL
140	ROM_LOAD
141	CPU_RDL_OPEN
142	VDD[10]
143	GND[11]
144	CPU_CLK

Pin #	663 Signal Name
145	ECC_LE_SEL
146	MEM_WRL_OPEN
147	CRS_C2PWXS
148	MWS_P2MRXS
149	PCI_EXT_SEL
150	C2P_WRL_OPEN
151	PCI_OL_OPEN
152	TEST#
153	PCI_OUT_SEL
154	MEM_BE[0]
155	MEM_BE[1]
156	MEM_BE[2]
157	MEM_BE[3]
158	VDD[11]
159	GND[12]
160	MEM_DATA[59]
161	MEM_DATA[60]
162	MEM_DATA[61]
163	MEM_DATA[62]
164	MEM_DATA[63]
165	MEM_CHECK[0]
166	MEM_CHECK[1]
167	MEM_CHECK[2]
168	MEM_CHECK[3]
169	MEM_CHECK[4]
170	MEM_CHECK[5]
171	MEM_CHECK[6]
172	VDD[12]
173	GND[13]
174	MEM_CHECK[7]
175	DUAL_CTRL_REF
176	AOS_RR_MMRS

Pin #	663 Signal Name
177	PCI_IRDY#
178	CPU_PAR_ERR#
179	MEM_ERR#
180	SBE#
181	VDD[13]
182	MEM_DATA[10]
183	MEM_DATA[11]
184	MEM_DATA[12]
185	MEM_DATA[13]
186	MEM_DATA[14]
187	MEM_DATA[15]
188	MEM_DATA[16]
189	MEM_DATA[17]
190	GND[14]
191	GND[15]
192	MEM_DATA[18]
193	MEM_DATA[19]
194	MEM_DATA[20]
195	MEM_DATA[21]
196	MEM_DATA[22]
197	MEM_DATA[23]
198	MEM_DATA[24]
199	MEM_DATA[25]
200	MEM_DATA[26]
201	MEM_DATA[27]
202	MEM_DATA[28]
203	MEM_DATA[29]
204	VDD[14]
205	GND[16]
206	MEM_DATA[30]
207	MEM_DATA[31]
208	MEM_DATA[32]

Pin #	663 Signal Name
209	MEM_DATA[33]
210	VDD[15]
211	MEM_DATA[34]
212	MEM_DATA[35]
213	MEM_DATA[36]
214	MEM_DATA[37]
215	MEM_DATA[38]
216	VDD[16]
217	GND[17]
218	MEM_DATA[39]
219	MEM_DATA[40]
220	MEM_DATA[41]
221	MEM_DATA[42]
222	MEM_DATA[43]
223	MEM_DATA[44]
224	MEM_DATA[45]
225	MEM_DATA[46]
226	MEM_DATA[47]
227	MEM_DATA[48]
228	MEM_DATA[49]
229	MEM_DATA[50]
230	VDD[17]
231	RESERVED2
232	MEM_DATA[51]
233	MEM_DATA[52]
234	MEM_DATA[53]
235	MEM_DATA[54]
236	MEM_DATA[55]
237	MEM_DATA[56]
238	MEM_DATA[57]
239	MEM_DATA[58]
240	GND[18]

E.2 664 Controller Pin Lists

E.2.1 664 Controller Alphabetic Pins

Table E-3. 664 Pin List

664 Signal Name	Pin #
AACK#	109
ADDR[00]	72
ADDR[01]	73
ADDR[02]	74
ADDR[03]	75
ADDR[04]	76
ADDR[05]	77
ADDR[06]	80
ADDR[07]	81
ADDR[08]	82
ADDR[09]	83
ADDR[10]	84
ADDR[11]	85
ADDR[12]	86
ADDR[13]	87
ADDR[14]	89
ADDR[15]	90
ADDR[16]	91
ADDR[17]	92
ADDR[18]	93
ADDR[19]	96
ADDR[20]	97
ADDR[21]	98
ADDR[22]	99
ADDR[23]	100
ADDR[24]	101
ADDR[25]	102
ADDR[26]	103
ADDR[27]	104
ADDR[28]	105
ADDR[29]	106
ADDR[30]	107
ADDR[31]	108
AOS_RR_MMRS	69
ARTRY#	110
C2P_WRL_OPEN	61
CAS[0]#	174
CAS[1]#	173
CAS[2]#	172
CAS[3]#	171
CAS[4]#	170
CAS[5]#	169
CAS[6]#	168
CAS[7]#	165
CPU_BUS_CLAIM#	132
CPU_CLK	121
CPU_DATA_OE#	197
CPU_GNT1#	134
CPU_GNT2#	135
CPU_PAR_ERR#	192
CPU_RDL_OPEN	50
CPU_REQ1#	127
CPU_REQ2#	128
CRS_C2PWXS	65
DPE#	133
DUAL_CTRL_REF	205
ECC_LE_SEL	2
GBL#	120
GND[01]	9
GND[02]	17
GND[03]	27

E

664 Signal Name	Pin #
GND[04]	45
GND[05]	52
GND[06]	63
GND[07]	79
GND[08]	88
GND[09]	95
GND[10]	113
GND[11]	131
GND[12]	149
GND[13]	167
GND[14]	183
GND[15]	199
IGN_PCI_AD31	57
INT_CPU#	139
INT_REQ	55
MA[00]	190
MA[01]	189
MA[02]	188
MA[03]	187
MA[04]	186
MA[05]	185
MA[06]	184
MA[07]	181
MA[08]	180
MA[09]	179
MA[10]	178
MA[11]	177
MCP#	138
MEM_BE[0]	206
MEM_BE[1]	207
MEM_BE[2]	208
MEM_BE[3]	1

664 Signal Name	Pin #
MEM_DATA_OE#	196
MEM_ERR#	194
MEM_RD_SMPL	49
MEM_WRL_OPEN	51
MIO_TEST	154
MWS_P2MRXS	66
NMI_REQ	56
PCI_AD[00]	48
PCI_AD[01]	59
PCI_AD[02]	46
PCI_AD[03]	43
PCI_AD[04]	42
PCI_AD[05]	41
PCI_AD[06]	40
PCI_AD[07]	39
PCI_AD[08]	38
PCI_AD[09]	37
PCI_AD[10]	36
PCI_AD[11]	35
PCI_AD[12]	34
PCI_AD[13]	33
PCI_AD[14]	32
PCI_AD[15]	31
PCI_AD[16]	30
PCI_AD[17]	29
PCI_AD[18]	28
PCI_AD[19]	25
PCI_AD[20]	24
PCI_AD[21]	23
PCI_AD[22]	22
PCI_AD[23]	21
PCI_AD[24]	20

E

664 Signal Name	Pin #
PCI_AD[25]	19
PCI_AD[26]	18
PCI_AD[27]	15
PCI_AD[28]	14
PCI_AD[29]	13
PCI_AD[30]	12
PCI_AD[31]	11
PCI_AD_OE#	195
PCI_C/BE[0]#	6
PCI_C/BE[1]#	5
PCI_C/BE[2]#	4
PCI_C/BE[3]#	3
PCI_CLK	123
PCI_DEVSEL#	204
PCI_EXT_SEL	67
PCI_FRAME#	200
PCI_GNT#	54
PCI_IRDY#	201
PCI_LOCK#	53
PCI_OL_OPEN	64
PCI_OUT_SEL	68
PCI_PAR	7
PCI_PERR#	10
PCI_REQ#	58
PCI_SERR#	71
PCI_STOP#	203
PCI_TRDY#	202
RAS[0]#	164
RAS[1]#	163
RAS[2]#	162
RAS[3]#	161
RAS[4]#	160

664 Signal Name	Pin #
RAS[5]#	159
RAS[6]#	158
RAS[7]#	157
RESERVED	140
RESET#	156
ROM_LOAD	70
ROM_OE#	47
ROM_WE#	60
SBE#	193
SHD#	141
SRAM_ADS#/ADDR0	124
SRAM_ALE	119
SRAM_CNT_EN#/ADDR1	125
SRAM_OE#	117
SRAM_WE#	118
STOP_CLK_EN#	151
TA#	111
TAG_CLR#	116
TAG_MATCH	142
TAG_VALID	115
TAG_WE#	114
TBST#	144
TEA#	137
TEST#	155
TS#	143
TSIZE[0]	145
TSIZE[1]	146
TSIZE[2]	147
TT[0]	150
TT[1]	152
TT[2]	153
TT[3]	126

664 Signal Name	Pin #
TT[4]	136
VDD[01]	8
VDD[02]	16
VDD[03]	26
VDD[04]	44
VDD[05]	62
VDD[06]	78
VDD[07]	94
VDD[08]	112
VDD[09]	122
VDD[10]	130
VDD[11]	148
VDD[12]	166
VDD[13]	182
VDD[14]	191
VDD[15]	198
WE[0]#	176
WE[1]#	175
XATS#	129

E.2.2 664 Controller Numeric Pins

Table E-4. 664 Pin List

Pin #	664 Signal Name
1	MEM_BE[3]
2	ECC_LE_SEL
3	PCI_C/BE[3]#
4	PCI_C/BE[2]#
5	PCI_C/BE[1]#
6	PCI_C/BE[0]#
7	PCI_PAR
8	VDD[01]
9	GND[01]
10	PCI_PERR#
11	PCI_AD[31]
12	PCI_AD[30]
13	PCI_AD[29]
14	PCI_AD[28]
15	PCI_AD[27]
16	VDD[02]
17	GND[02]
18	PCI_AD[26]
19	PCI_AD[25]
20	PCI_AD[24]
21	PCI_AD[23]
22	PCI_AD[22]
23	PCI_AD[21]
24	PCI_AD[20]
25	PCI_AD[19]
26	VDD[03]
27	GND[03]
28	PCI_AD[18]
29	PCI_AD[17]
30	PCI_AD[16]
31	PCI_AD[15]
32	PCI_AD[14]
33	PCI_AD[13]
34	PCI_AD[12]
35	PCI_AD[11]
36	PCI_AD[10]
37	PCI_AD[09]
38	PCI_AD[08]
39	PCI_AD[07]
40	PCI_AD[06]
41	PCI_AD[05]

E

Pin #	664 Signal Name
42	PCI_AD[04]
43	PCI_AD[03]
44	VDD[04]
45	GND[04]
46	PCI_AD[02]
47	ROM_OE#
48	PCI_AD[00]
49	MEM_RD_SMPL
50	CPU_RDL_OPEN
51	MEM_WRL_OPEN
52	GND[05]
53	PCI_LOCK#
54	PCI_GNT#
55	INT_REQ
56	NMI_REQ
57	IGN_PCI_AD31
58	PCI_REQ#
59	PCI_AD[01]
60	ROM_WE#
61	C2P_WRL_OPEN
62	VDD[05]
63	GND[06]
64	PCI_OL_OPEN
65	CRS_C2PWXS
66	MWS_P2MRXS
67	PCI_EXT_SEL
68	PCI_OUT_SEL
69	AOS_RR_MMRS
70	ROM_LOAD
71	PCI_SERR#
72	ADDR[00]
73	ADDR[01]

Pin #	664 Signal Name
74	ADDR[02]
75	ADDR[03]
76	ADDR[04]
77	ADDR[05]
78	VDD[06]
79	GND[07]
80	ADDR[06]
81	ADDR[07]
82	ADDR[08]
83	ADDR[09]
84	ADDR[10]
85	ADDR[11]
86	ADDR[12]
87	ADDR[13]
88	GND[08]
89	ADDR[14]
90	ADDR[15]
91	ADDR[16]
92	ADDR[17]
93	ADDR[18]
94	VDD[07]
95	GND[09]
96	ADDR[19]
97	ADDR[20]
98	ADDR[21]
99	ADDR[22]
100	ADDR[23]
101	ADDR[24]
102	ADDR[25]
103	ADDR[26]
104	ADDR[27]
105	ADDR[28]

E

Pin #	664 Signal Name
106	ADDR[29]
107	ADDR[30]
108	ADDR[31]
109	AACK#
110	ARTRY#
111	TA#
112	VDD[08]
113	GND[10]
114	TAG_WE#
115	TAG_VALID
116	TAG_CLR#
117	SRAM_OE#
118	SRAM_WE#
119	SRAM_ALE
120	GBL#
121	CPU_CLK
122	VDD[09]
123	PCI_CLK
124	SRAM_ADS#/ADDR0
125	SRAM_CNT_EN#/ADDR1
126	TT[3]
127	CPU_REQ1#
128	CPU_REQ2#
129	XATS#
130	VDD[10]
131	GND[11]
132	CPU_BUS_CLAIM#
133	DPE#
134	CPU_GNT1#
135	CPU_GNT2#
136	TT[4]
137	TEA_

Pin #	664 Signal Name
138	MCP#
139	INT_CPU#
140	RESERVED
141	SHD#
142	TAG_MATCH
143	TS#
144	TBST#
145	TSIZE[0]
146	TSIZE[1]
147	TSIZE[2]
148	VDD[11]
149	GND[12]
150	TT[0]
151	STOP_CLK_EN#
152	TT[1]
153	TT[2]
154	MIO_TEST
155	TEST#
156	RESET#
157	RAS[7]#
158	RAS[6]#
159	RAS[5]#
160	RAS[4]#
161	RAS[3]#
162	RAS[2]#
163	RAS[1]#
164	RAS[0]#
165	CAS[7]#
166	VDD[12]
167	GND[13]
168	CAS[6]#
169	CAS[5]#

E

Pin #	664 Signal Name
170	CAS[4]#
171	CAS[3]#
172	CAS[2]#
173	CAS[1]#
174	CAS[0]#
175	WE[1]#
176	WE[0]#
177	MA[11]
178	MA[10]
179	MA[09]
180	MA[08]
181	MA[07]
182	VDD[13]
183	GND[14]
184	MA[06]
185	MA[05]
186	MA[04]
187	MA[03]
188	MA[02]
189	MA[01]
190	MA[00]
191	VDD[14]
192	CPU_PAR_ERR#
193	SBE#
194	MEM_ERR#
195	PCI_AD_OE#
196	MEM_DATA_OE#
197	CPU_DATA_OE#
198	VDD[15]
199	GND[15]
200	PCI_FRAME#
201	PCI_IRDY#

Pin #	664 Signal Name
202	PCI_TRDY#
203	PCI_STOP#
204	PCI_DEVSEL#
205	DUAL_CTRL_REF
206	MEM_BE[0]
207	MEM_BE[1]
208	MEM_BE[2]

E

Appendix F
Glossary, Index, and Contacts

Glossary

TBD – To be determined. Sections marked TBD are placeholders for data that is not yet available.

Index

F

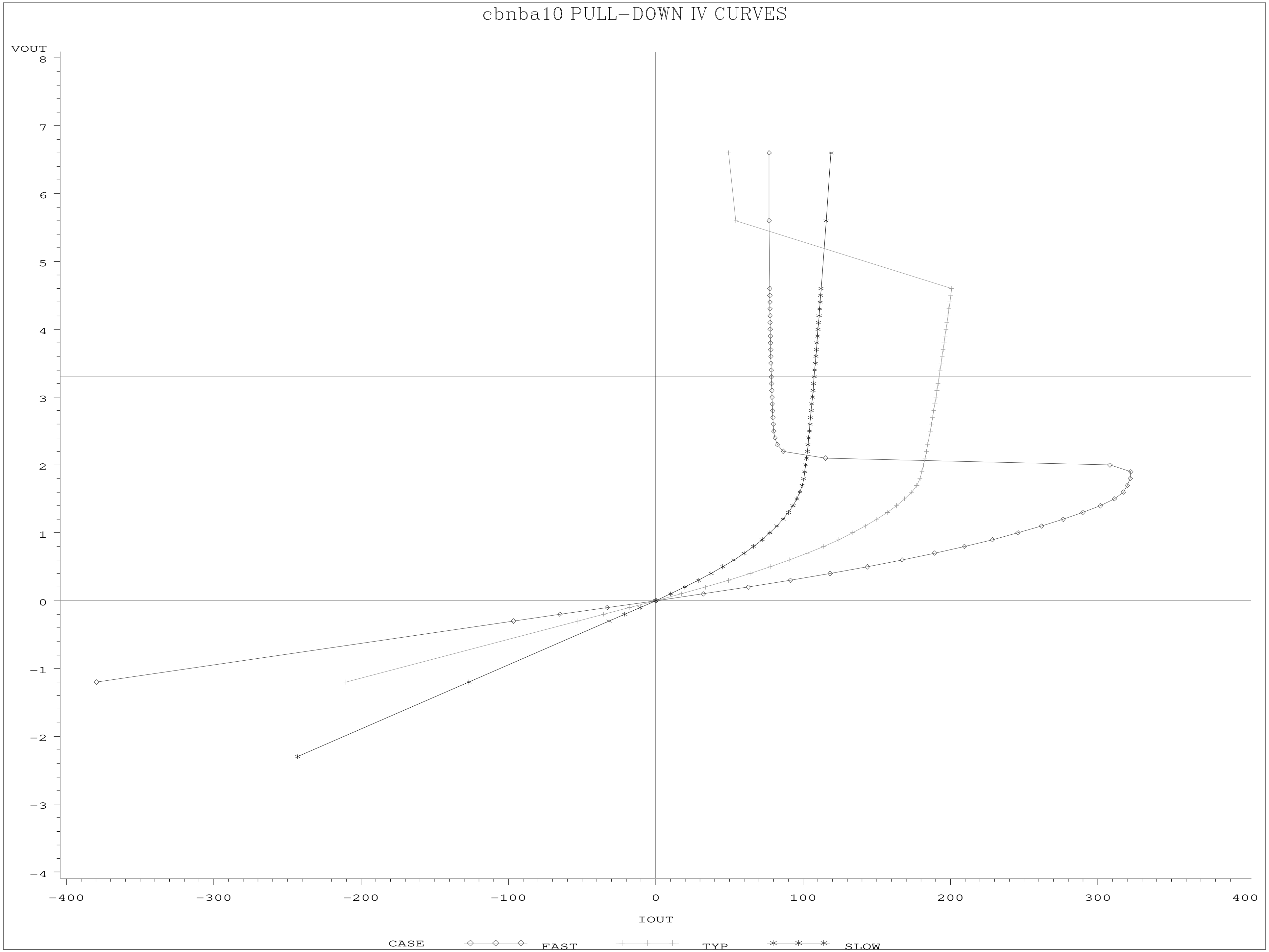
Contacts

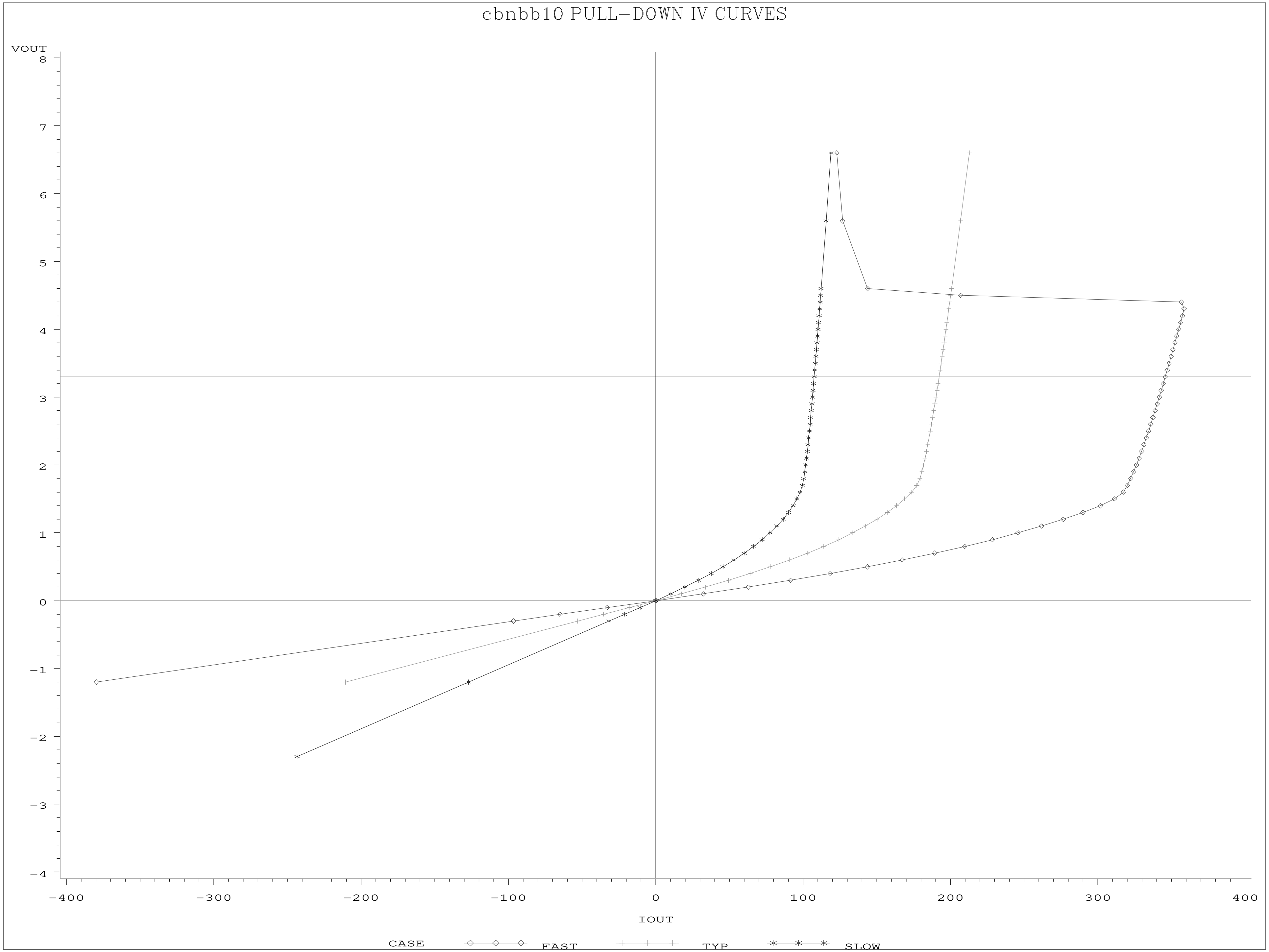
USA and Canada:
IBM Microelectronics Division
1580 Route 52, Bldg. 504
Hopewell Junction, NY 12533-6531
Tel: (800) PowerPC
Fax: (800) PowerFax

Europe:
IBM
La Pompignane BP 1021
34006 Montpellier, France
Tel: (33) 6713–5757 (Français)
(33) 6713–5756 (Italiano)
Fax: (33) 6713–5750
(from Paris add 16)

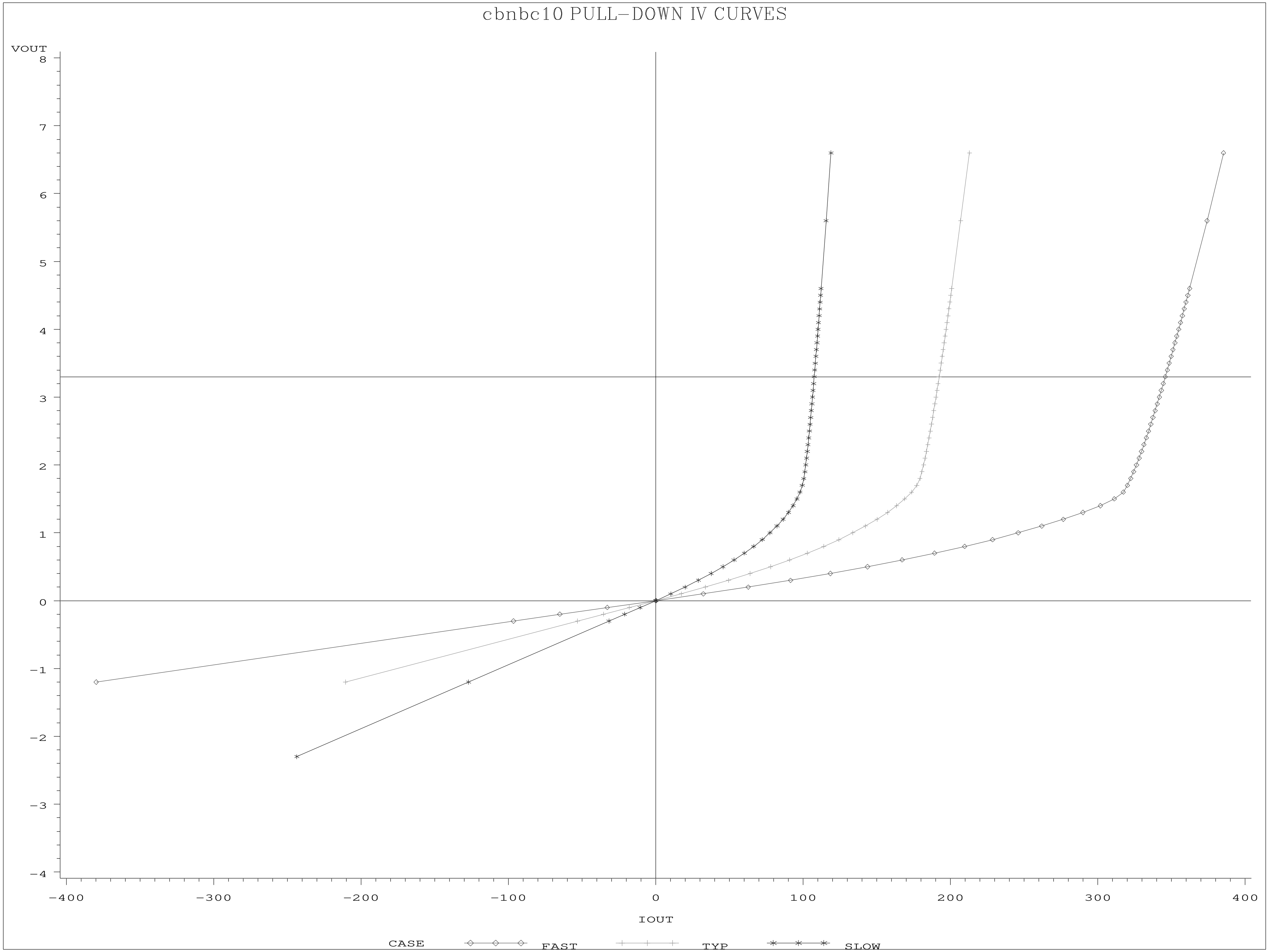
Europe:
IBM
Informations Systeme GmbH
Laatzener Str. 1
30539 Hannover, Germany
Tel: (49) 511–516–3444 (English)
(49) 511–516–3555 (Deutsche)
Fax: (49) 511–516–3888

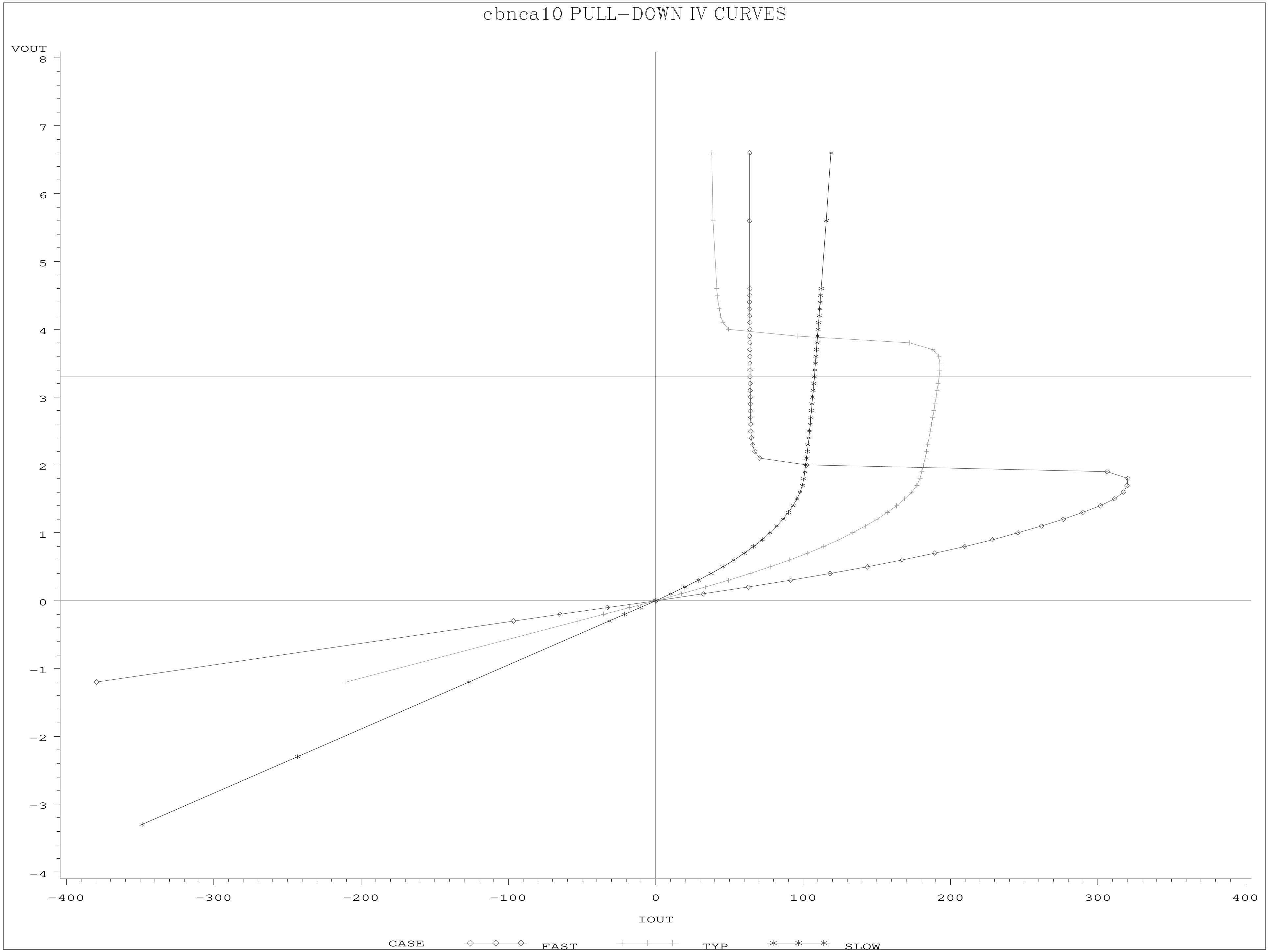
Japan:
IBM
800, Ichimiyake
Yasu–cho, Yasu–gun
Shiga–ken, Japan 520–23
Tel: (81) 775–87–4745
Fax: (81) 775–87–4735

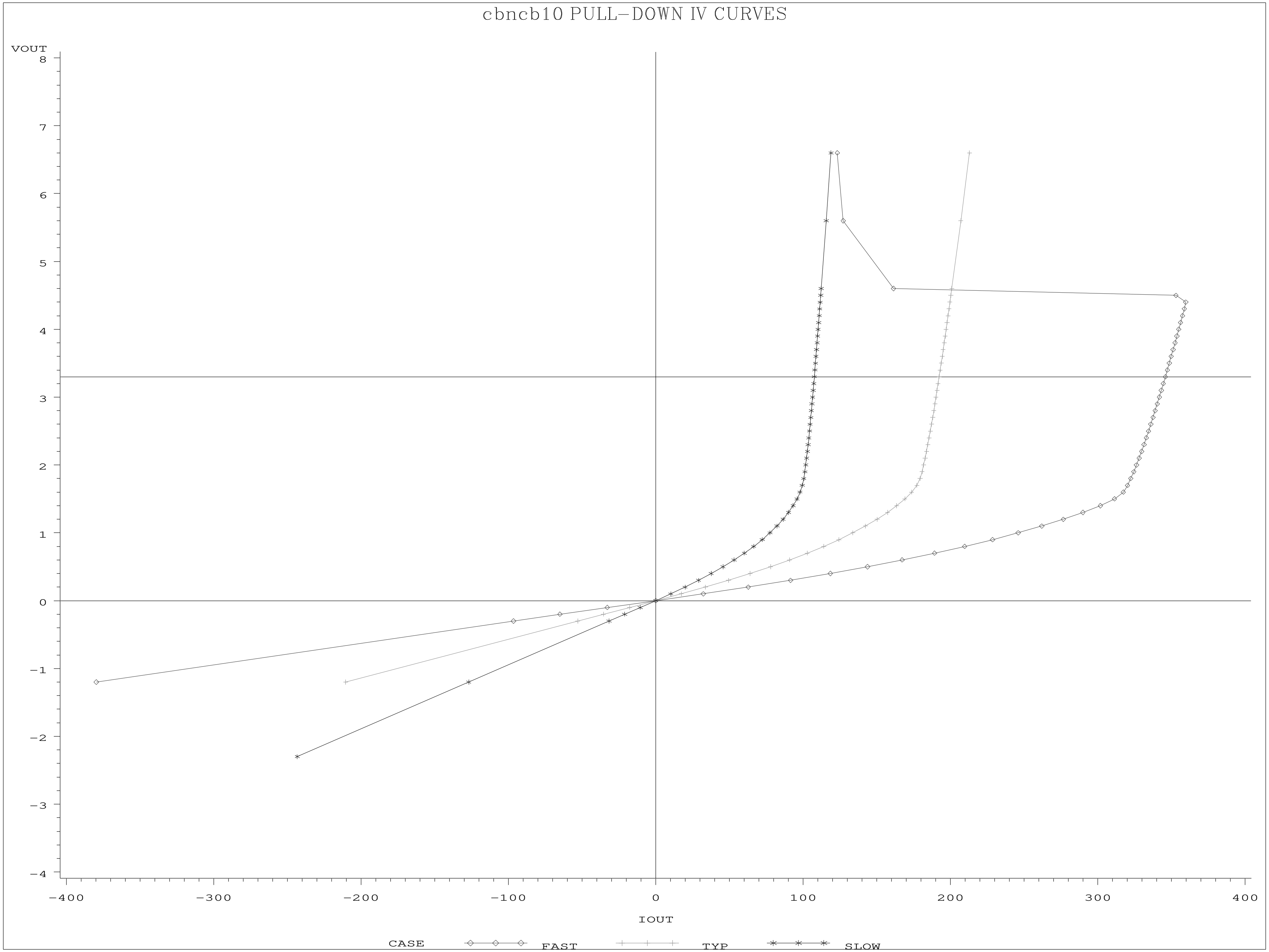

cbnba10 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

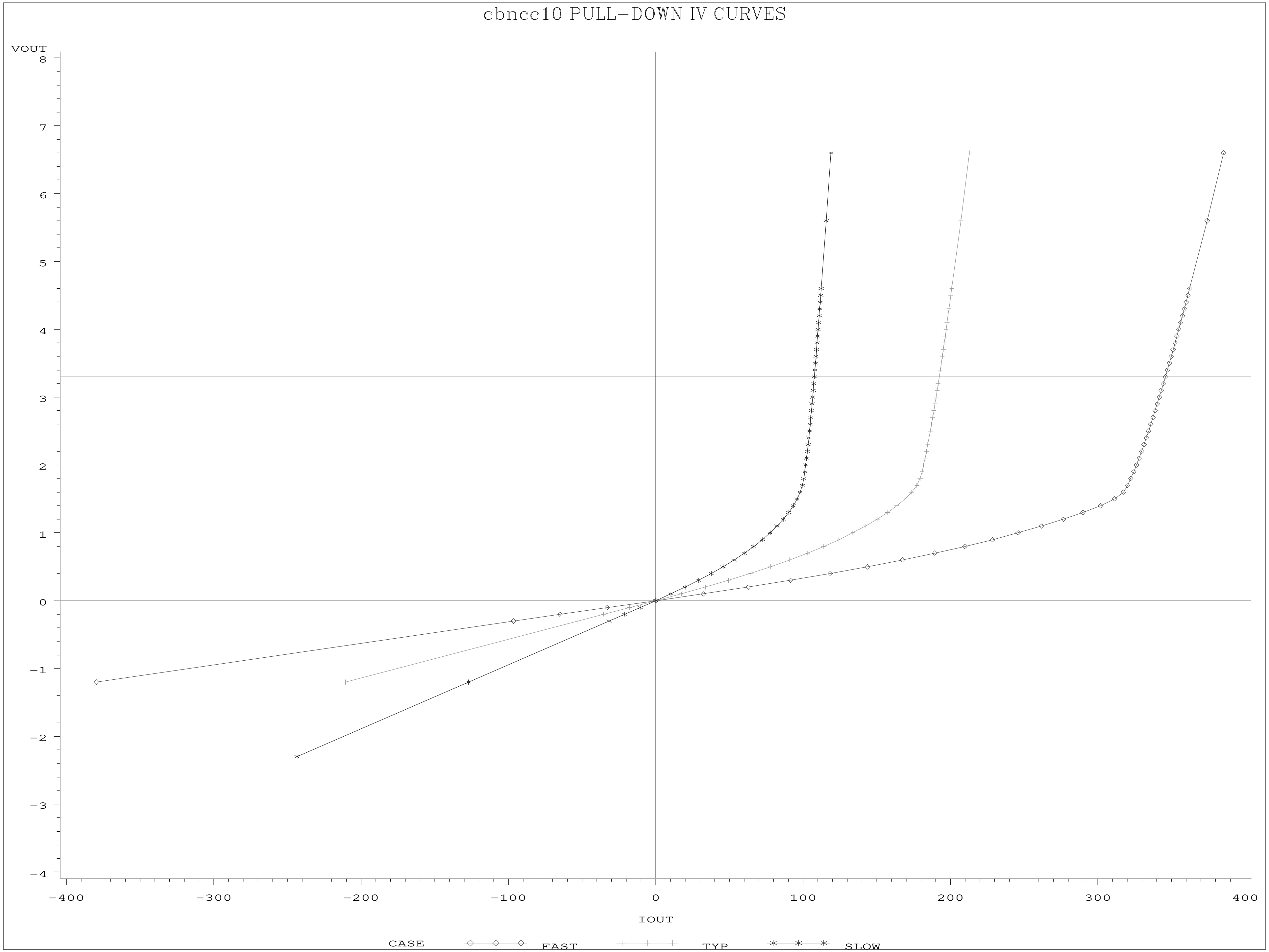


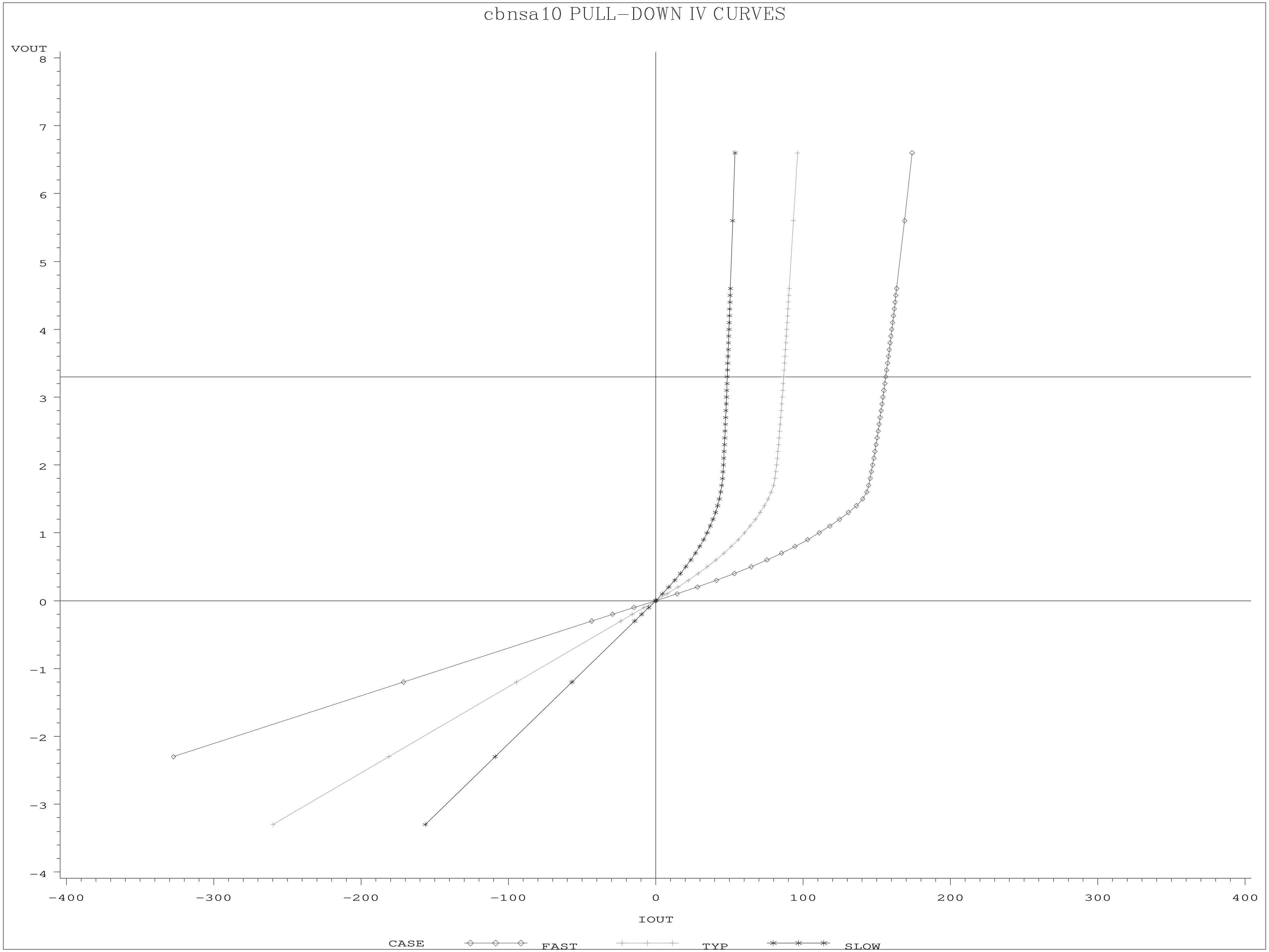
cbnbb10 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW


cbnbc10 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

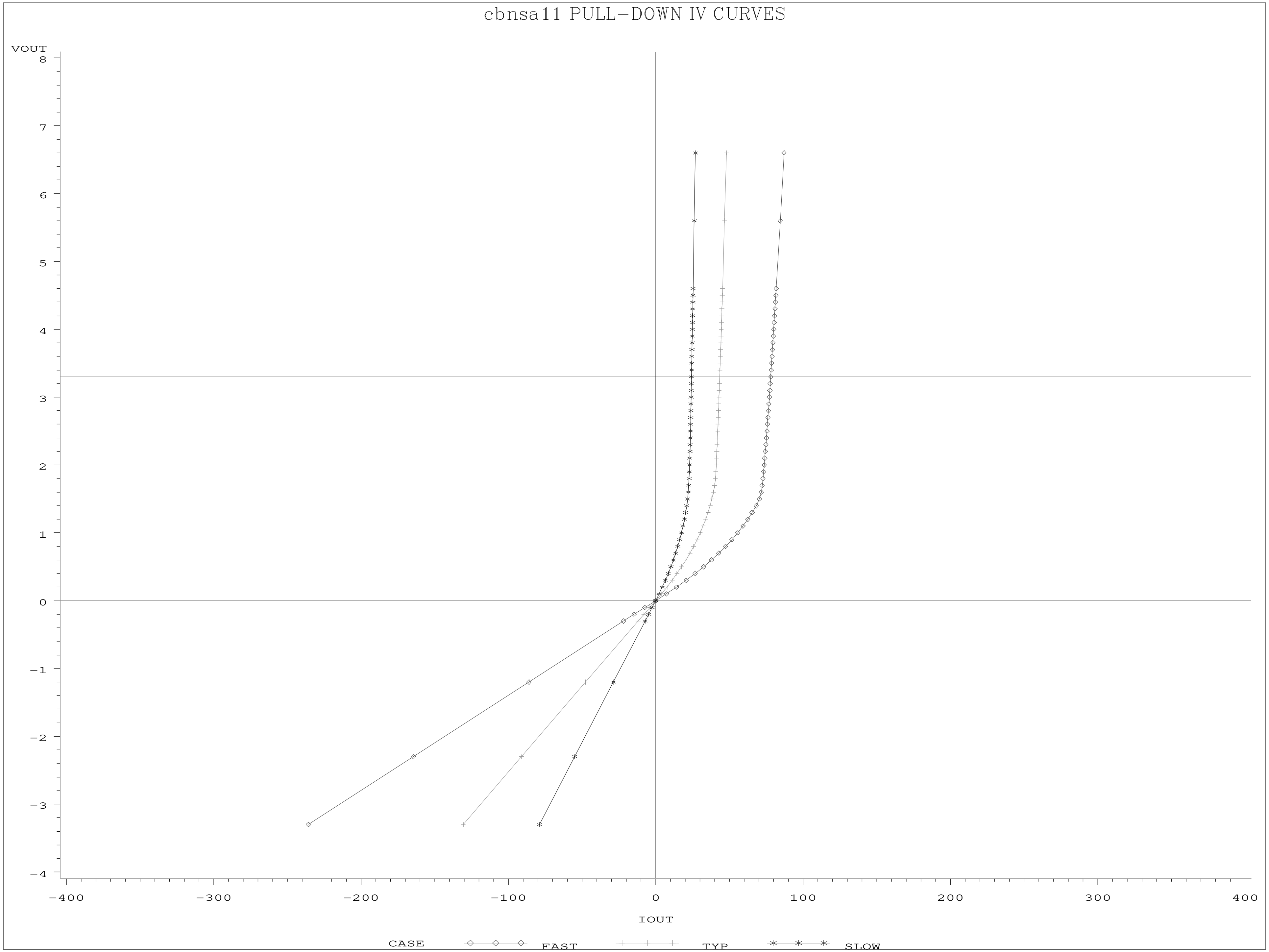

cbnca10 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

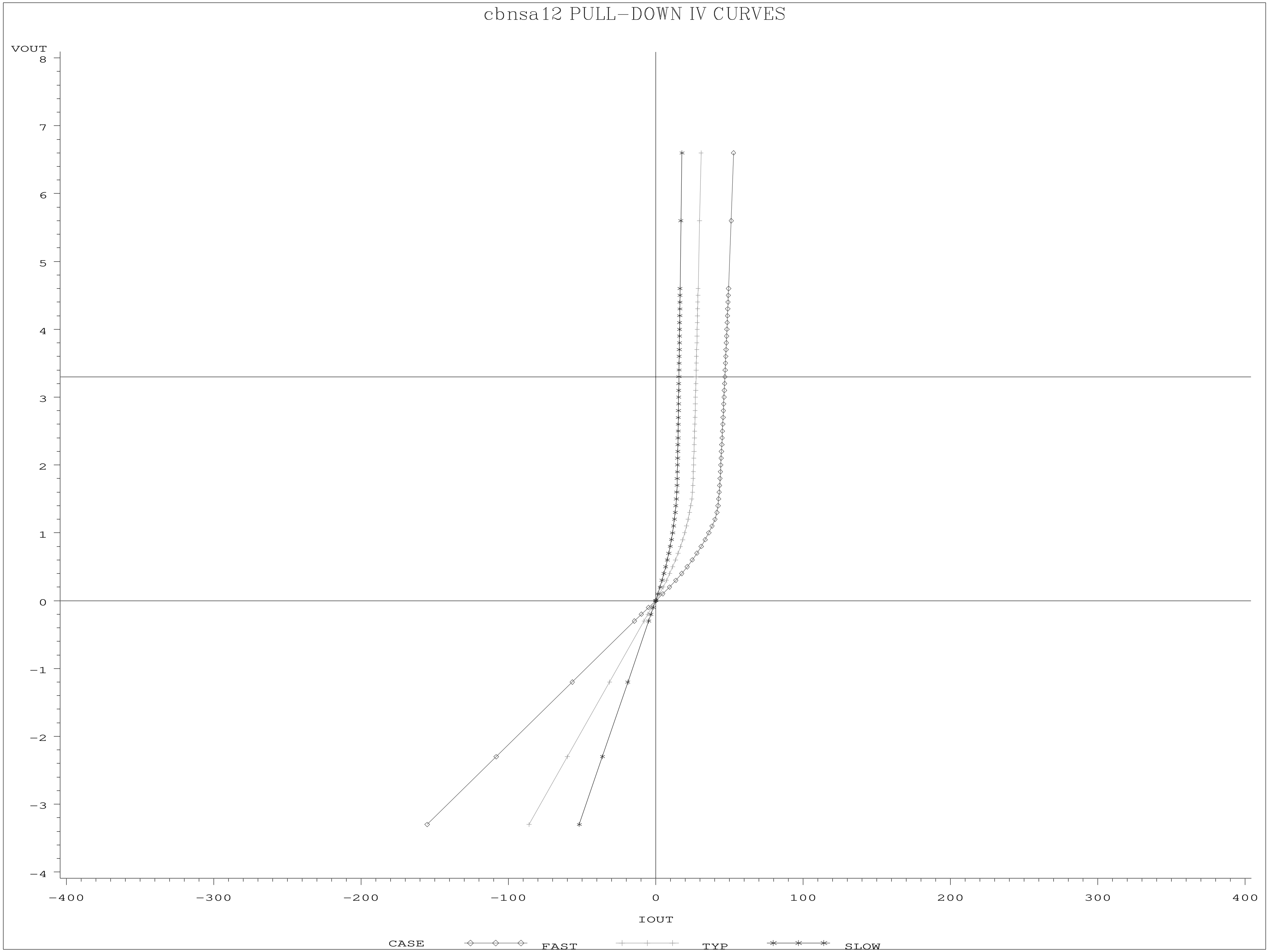

cbncb10 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW


cbncc10 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

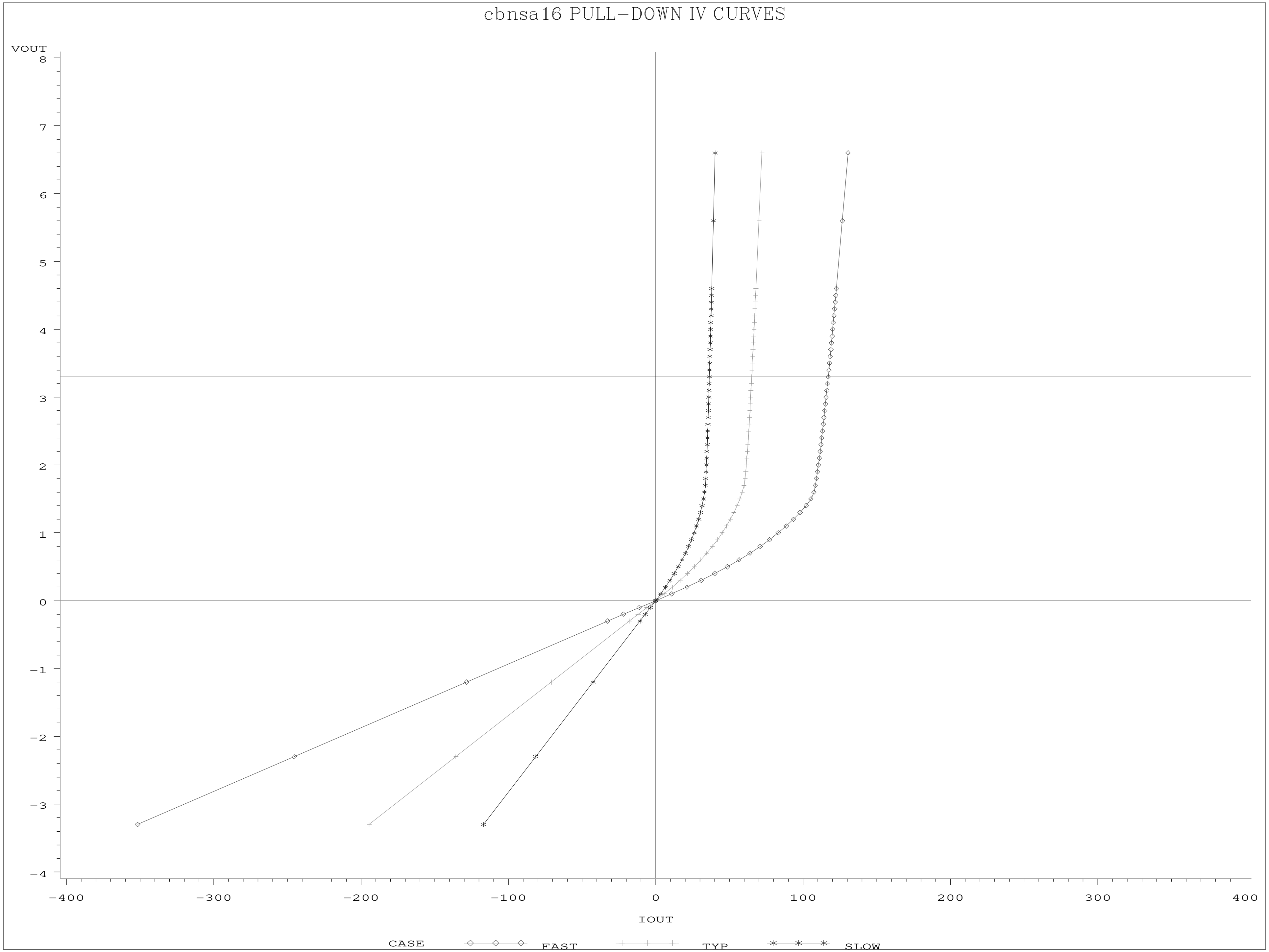


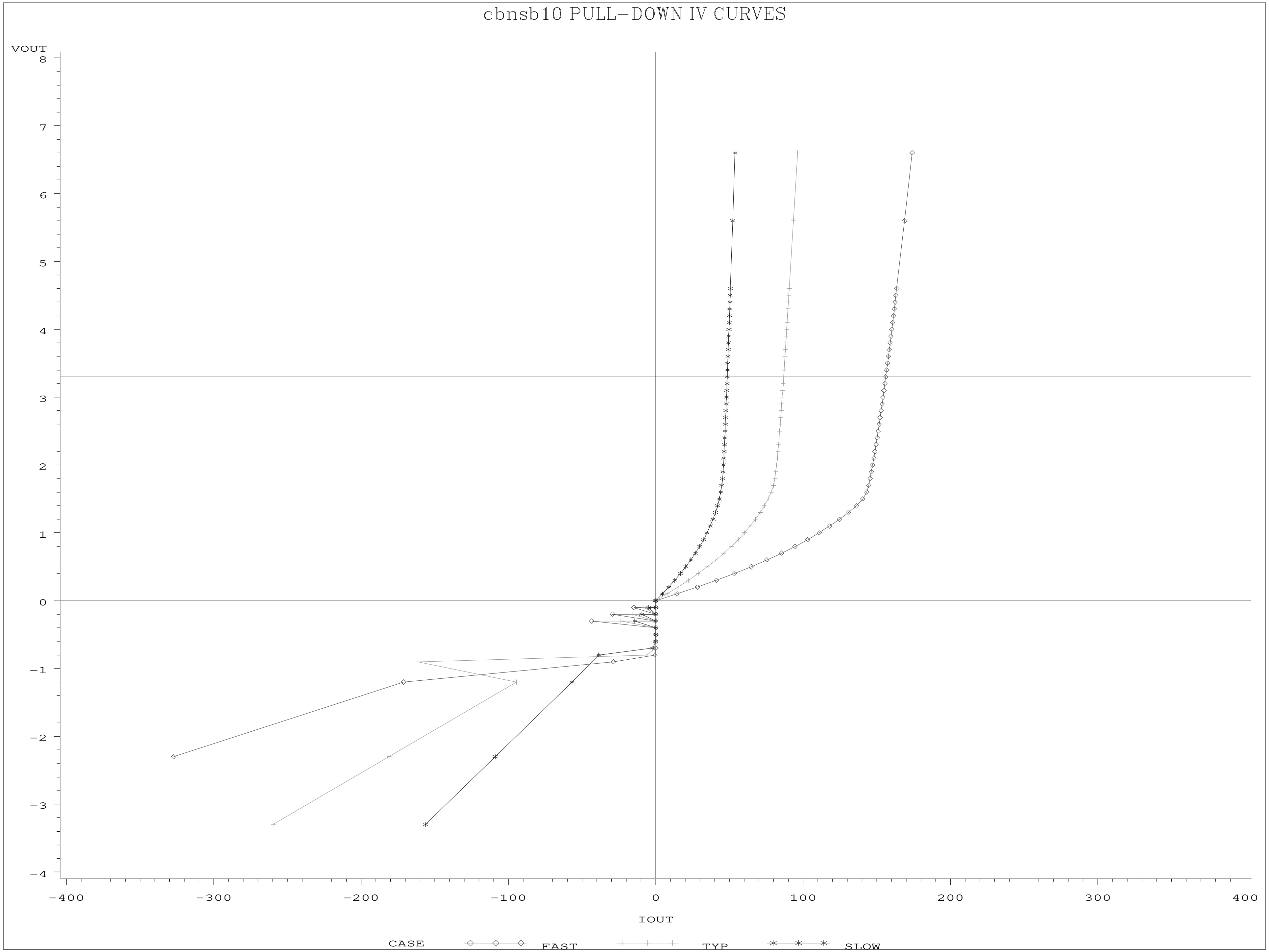
cbnsa10 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW


cbnsa11 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

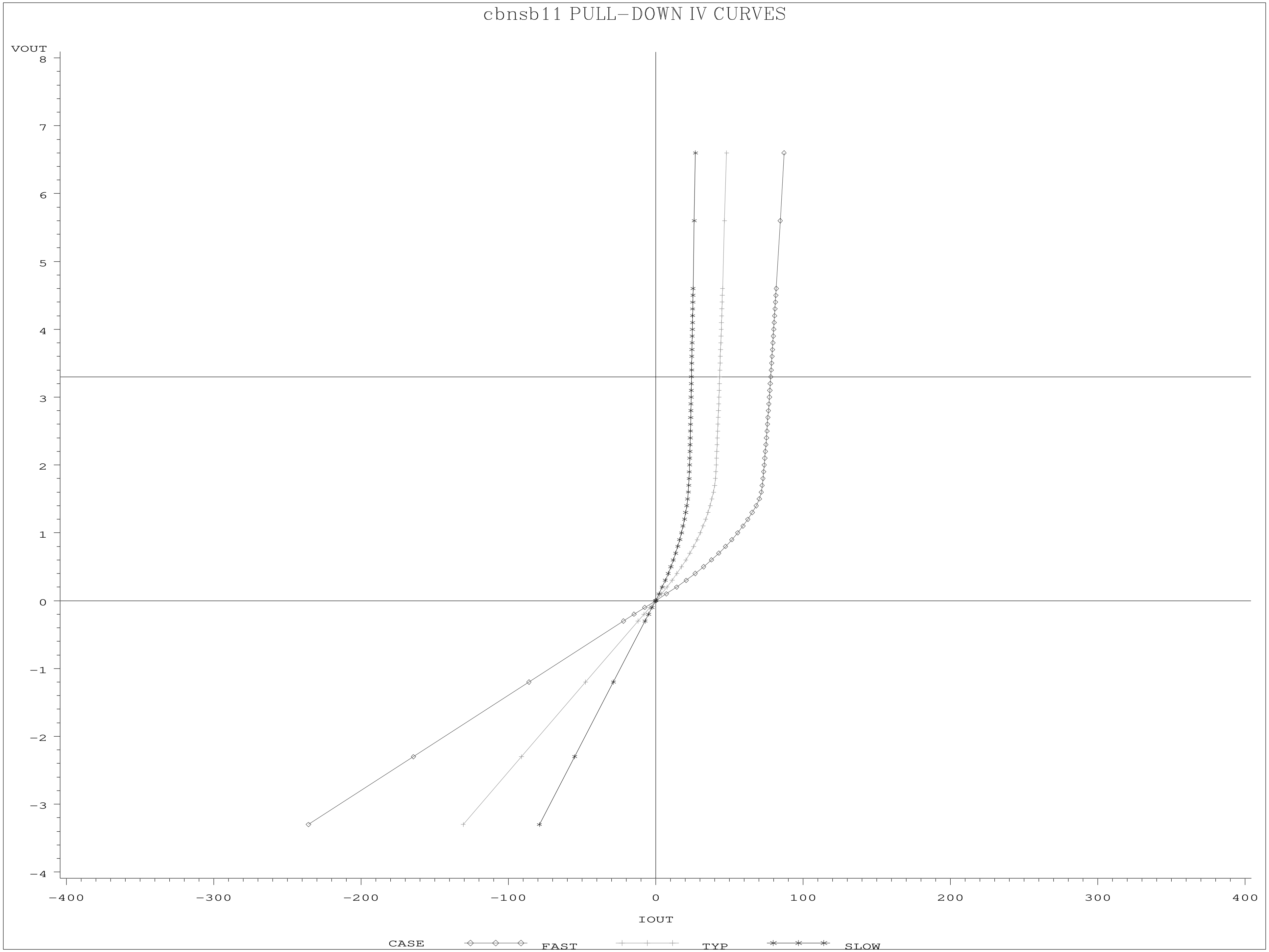


cbnsa12 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

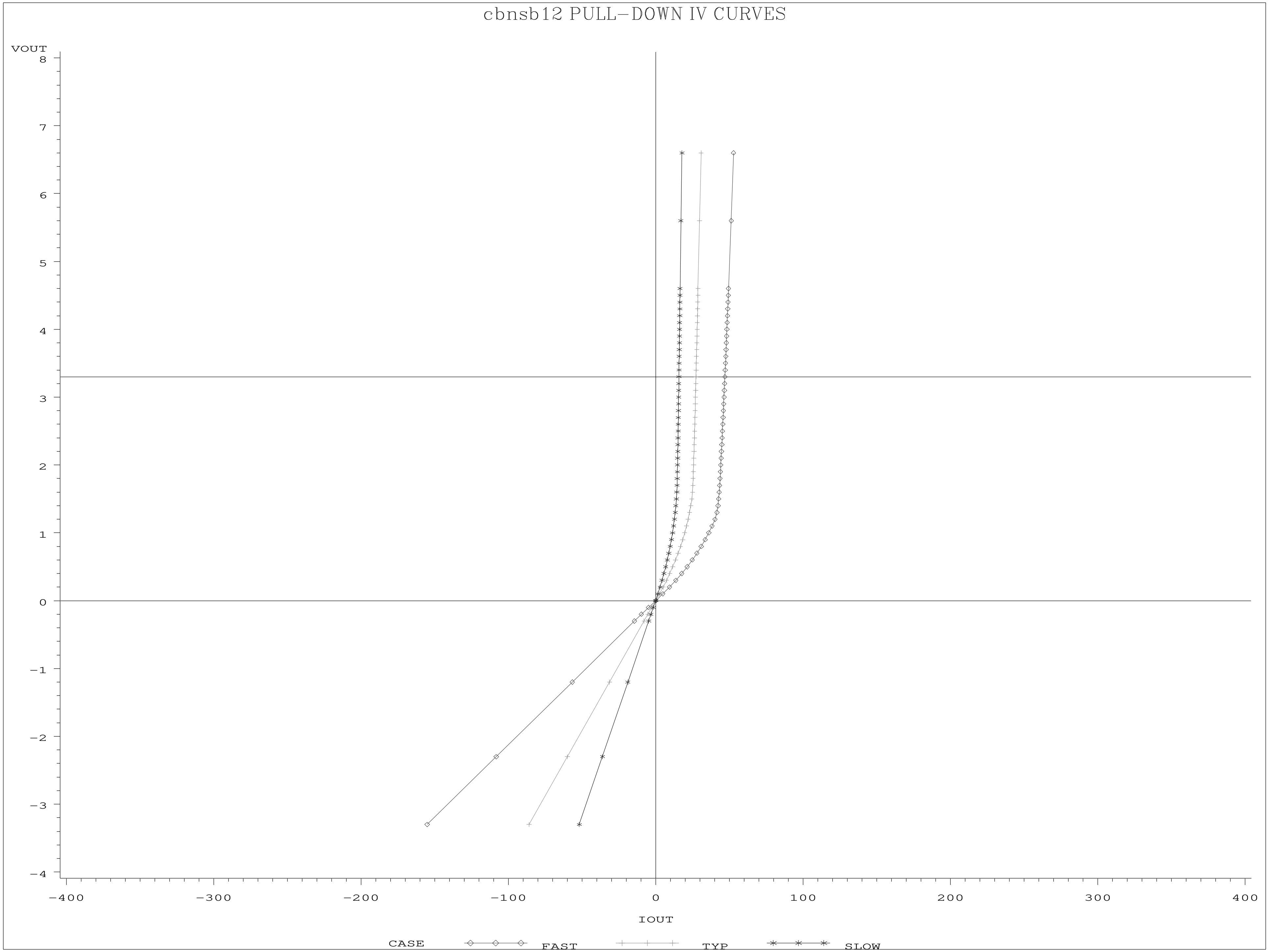

cbnsa16 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW



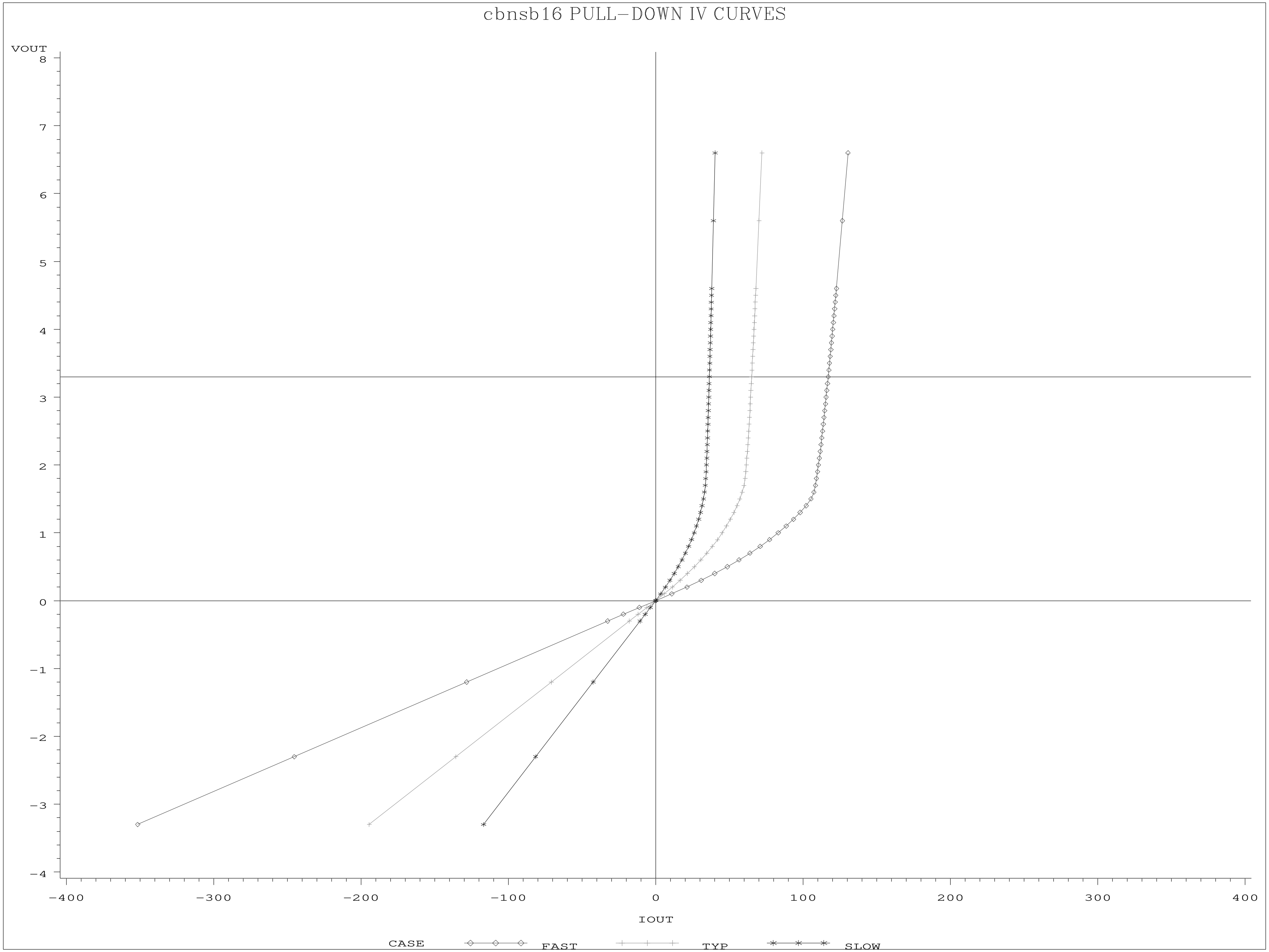
cbnsb10 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW



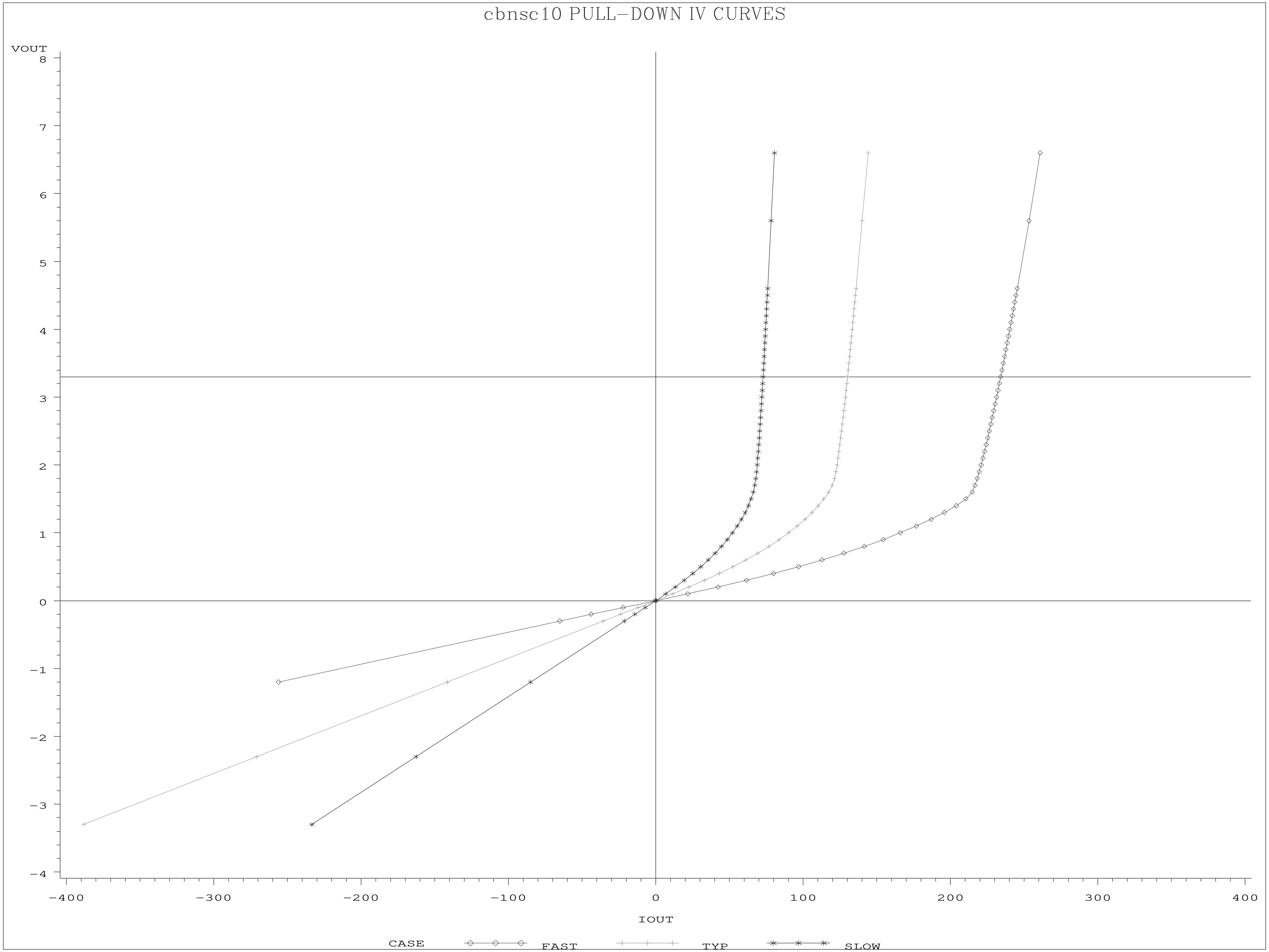
cbnsb11 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

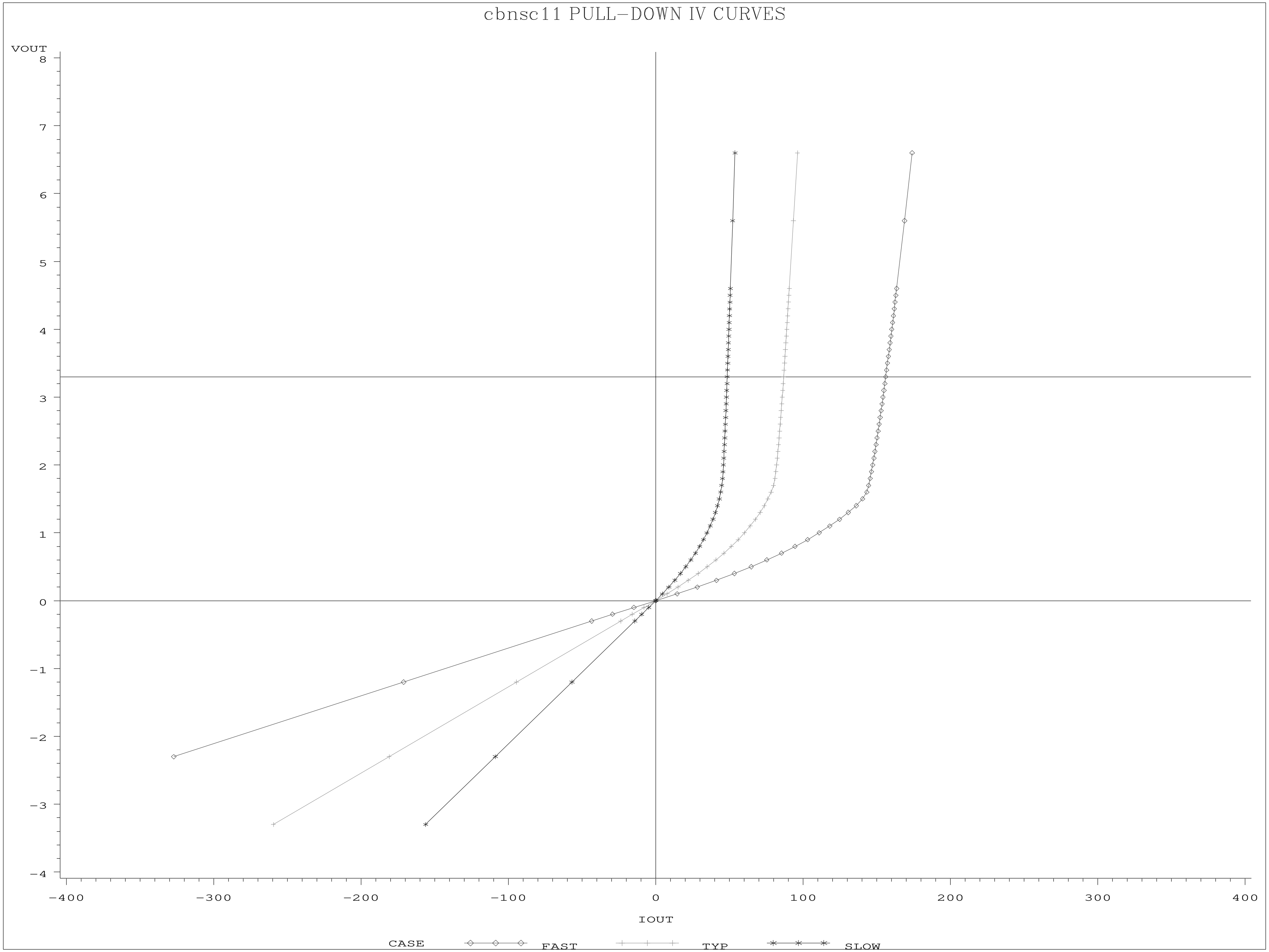


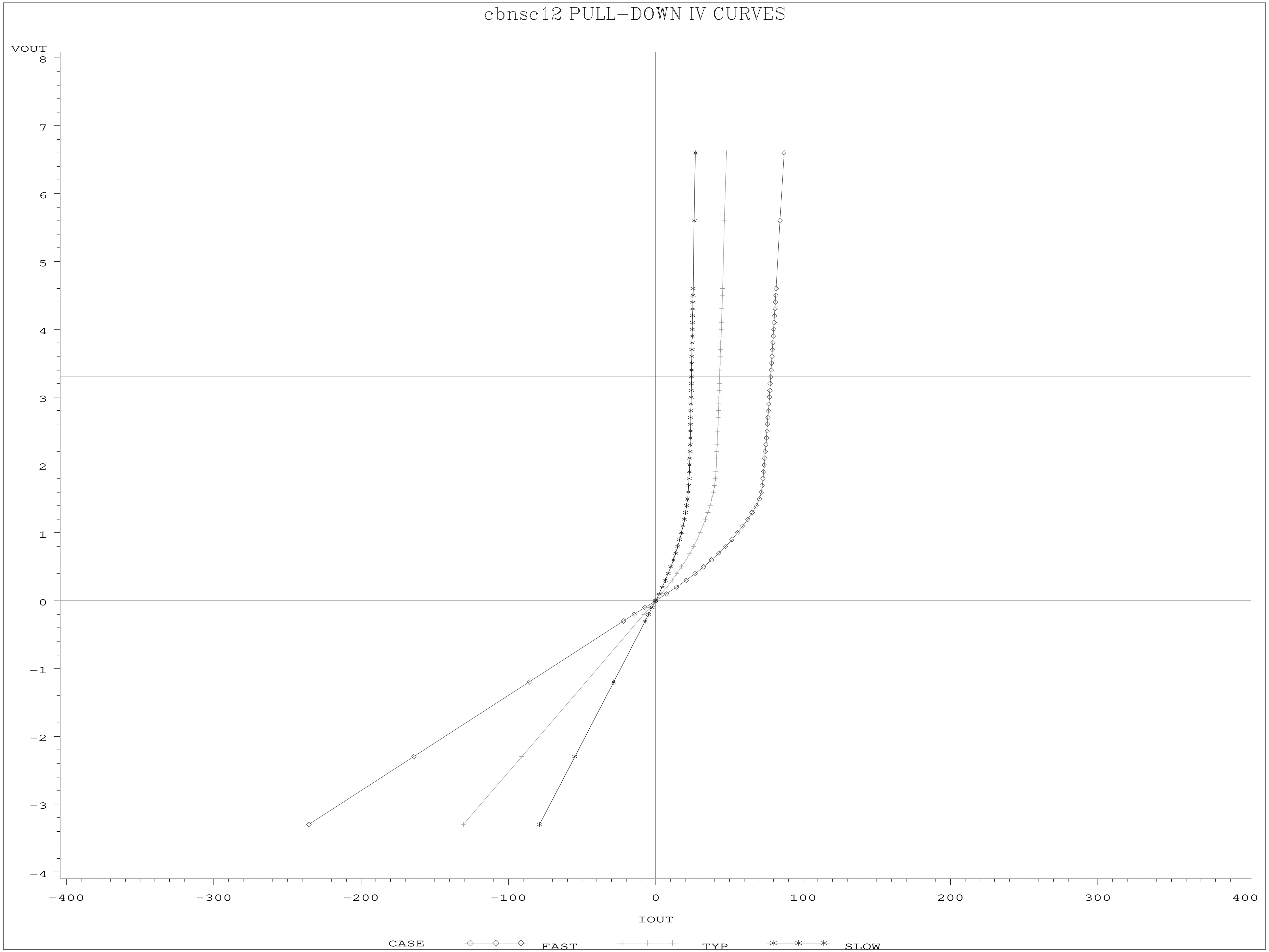
cbnsb12 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW



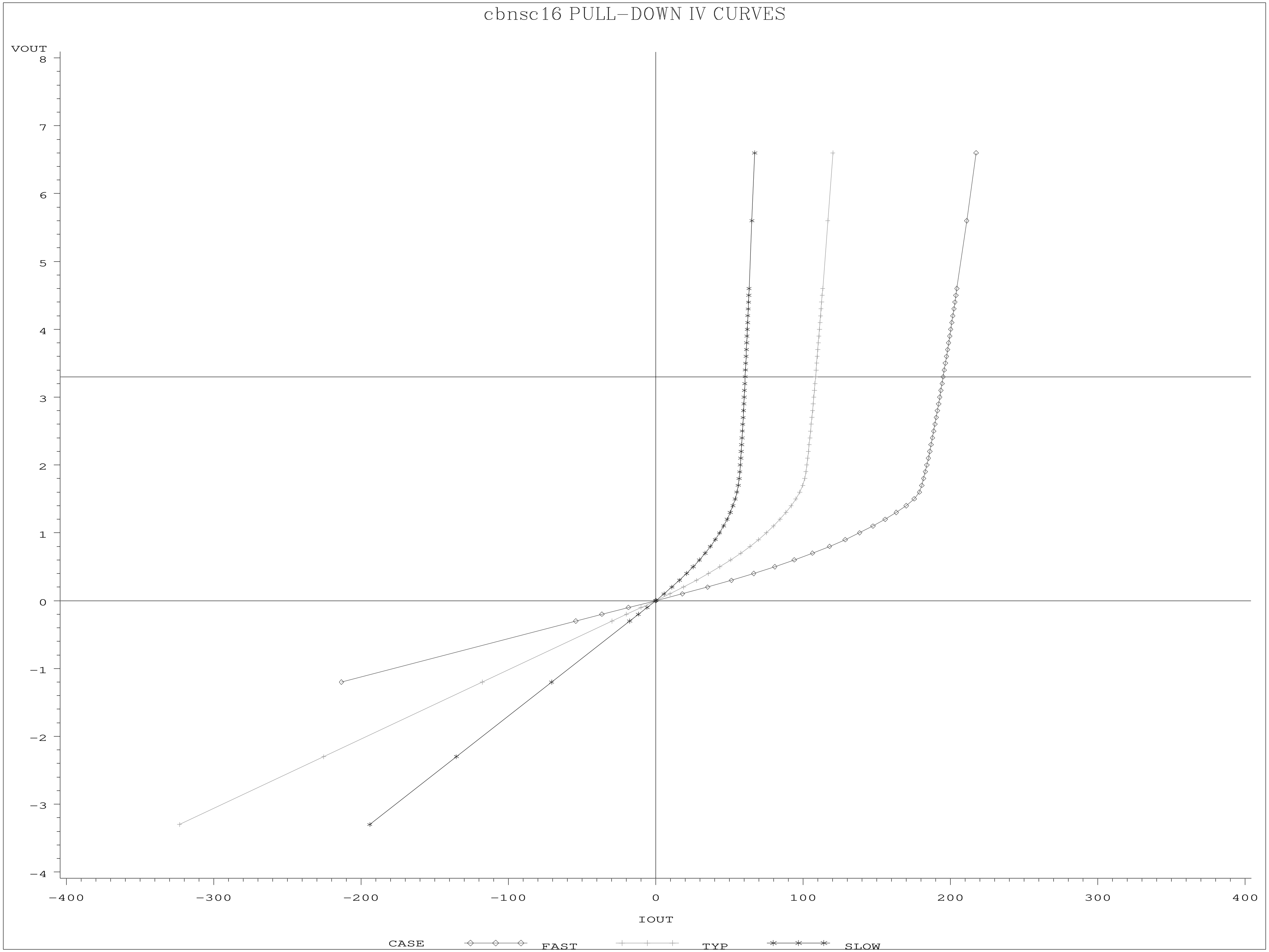
cbnsb16 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW


cbnsc10 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

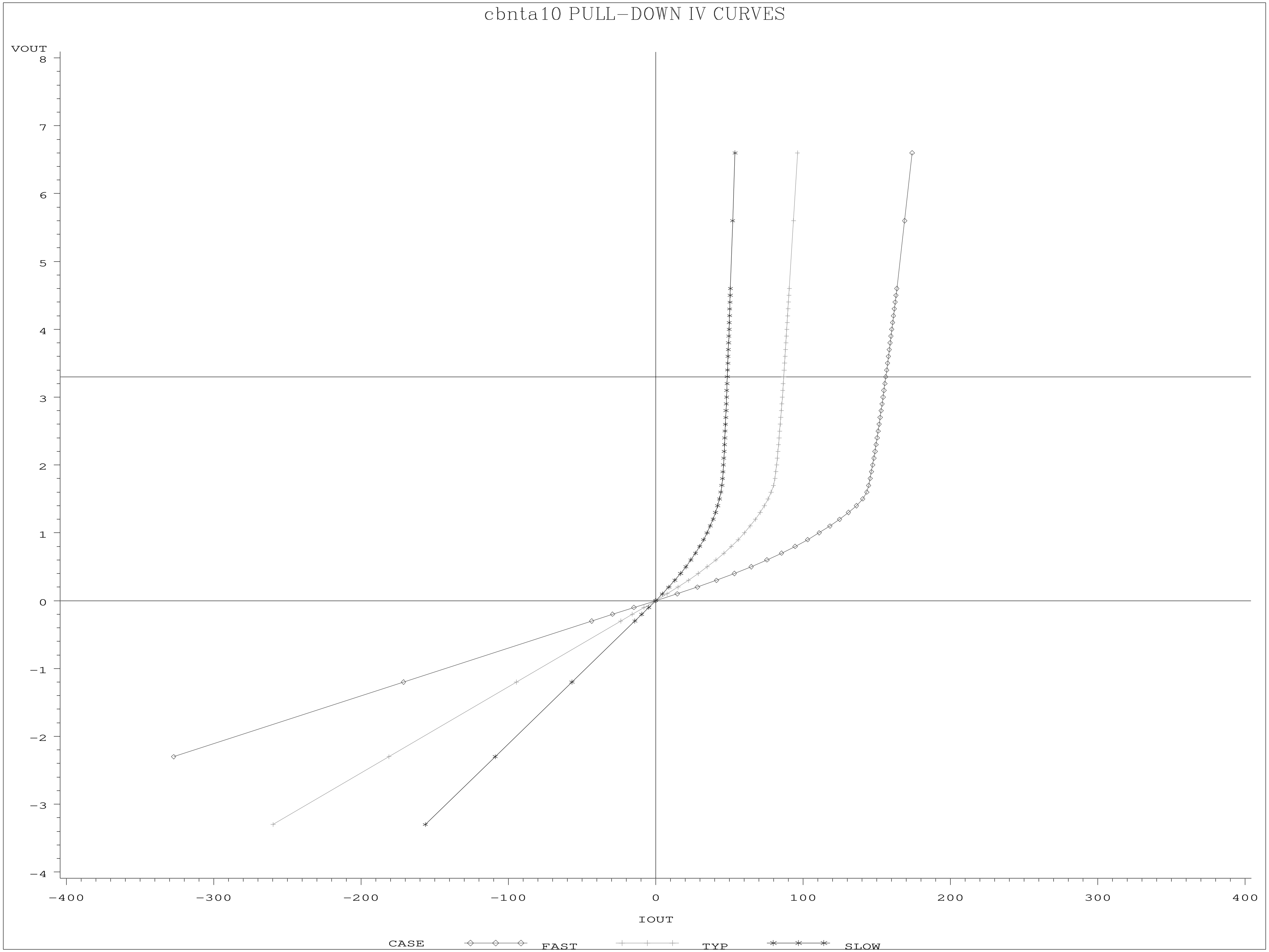

cbnsc11 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

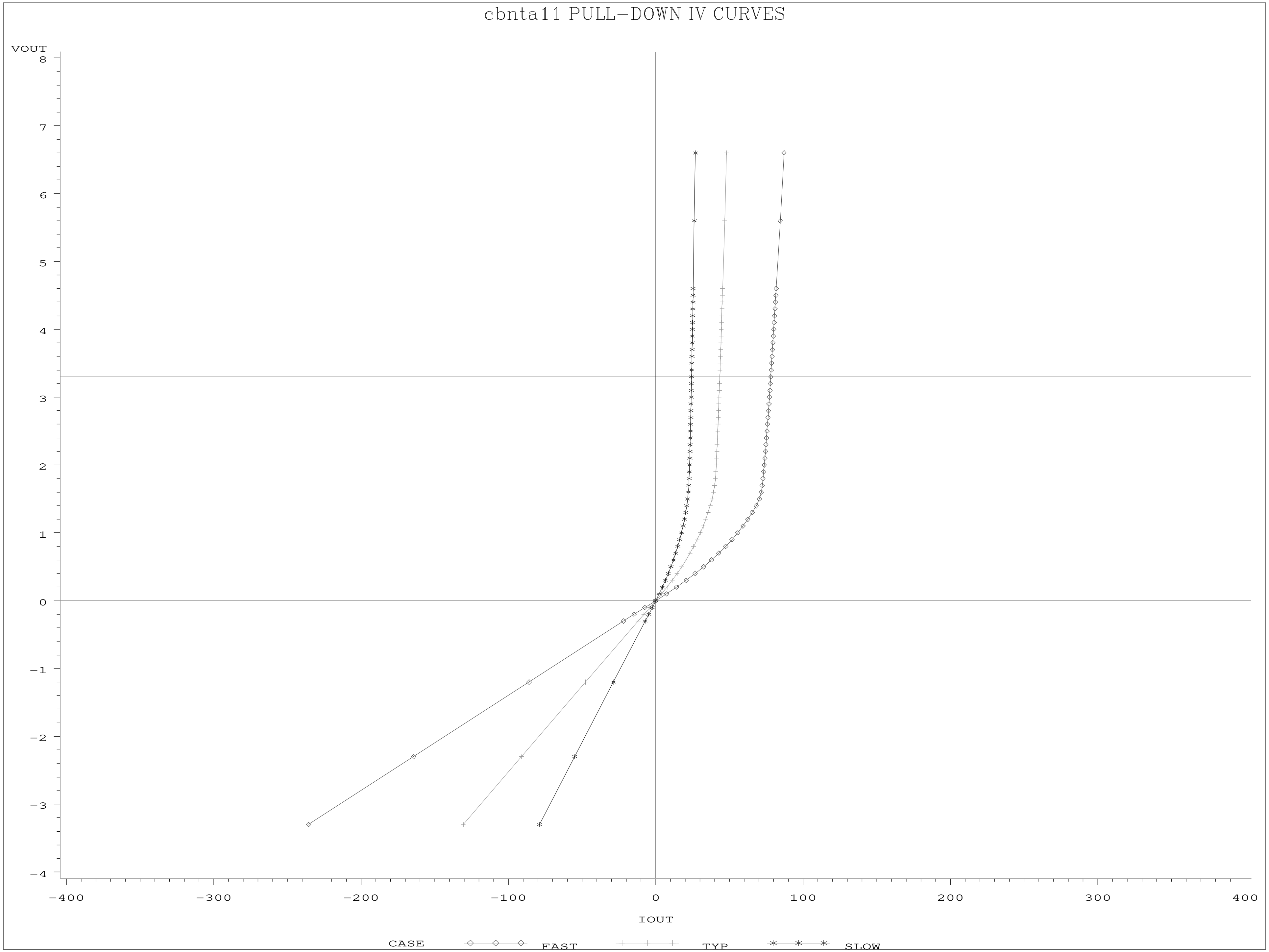


cbnsc12 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

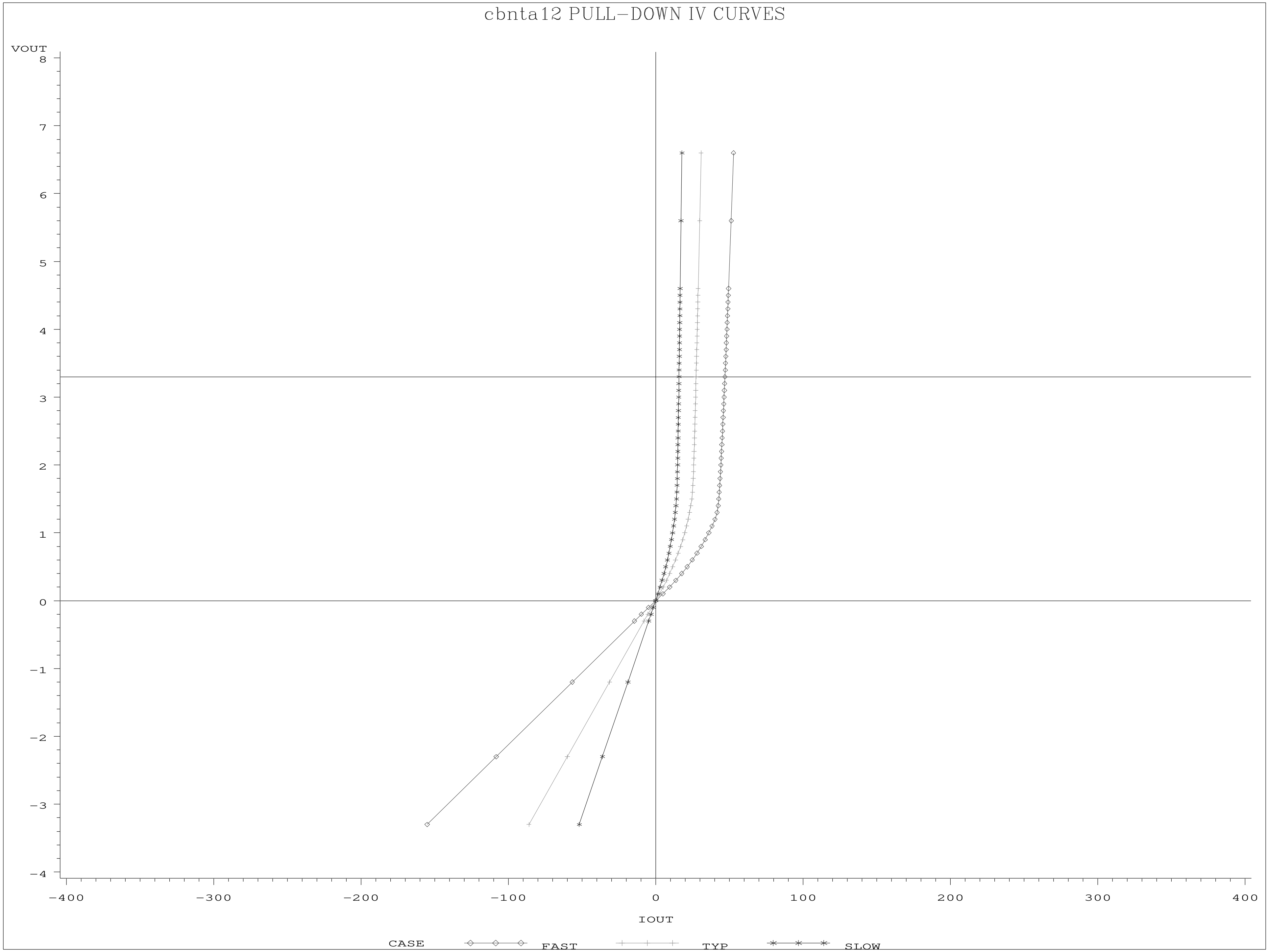


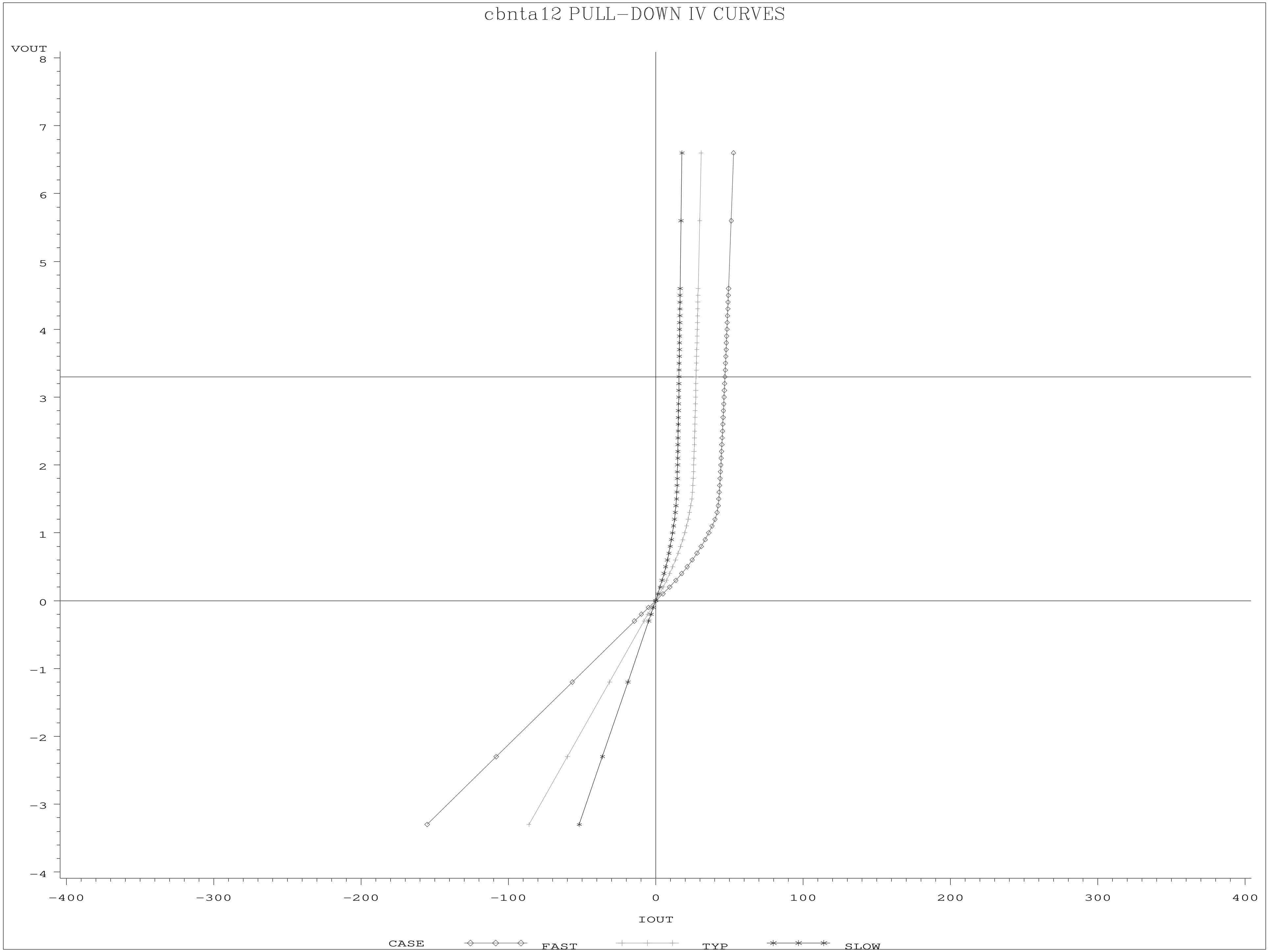
cbnsc16 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

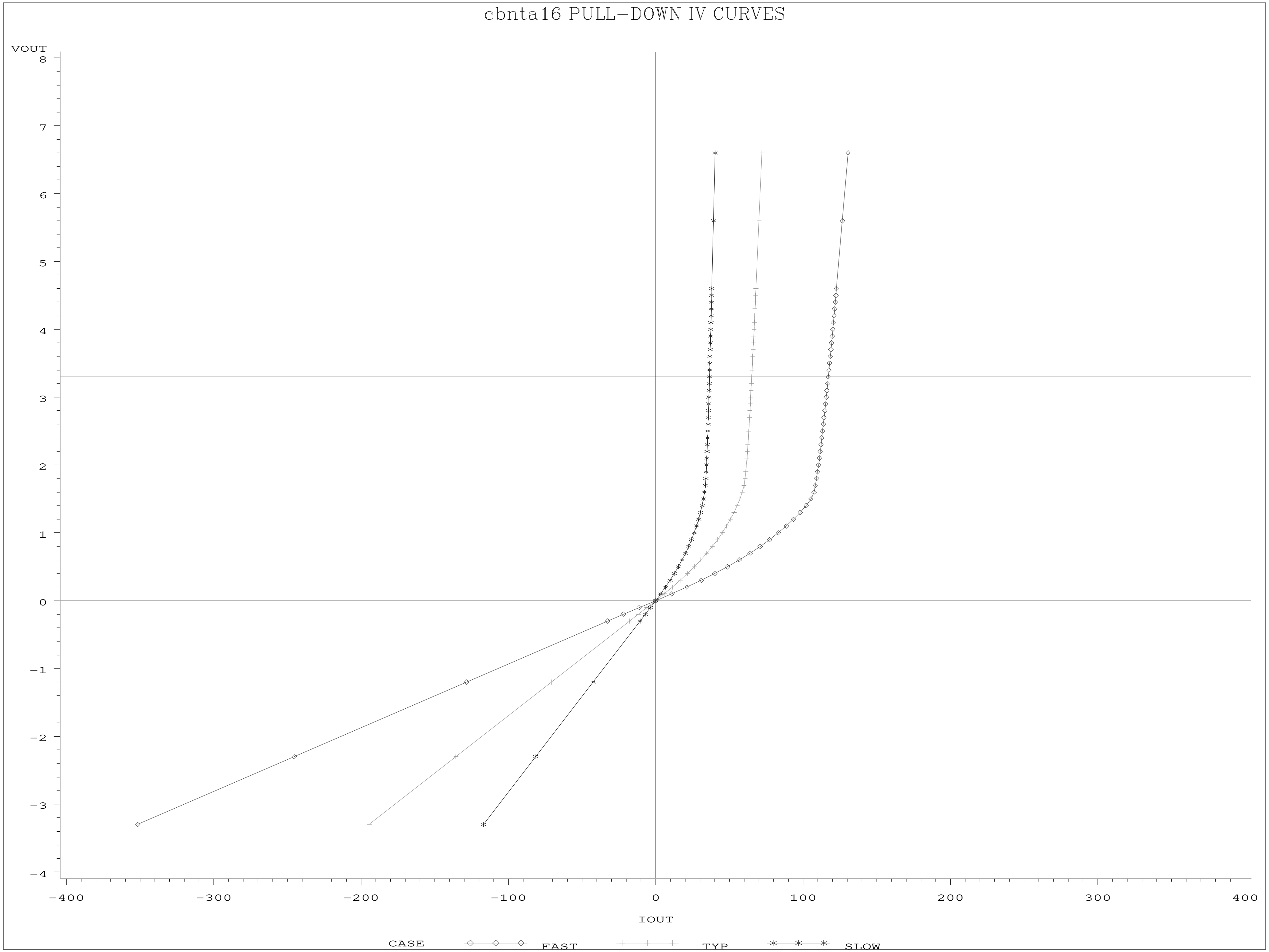

cbnta10 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW



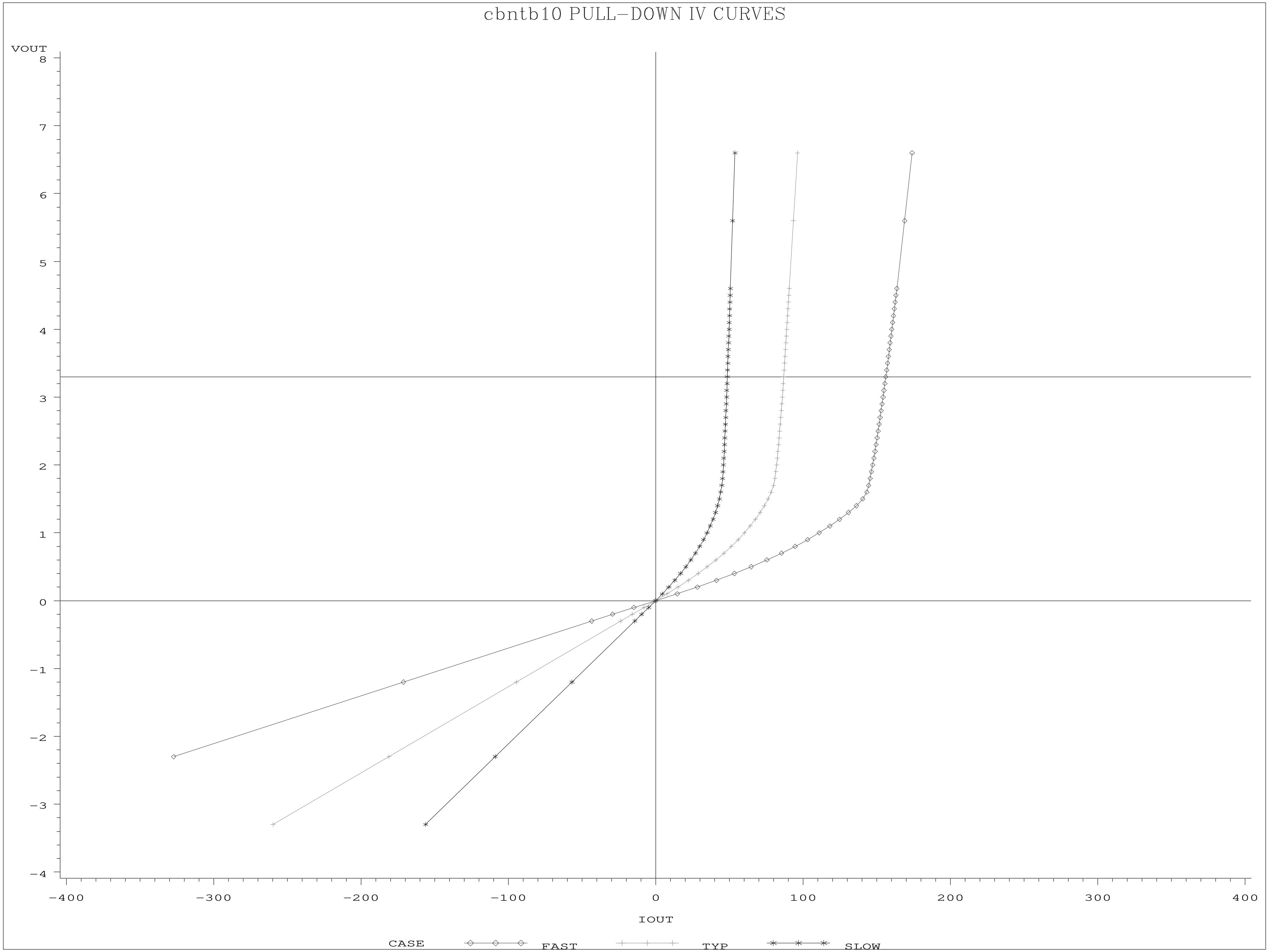
cbnta11 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

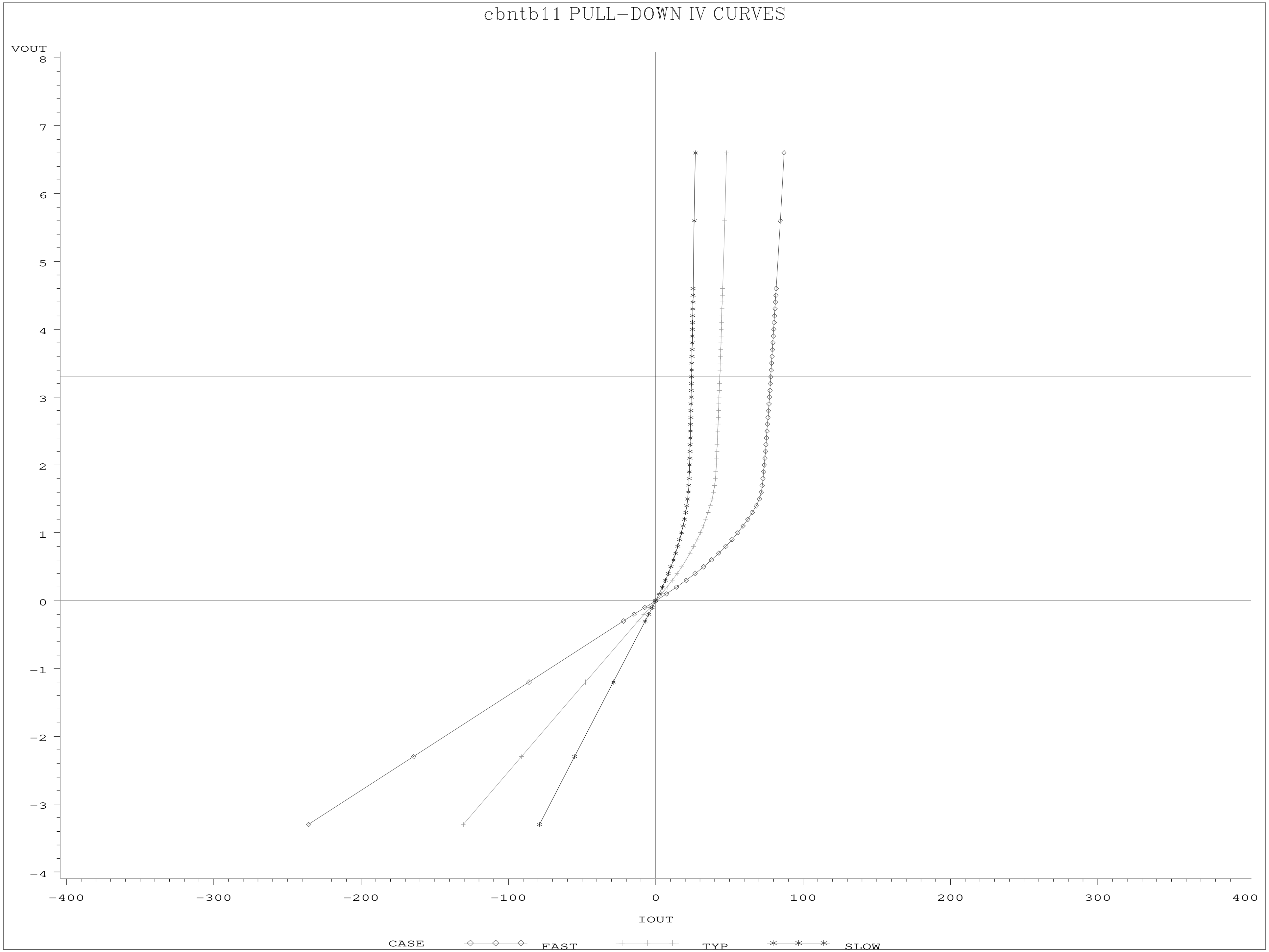

cbnta12 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW


cbnta12 PULL−DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

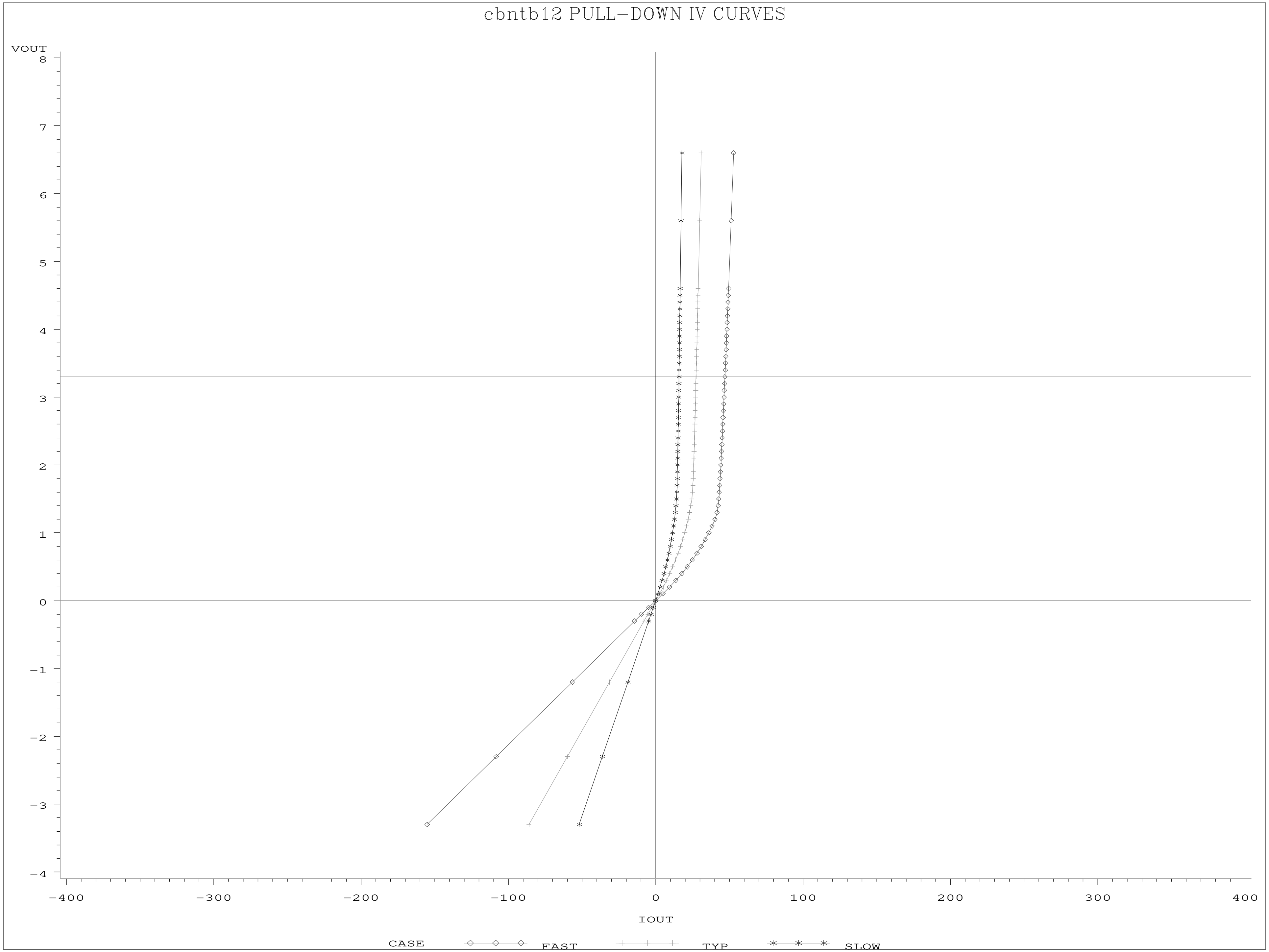


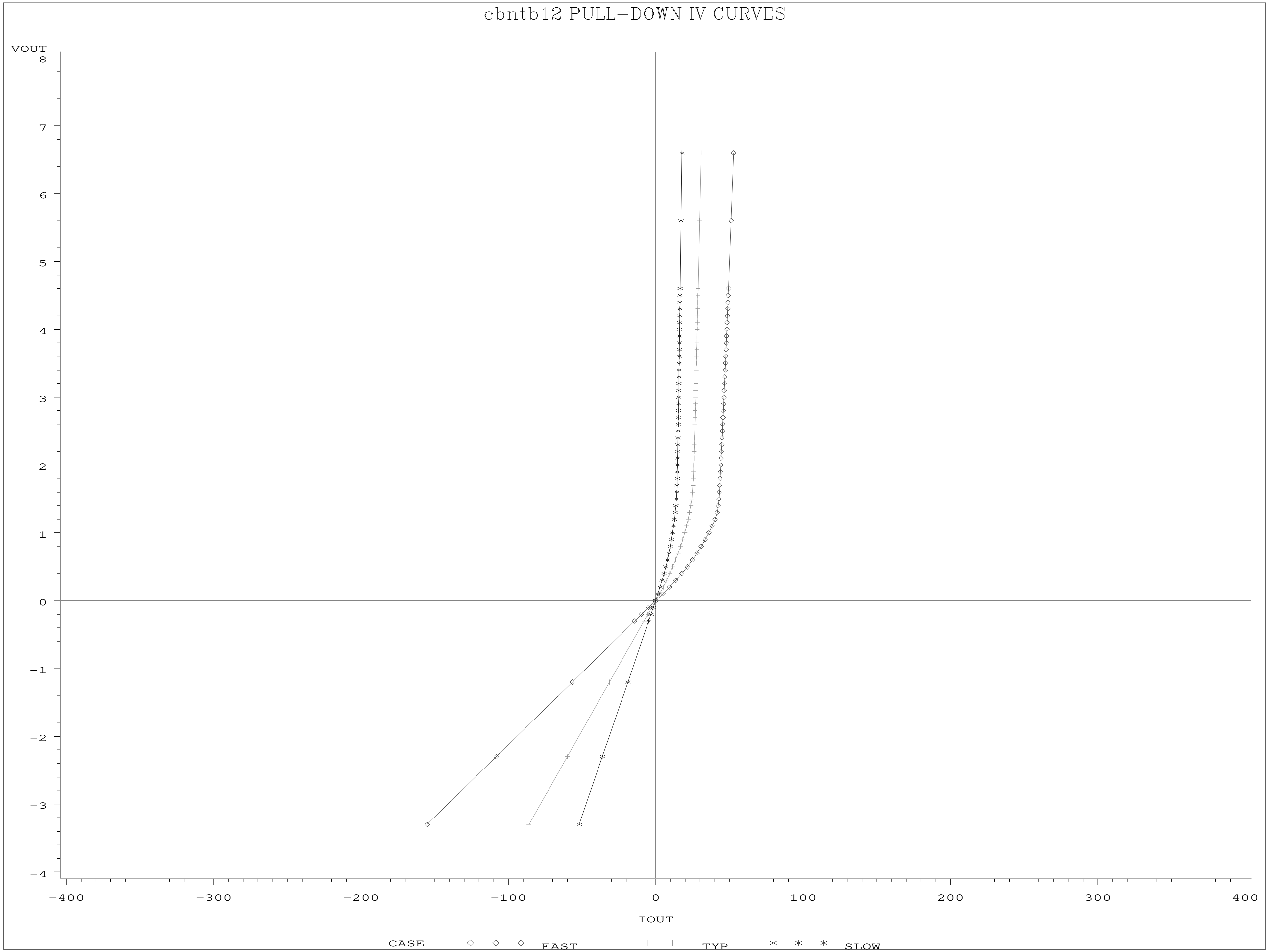
cbnta16 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

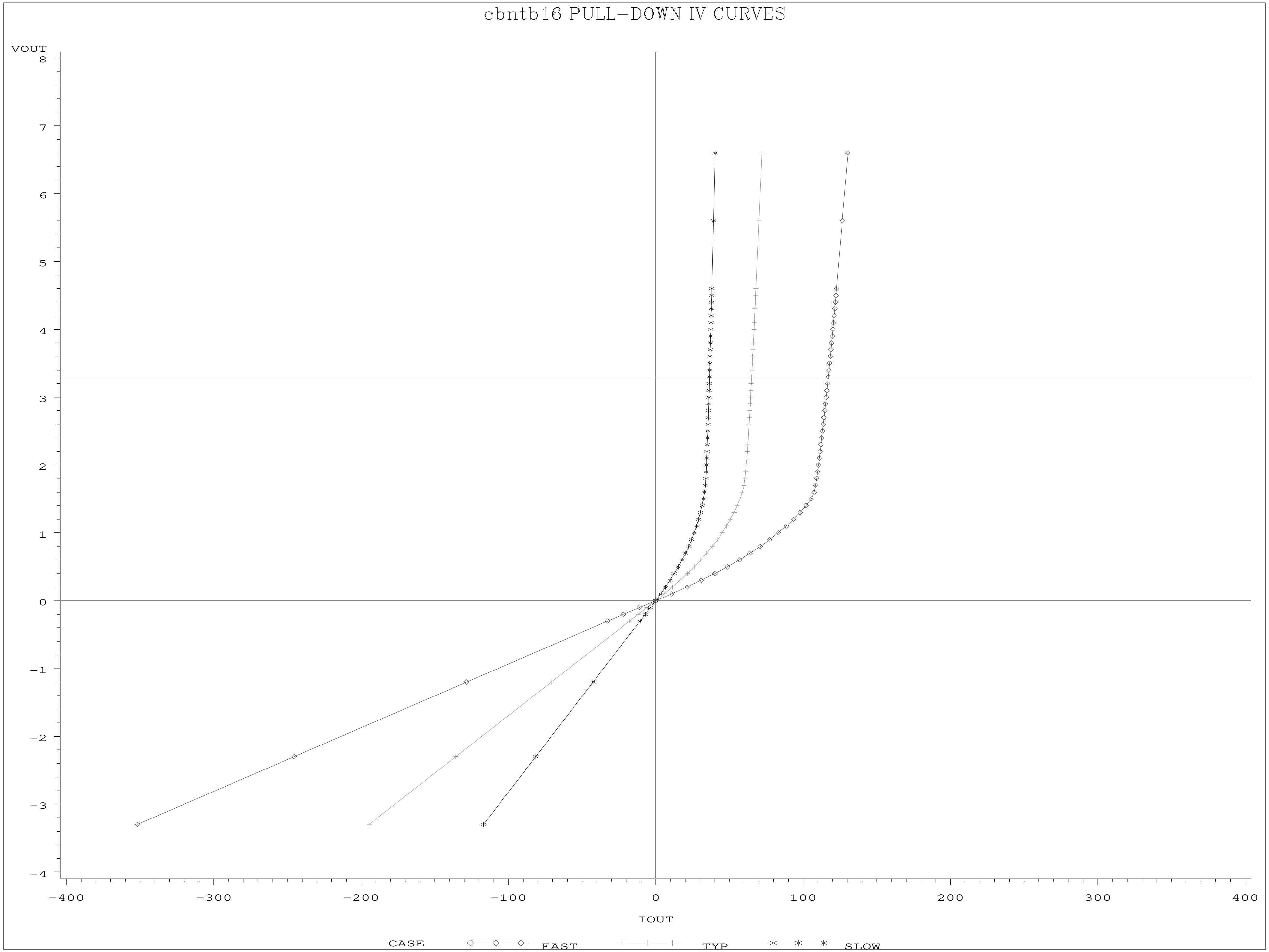

cbntb10 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW



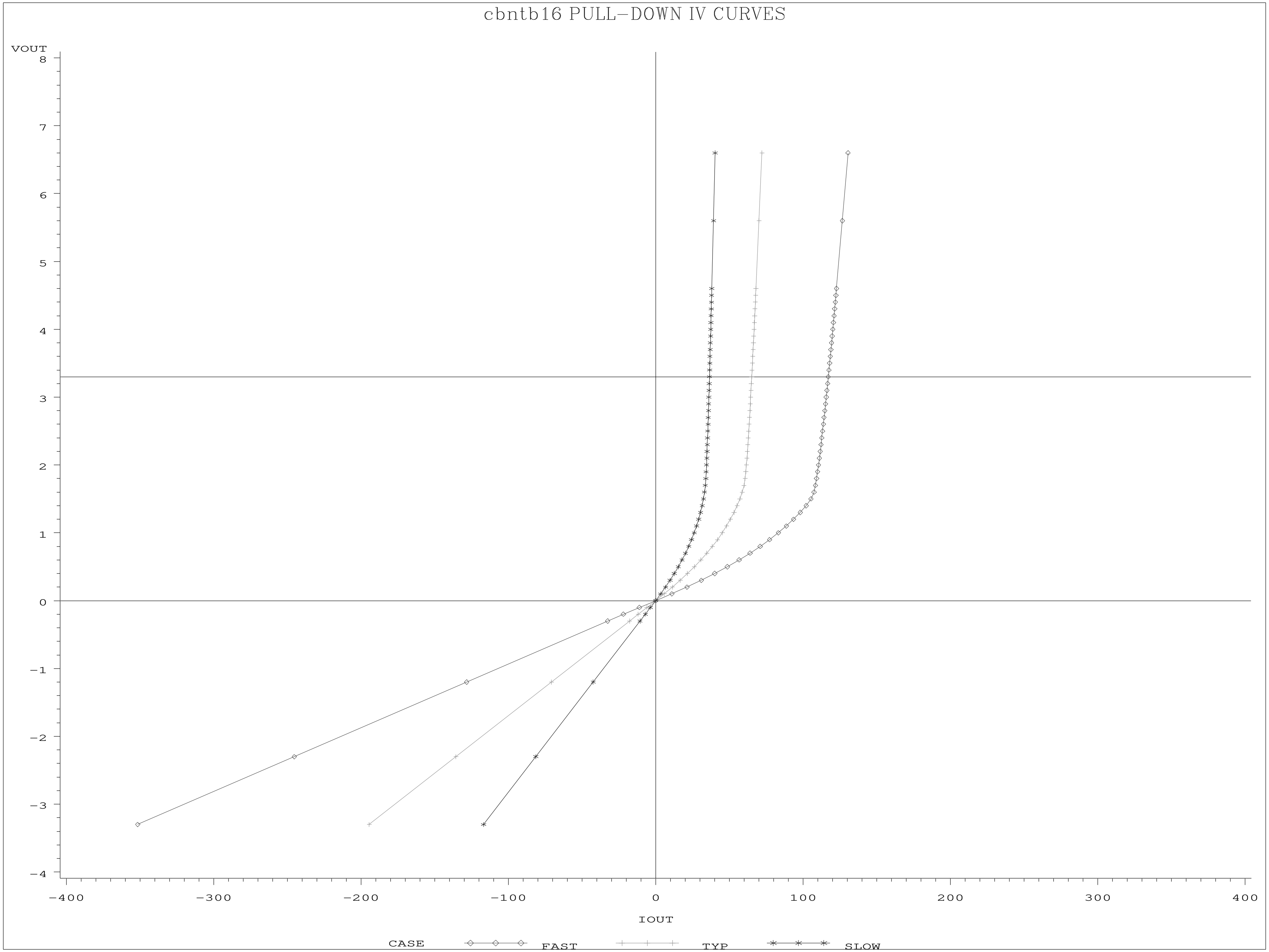
cbntb11 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

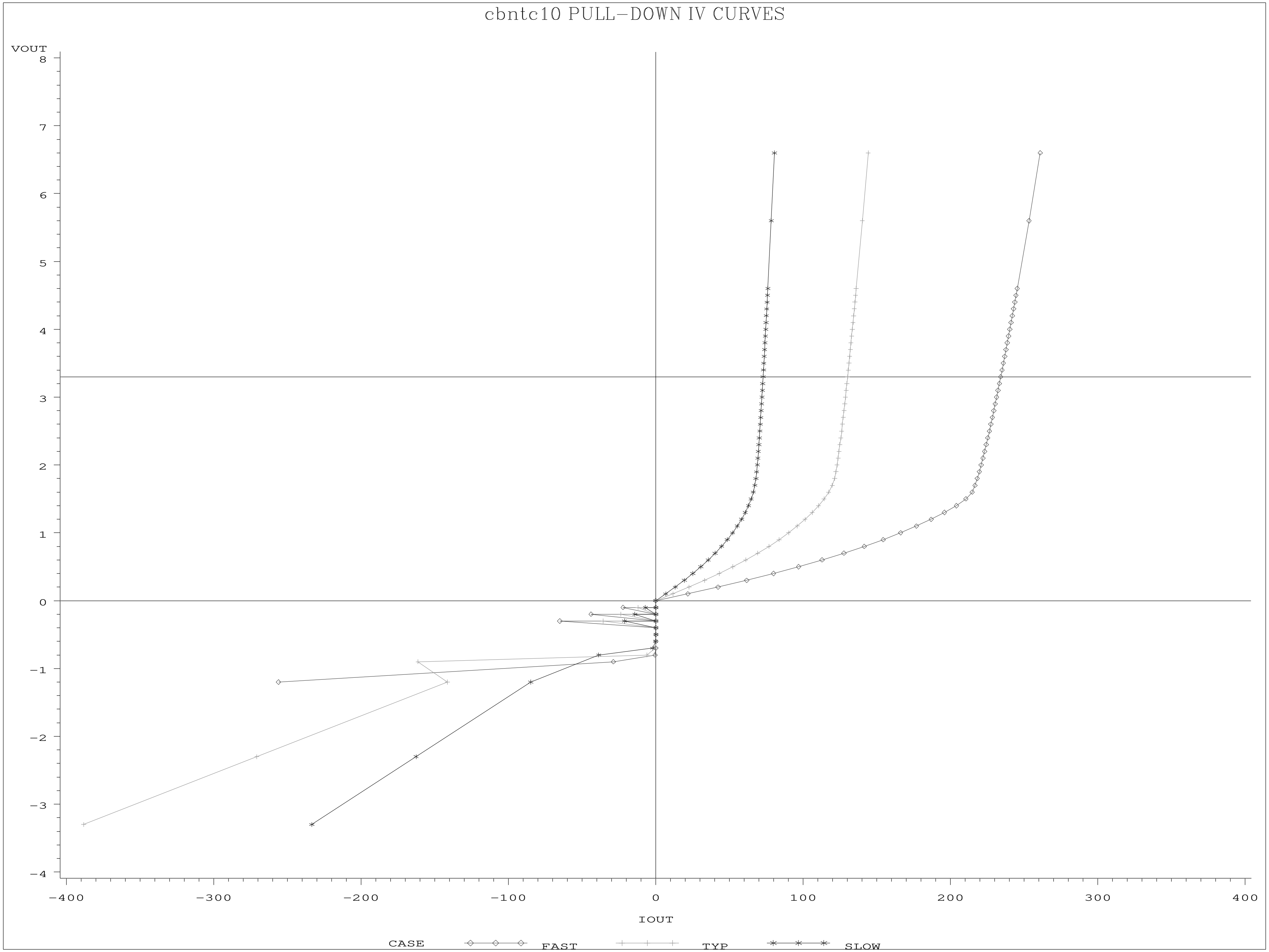

cbntb12 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW


cbntb12 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

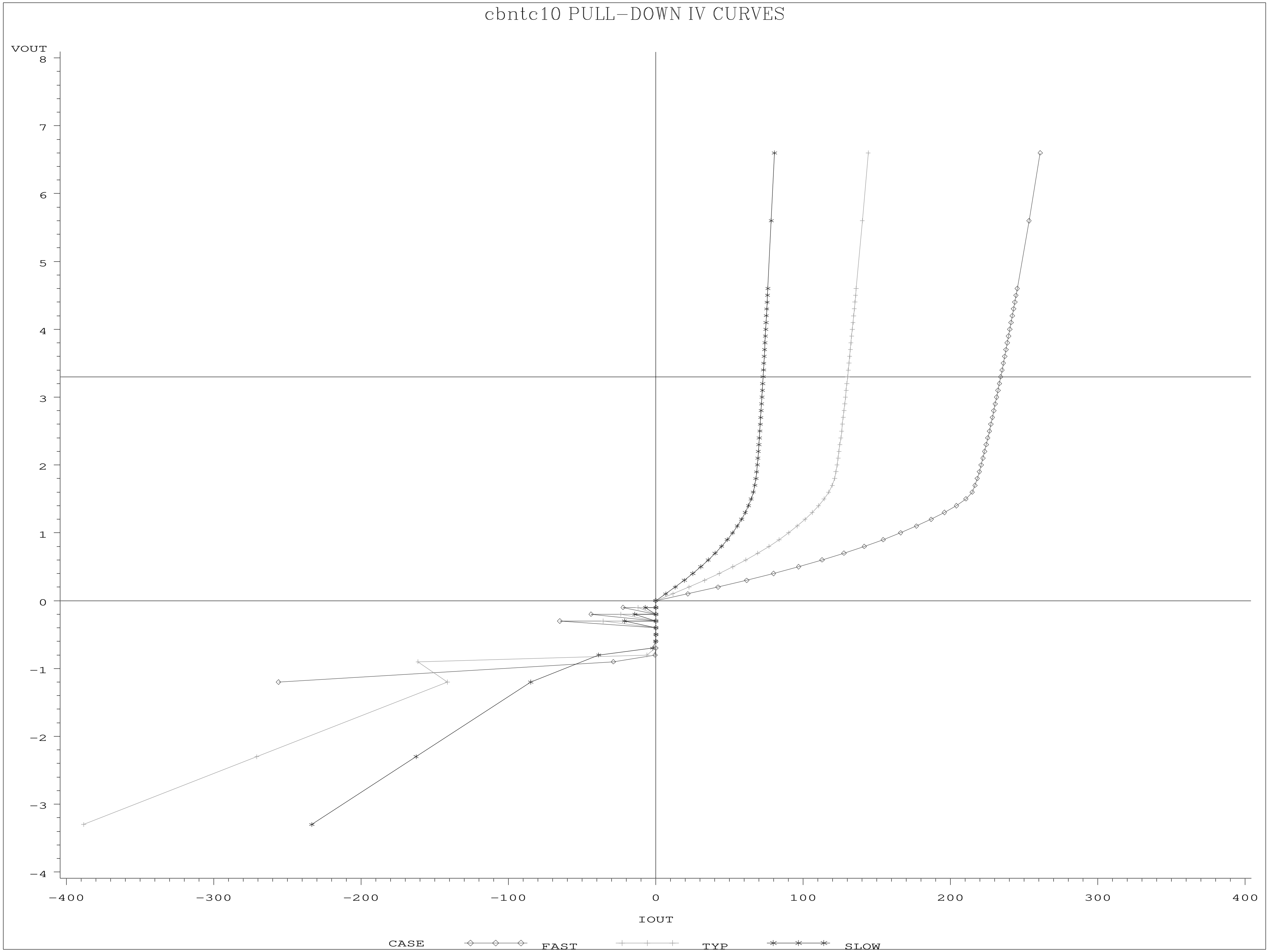


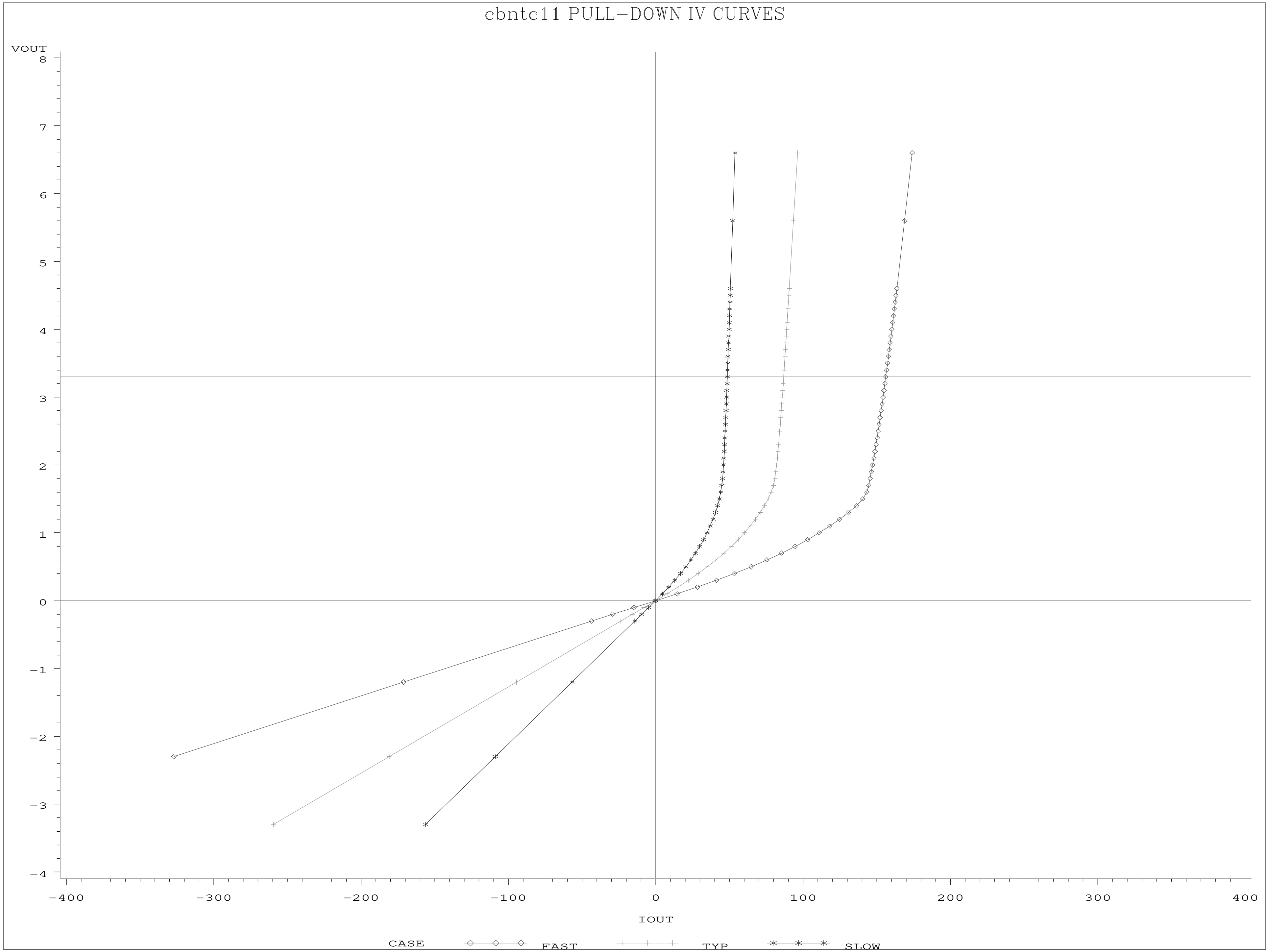
cbntb16 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

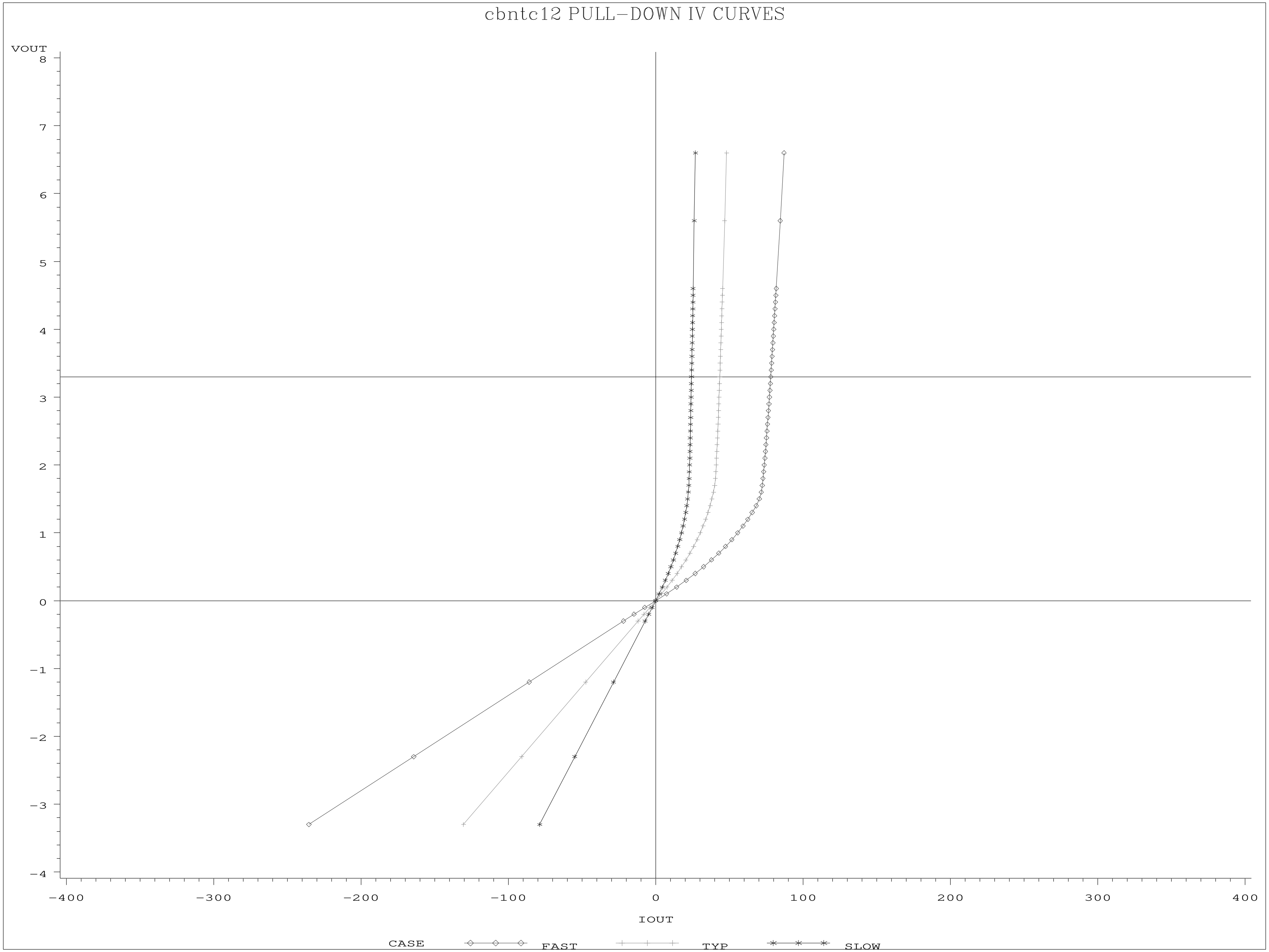

cbntb16 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

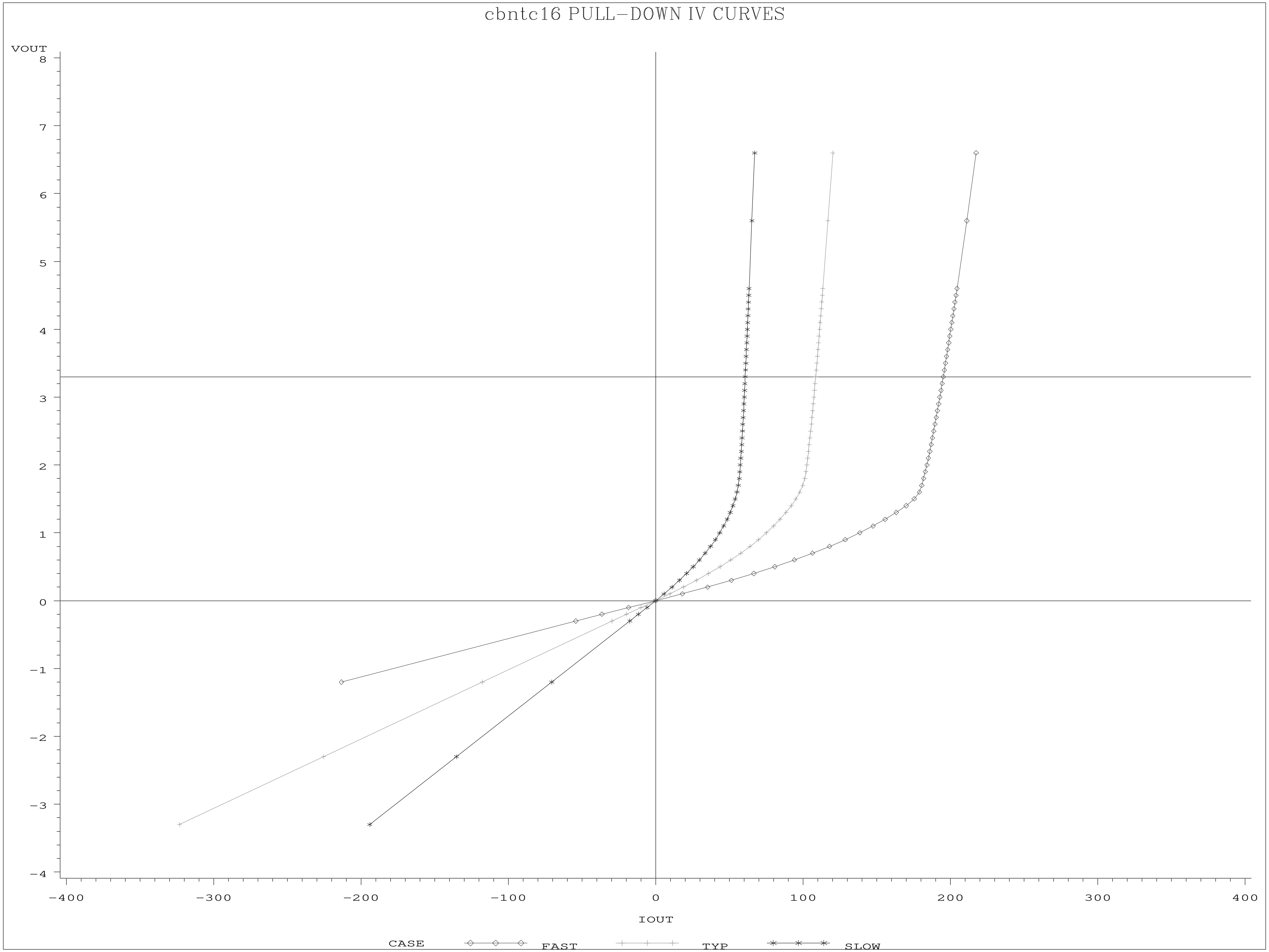


cbntc10 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW


cbntc10 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

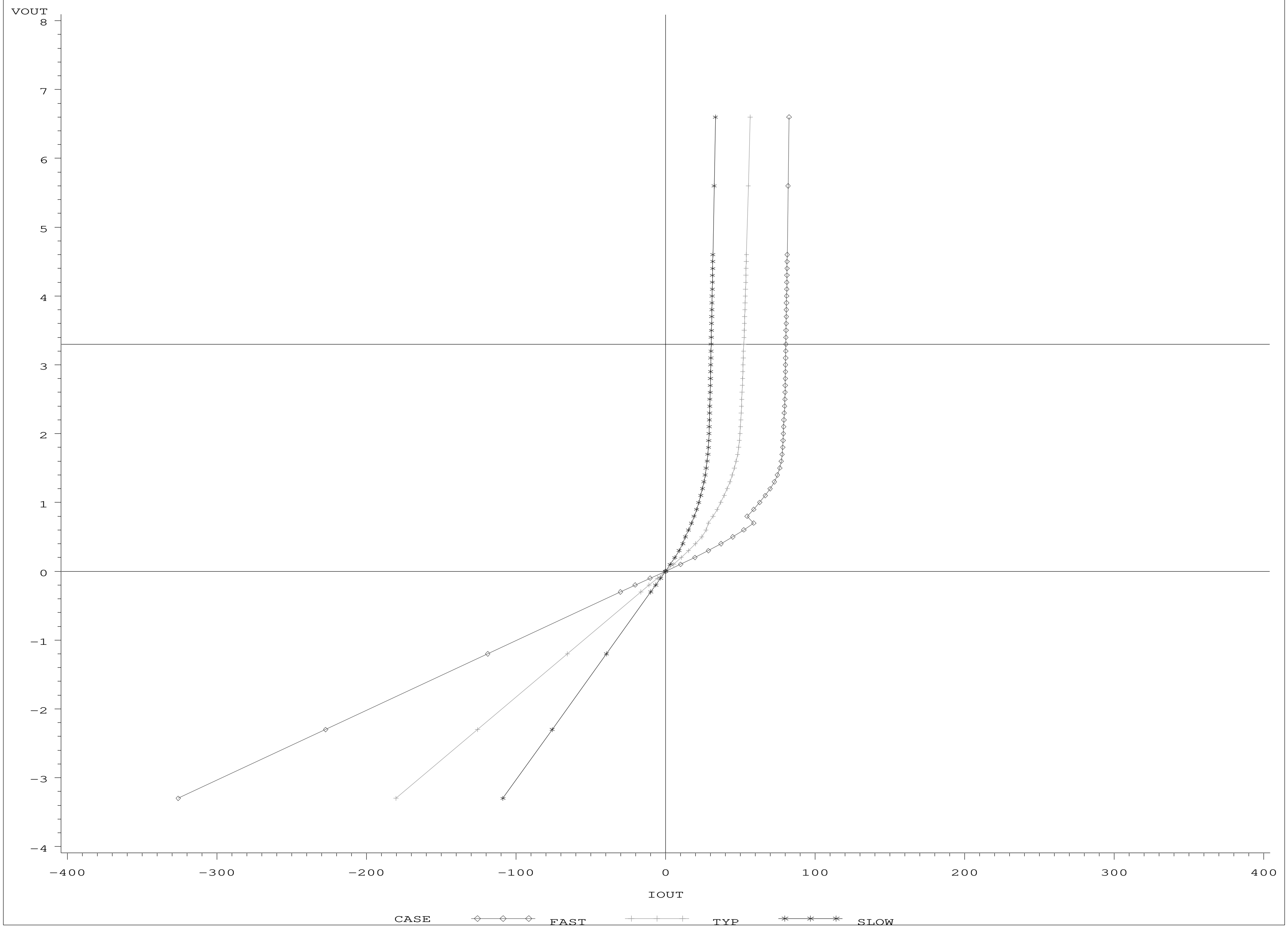

cbntc11 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

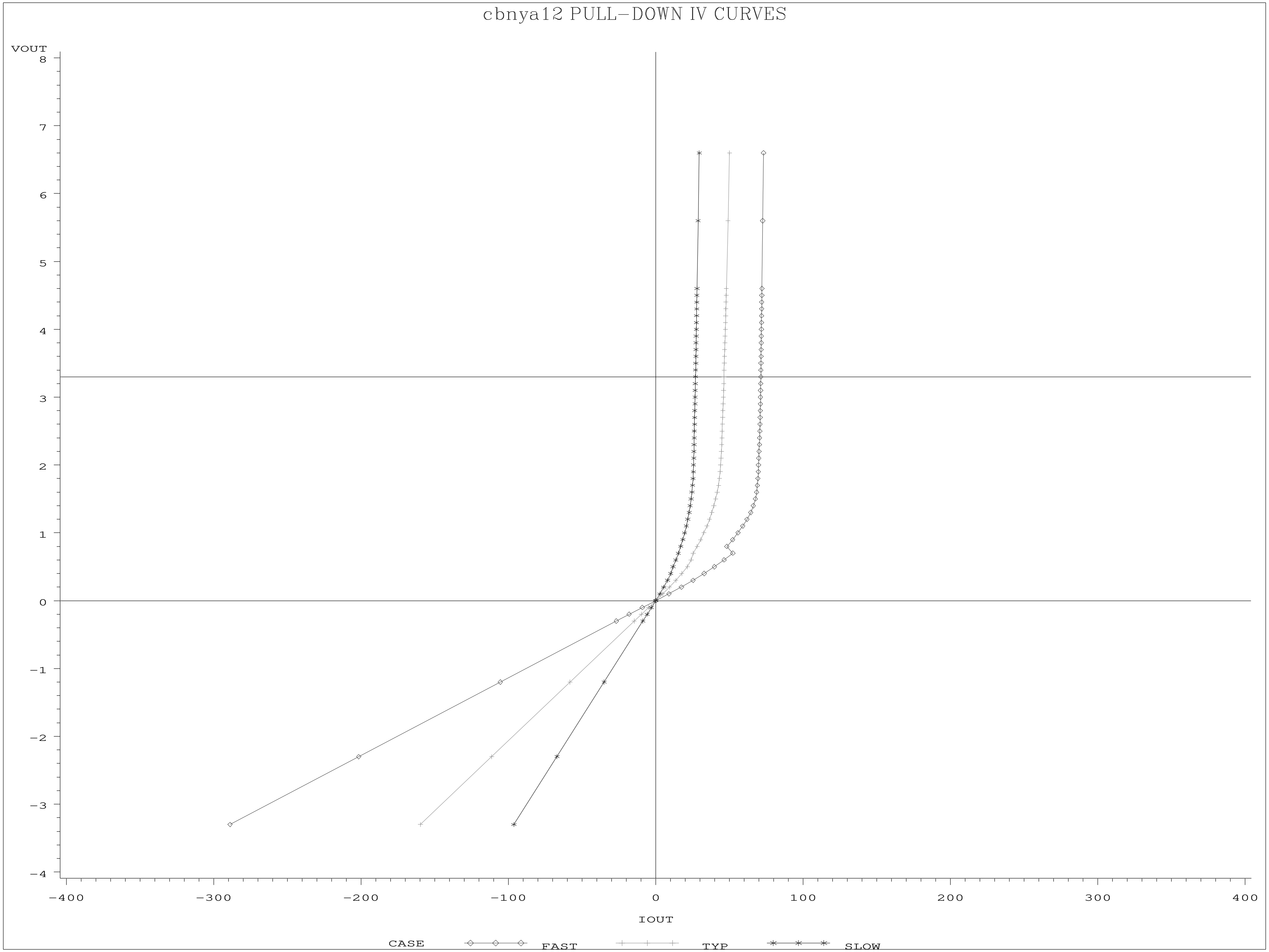

cbntc12 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW



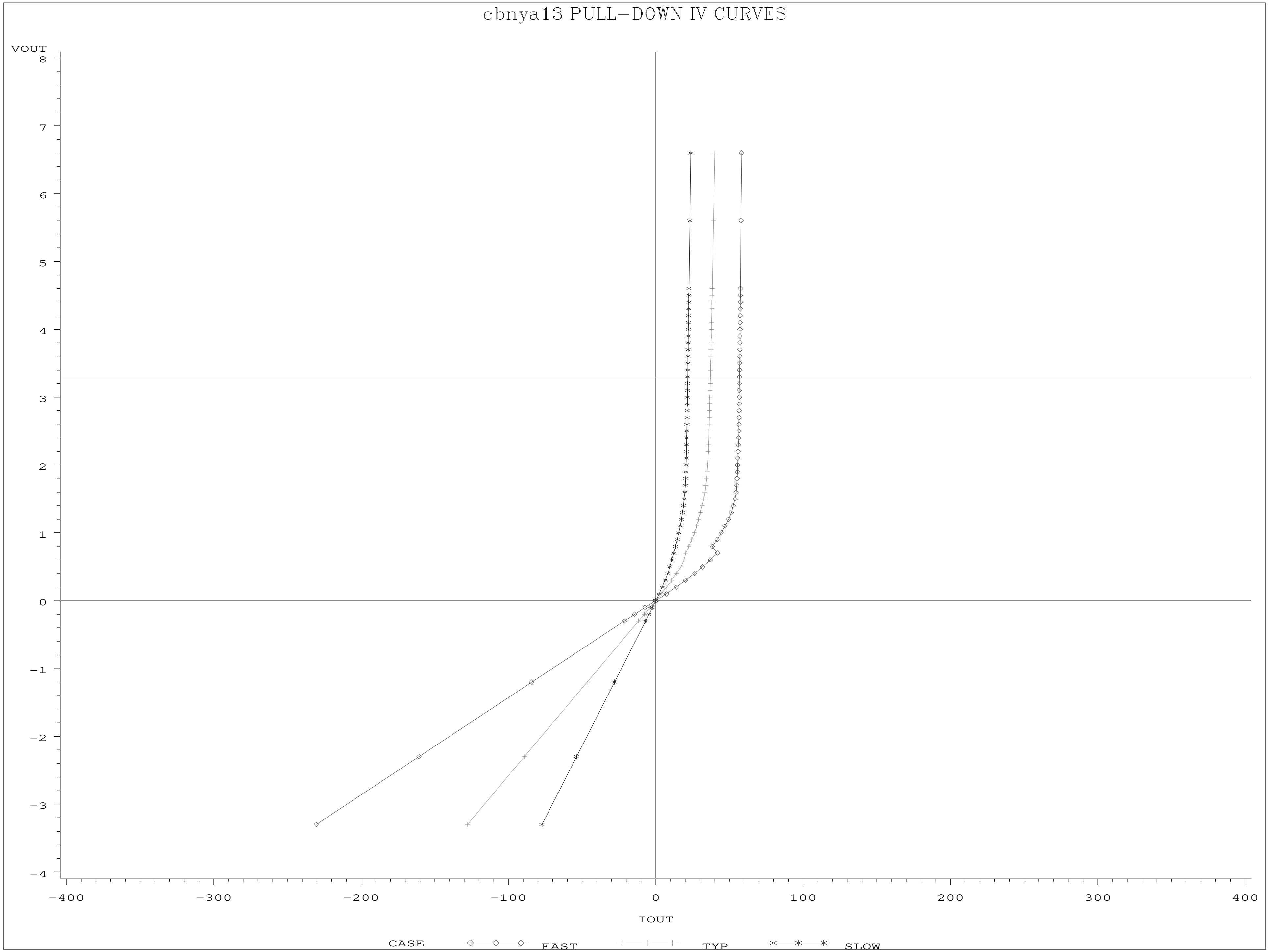
cbntc16 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

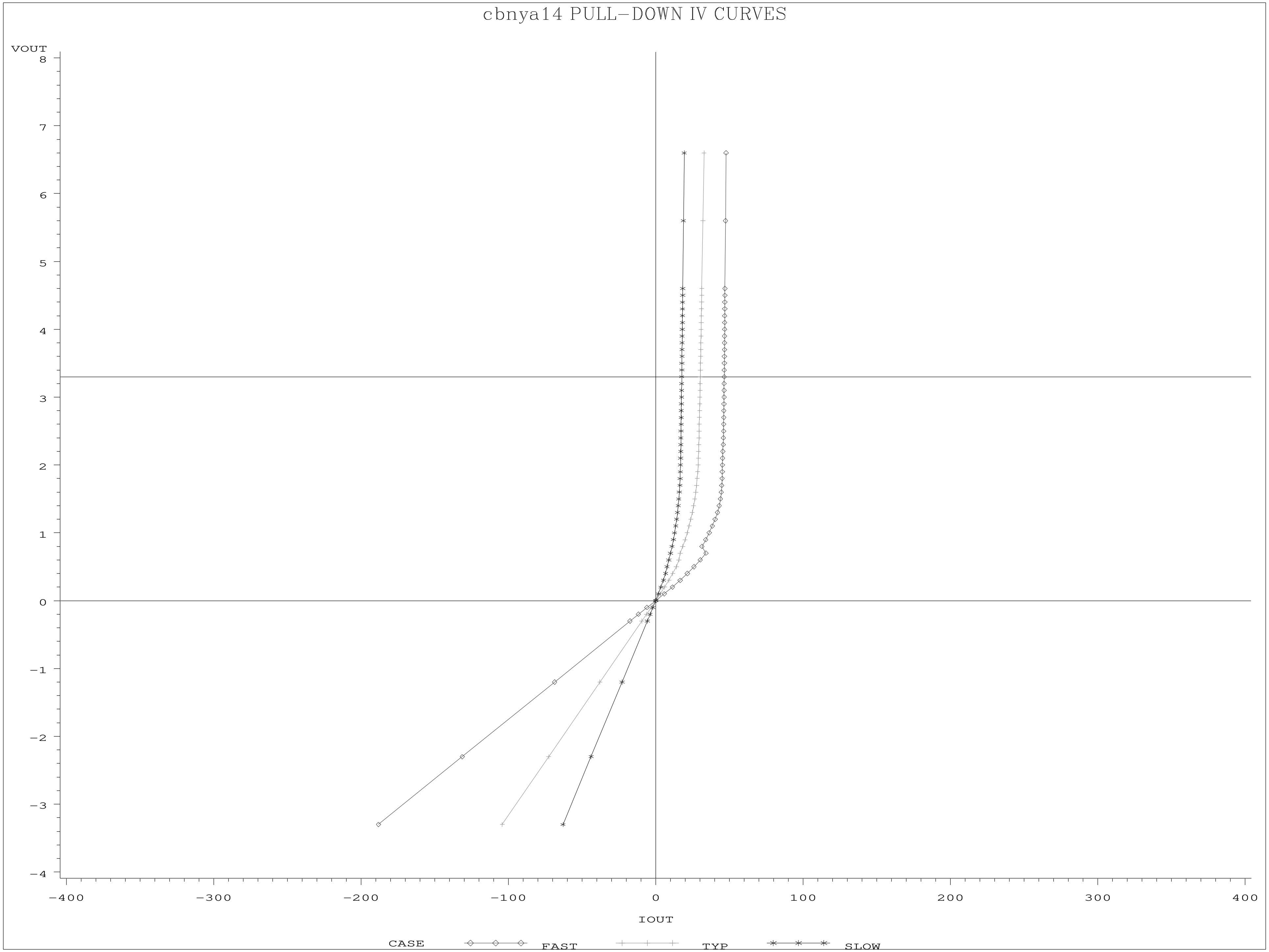
cbnya11 PULL-DOWN IV CURVES

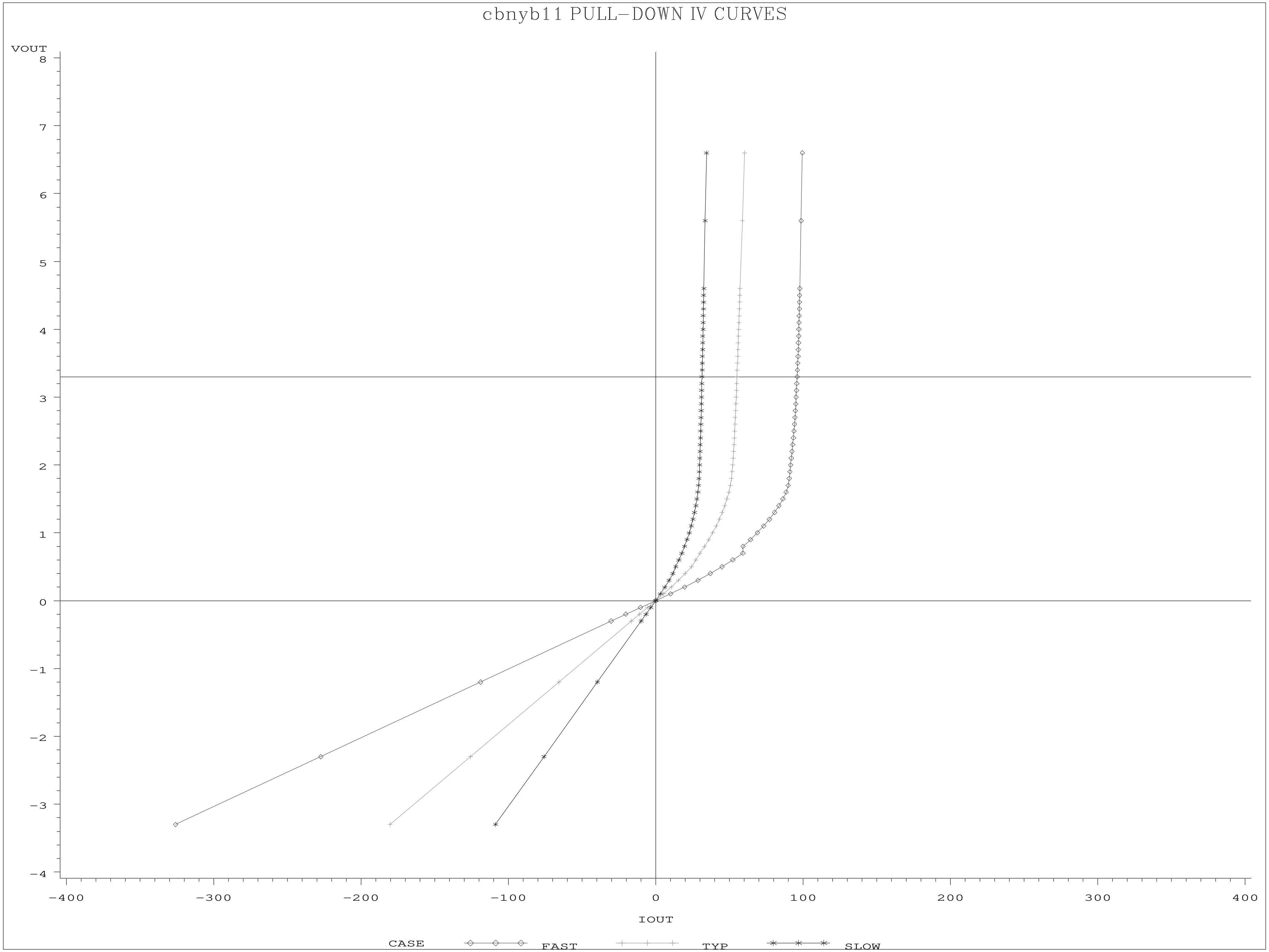


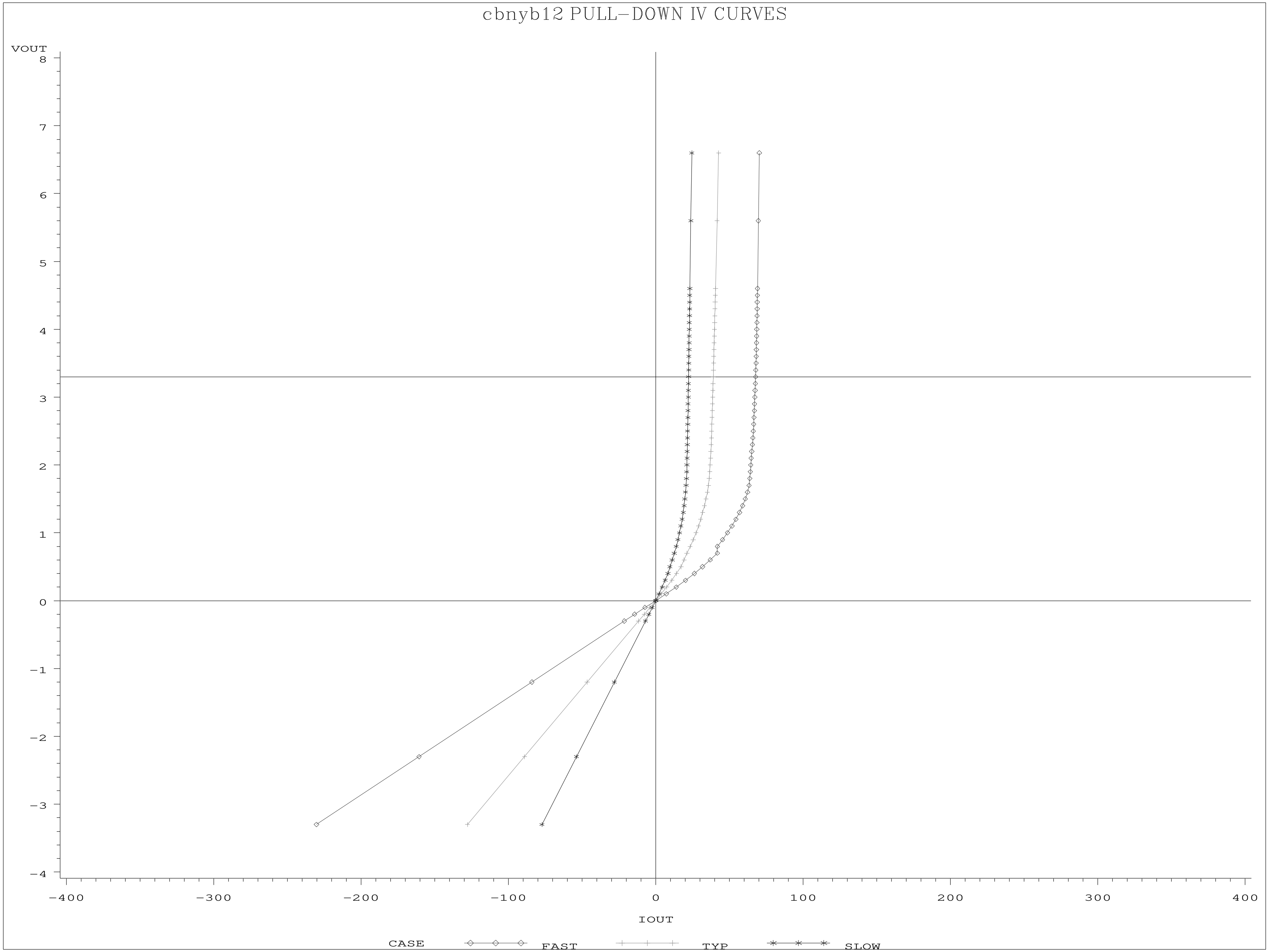


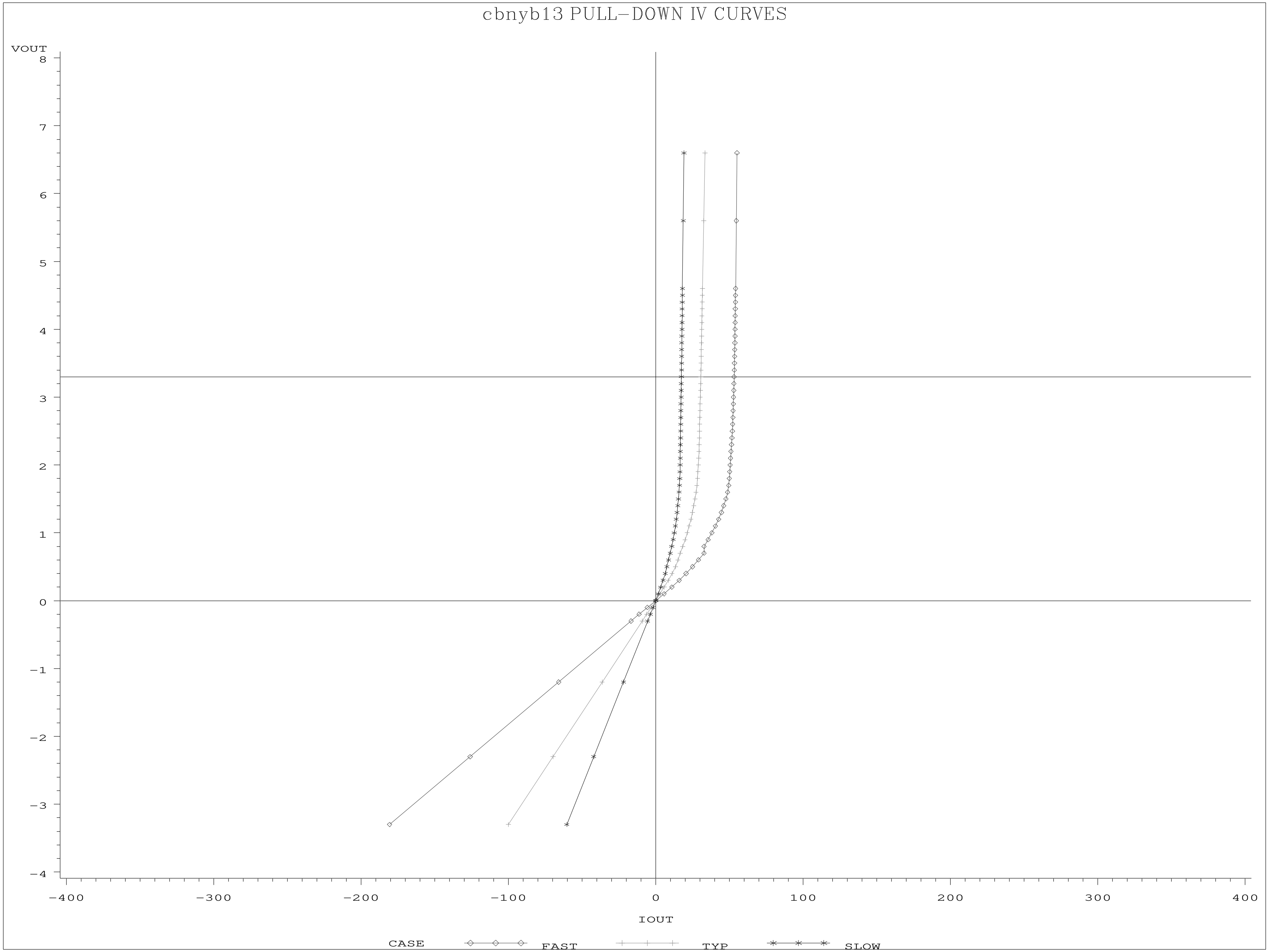
cbnya12 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW


cbnya13 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

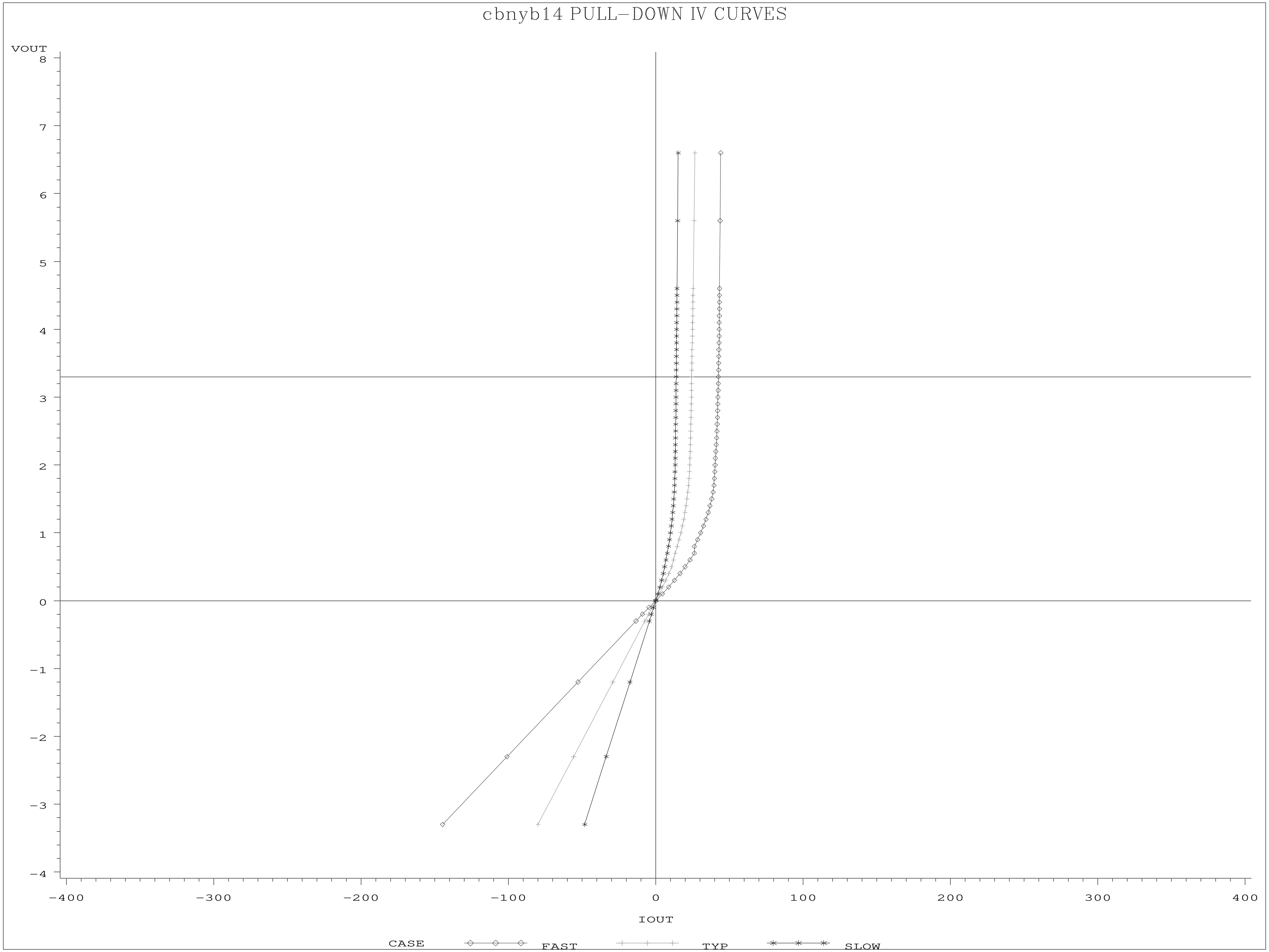

cbnya14 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

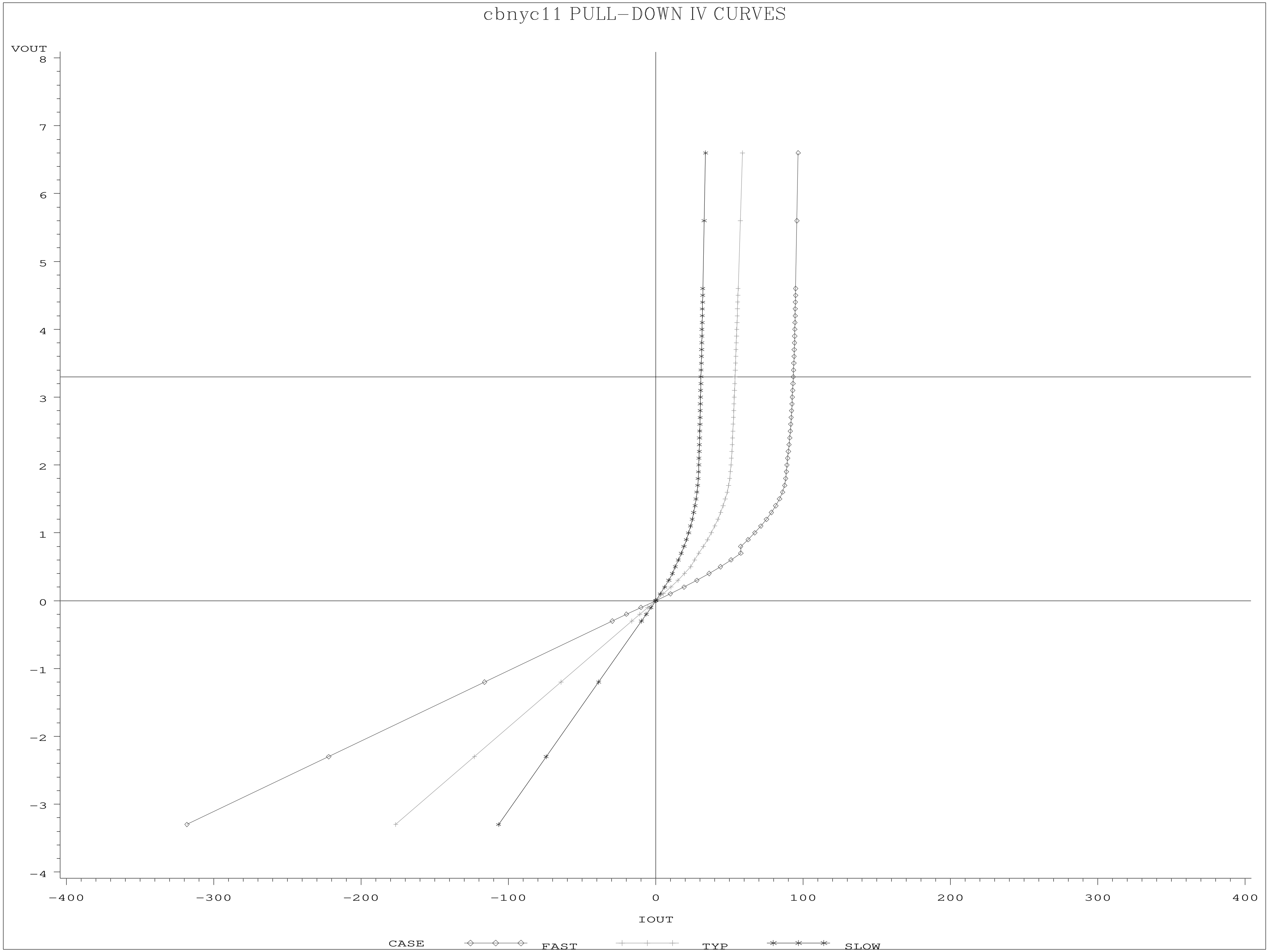

cbnyb11 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

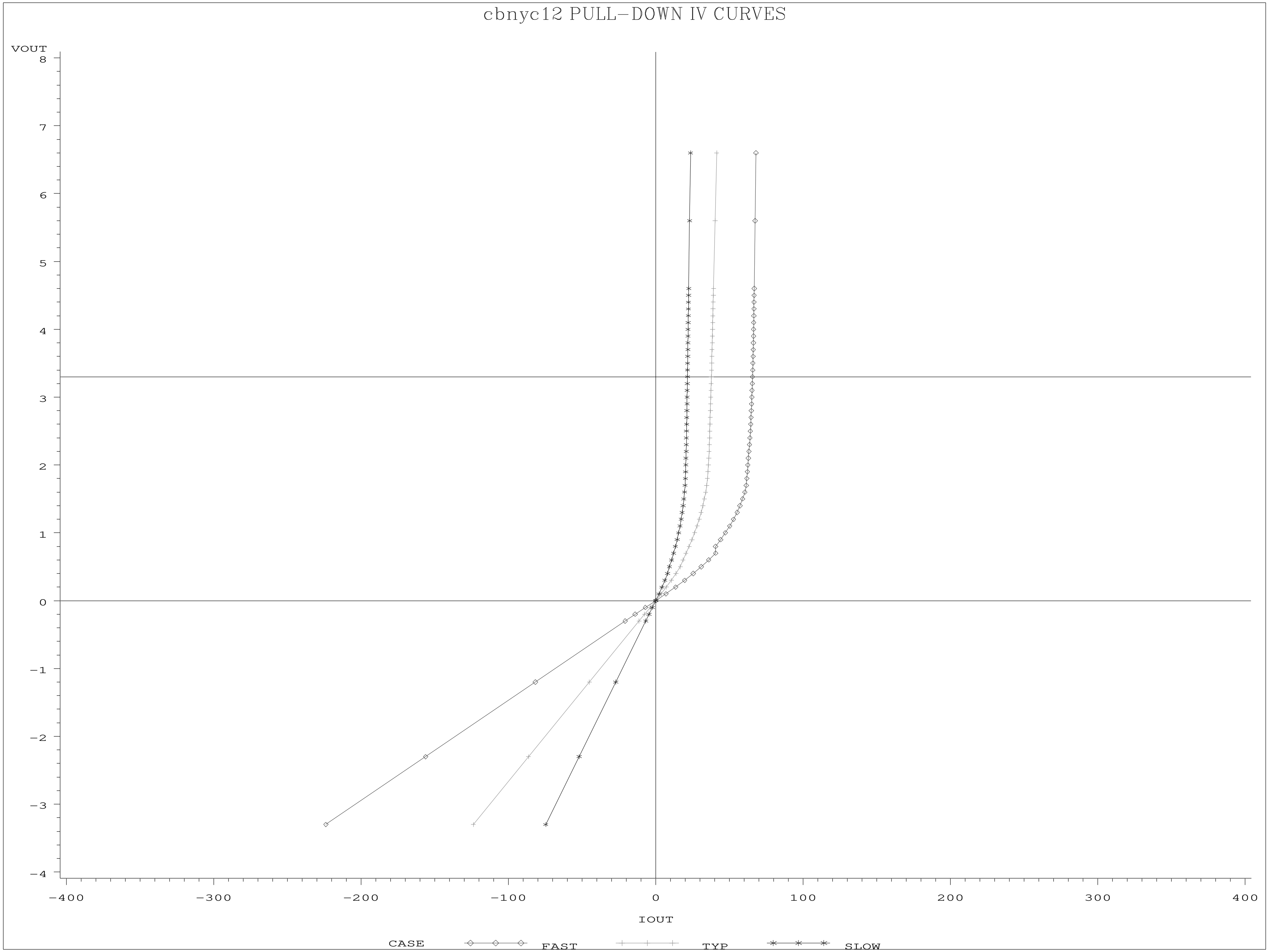

cbnyb12 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

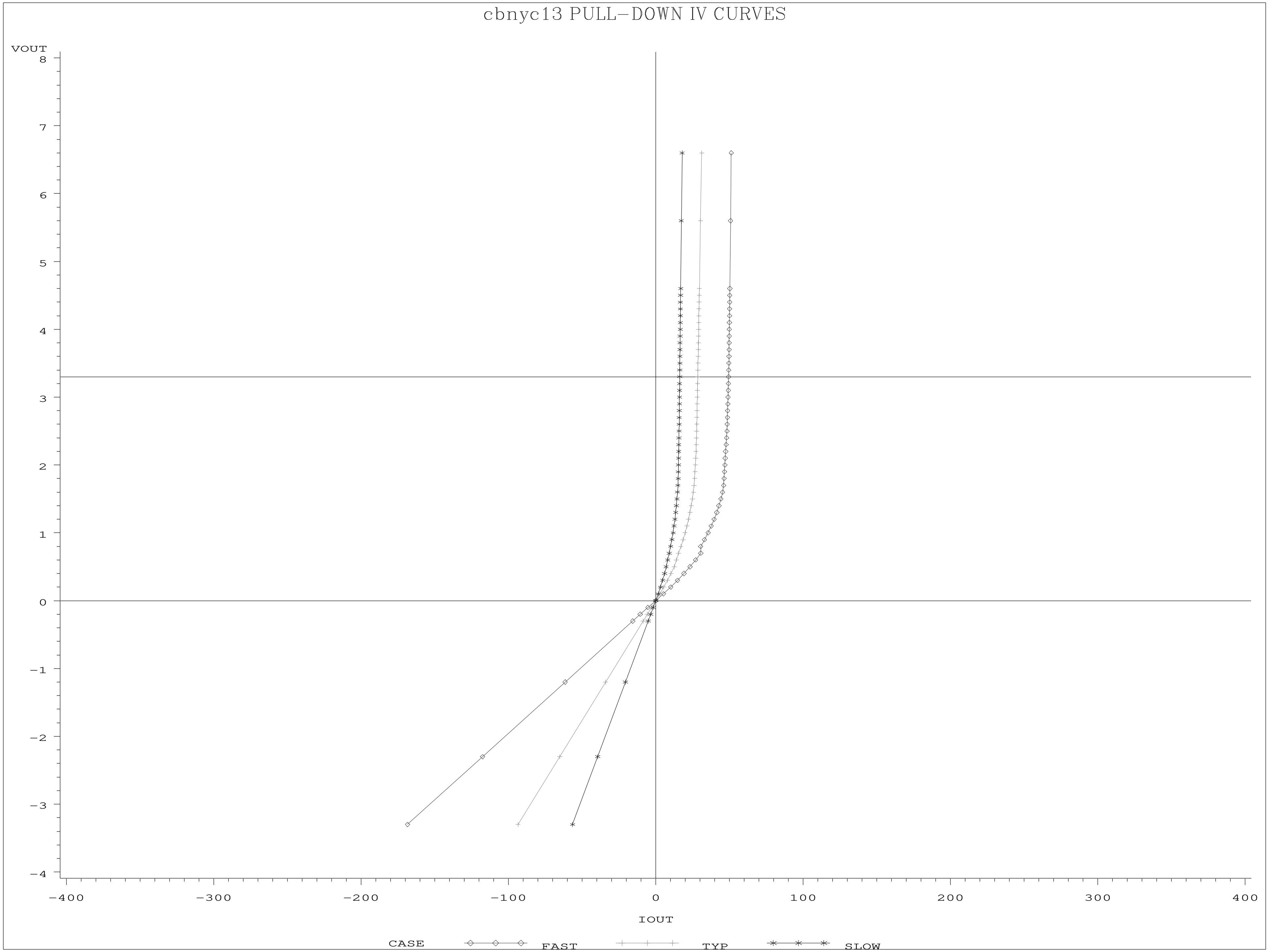


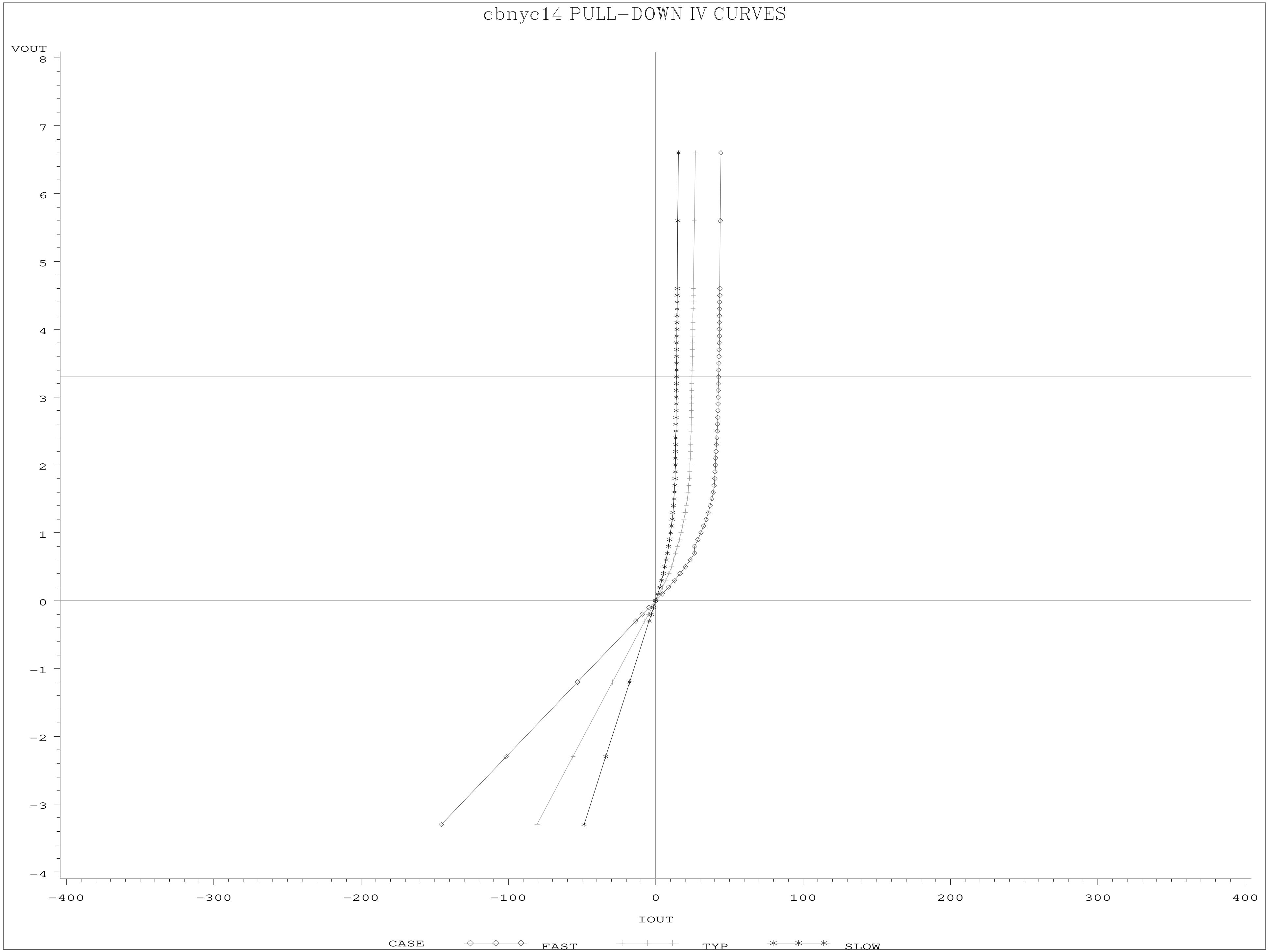
cbnyb13 PULL−DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

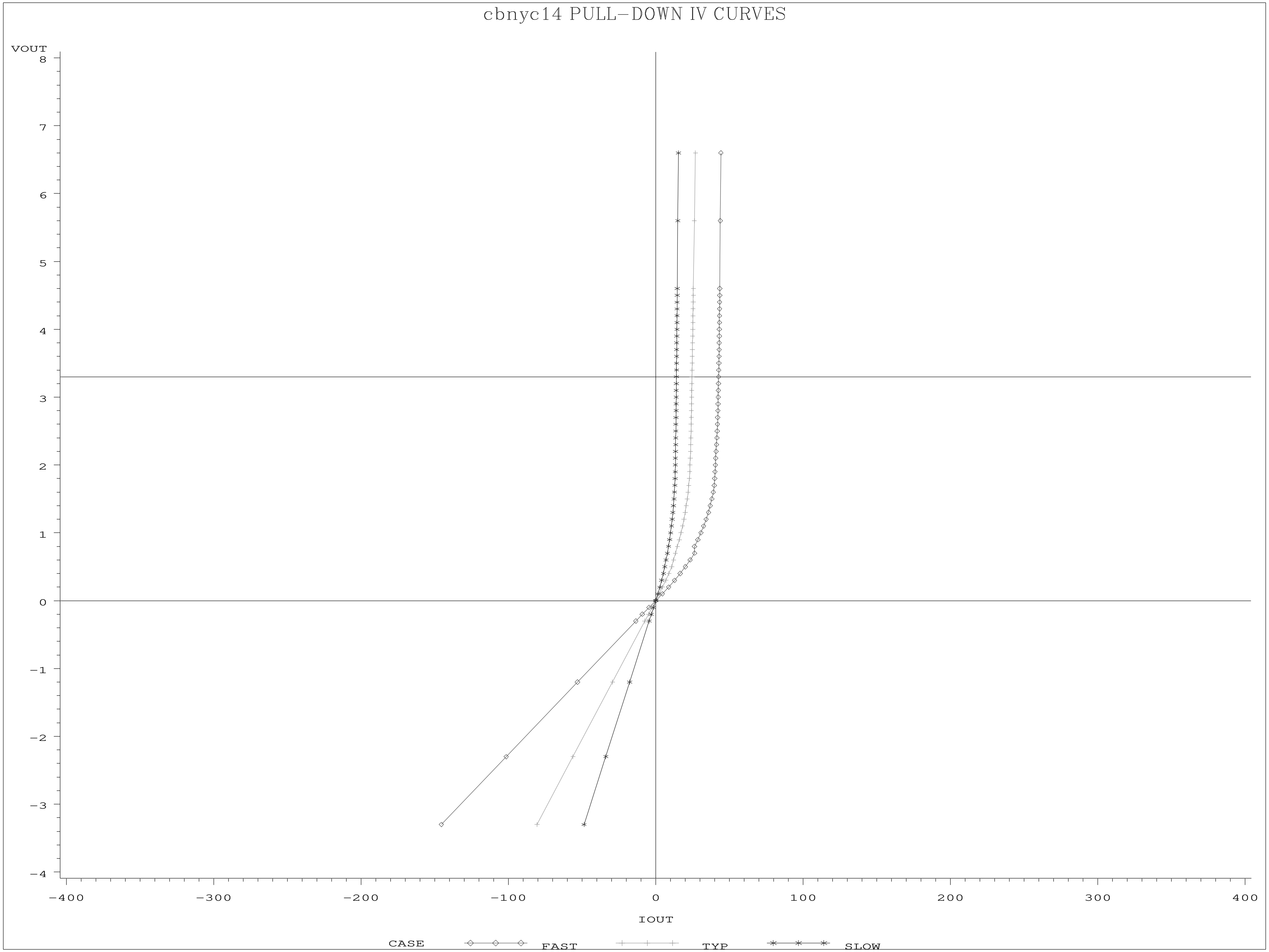

cbnyb14 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW


cbnyc11 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

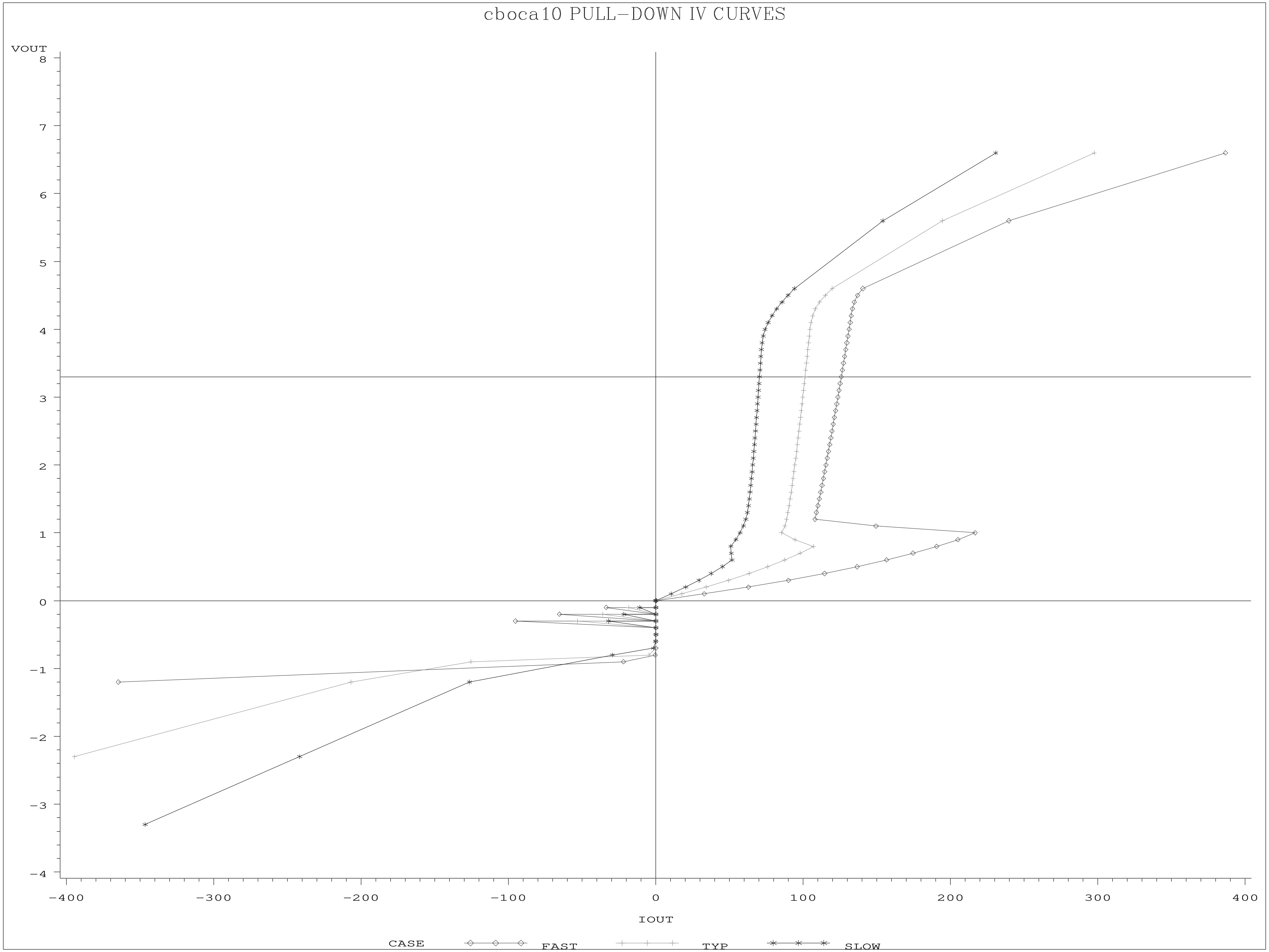

cbnyc12 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

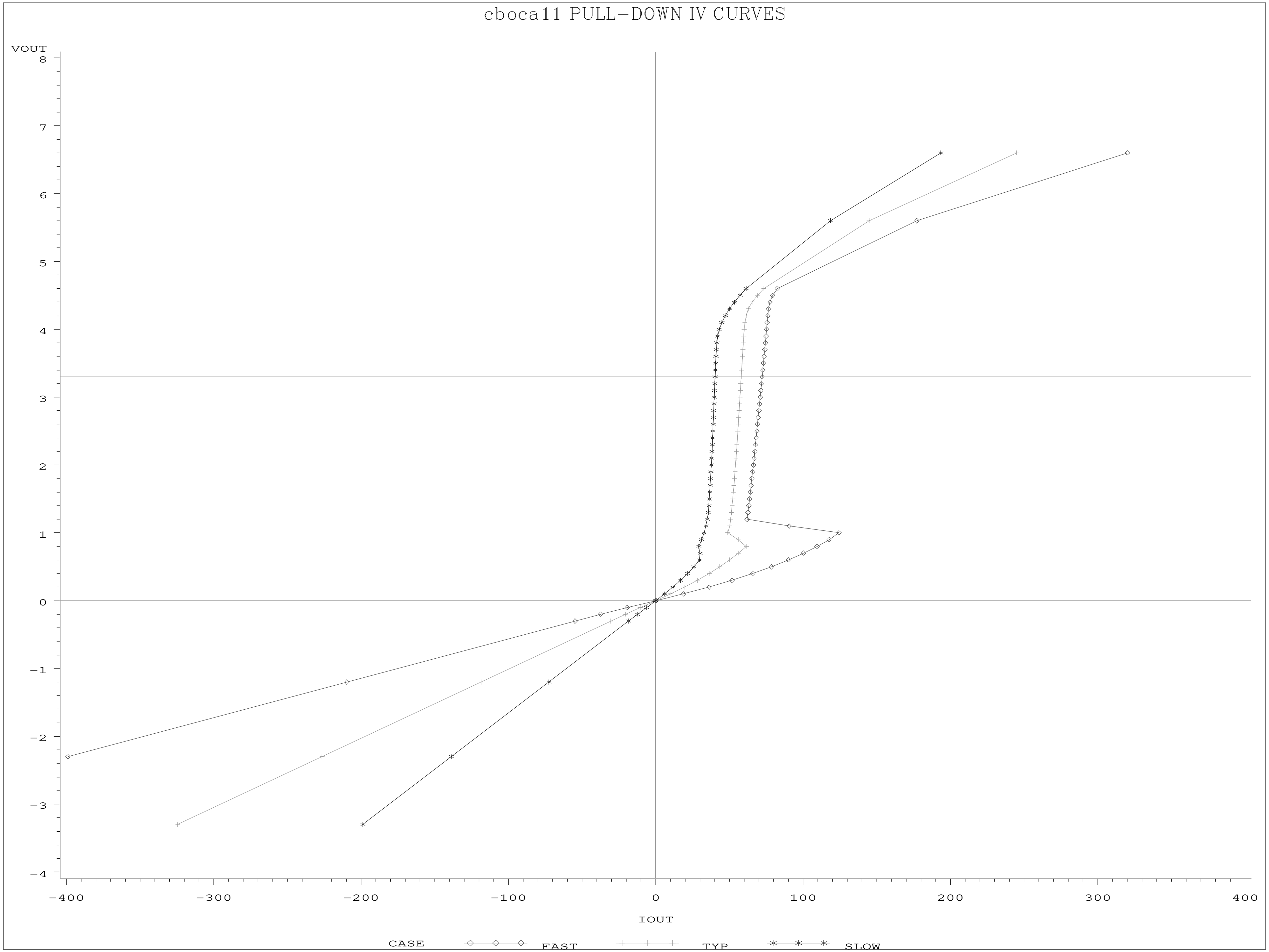

cbnyc13 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

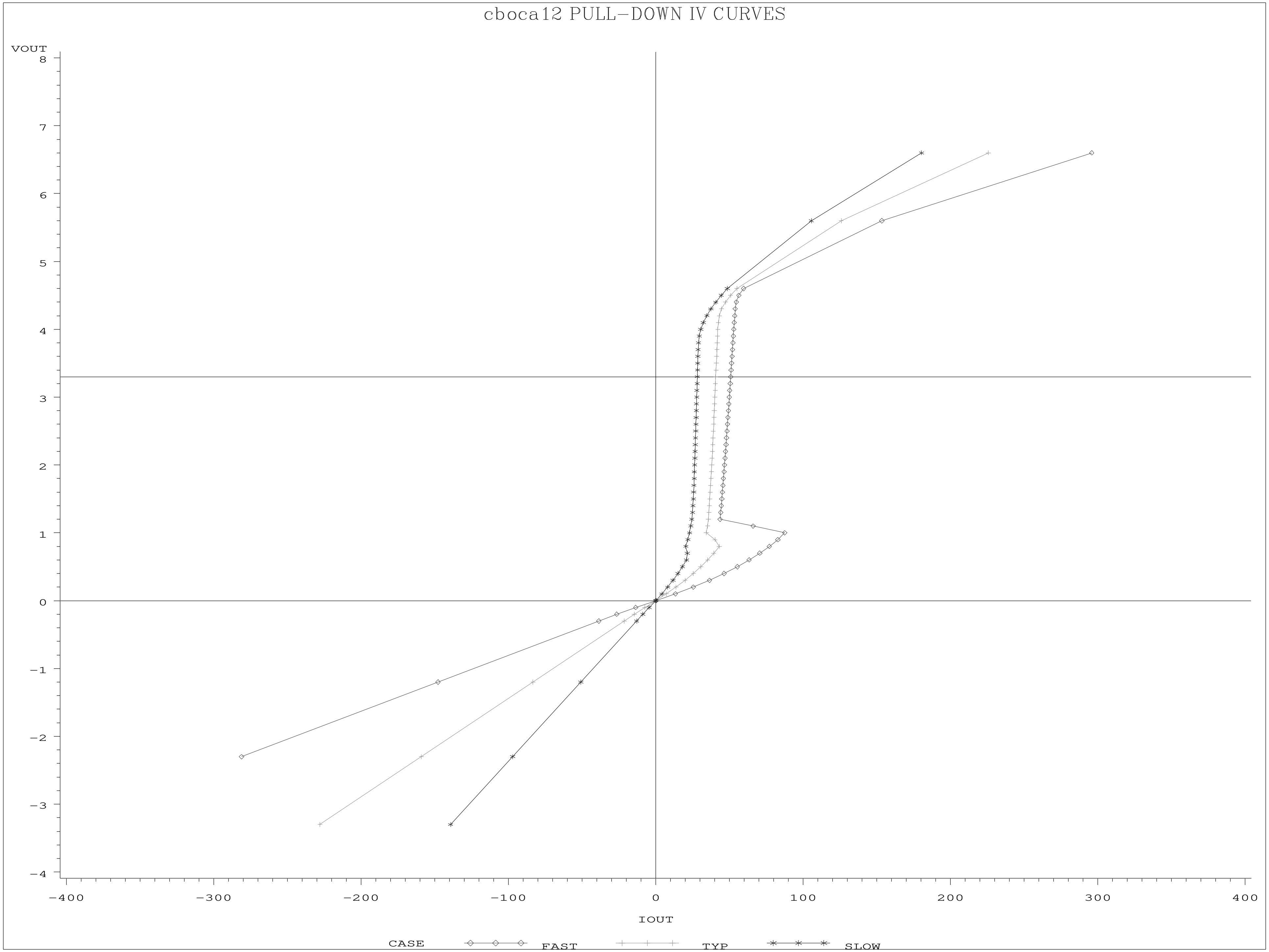

cbnyc14 PULL−DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

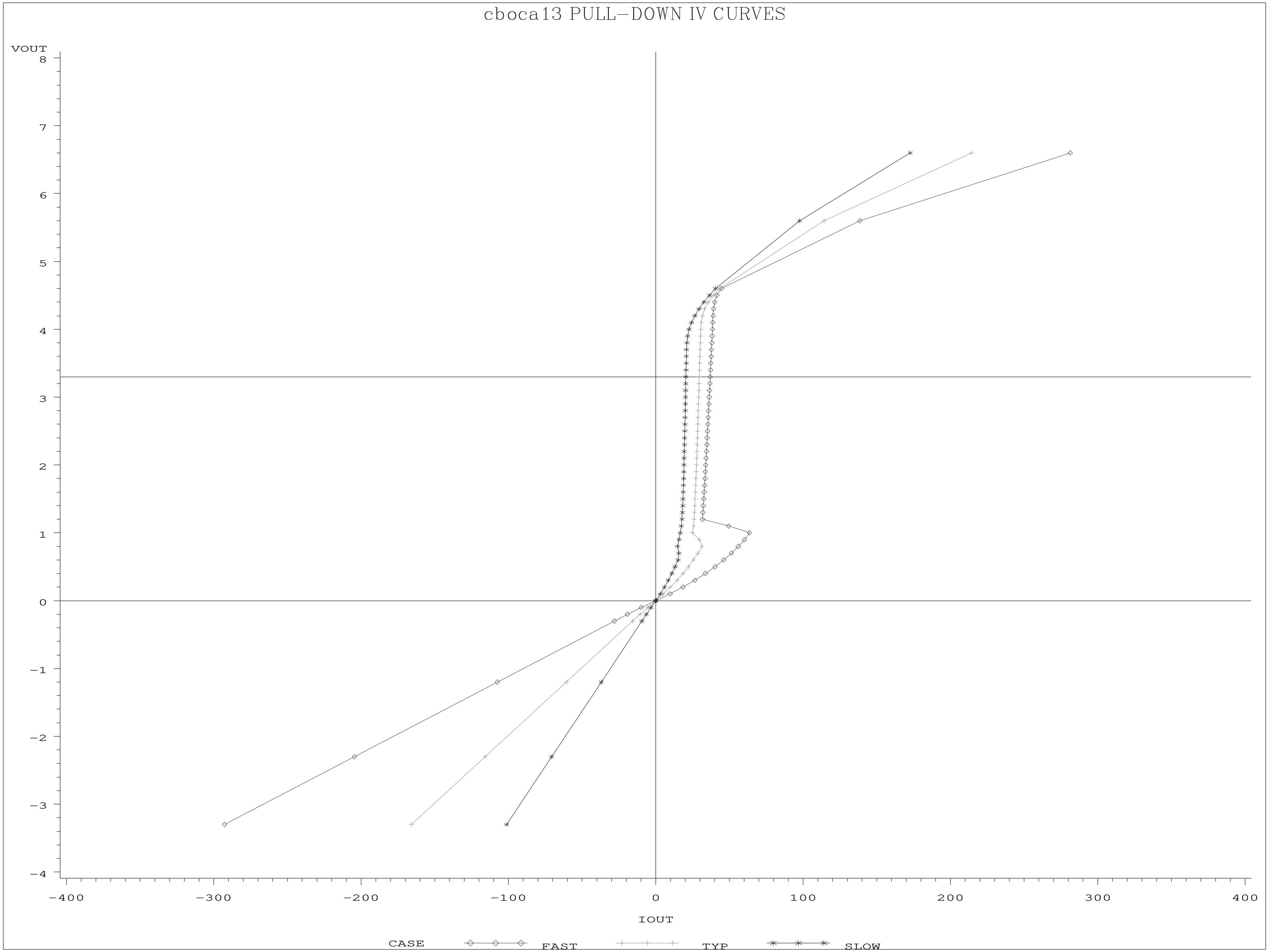


cbnyc14 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

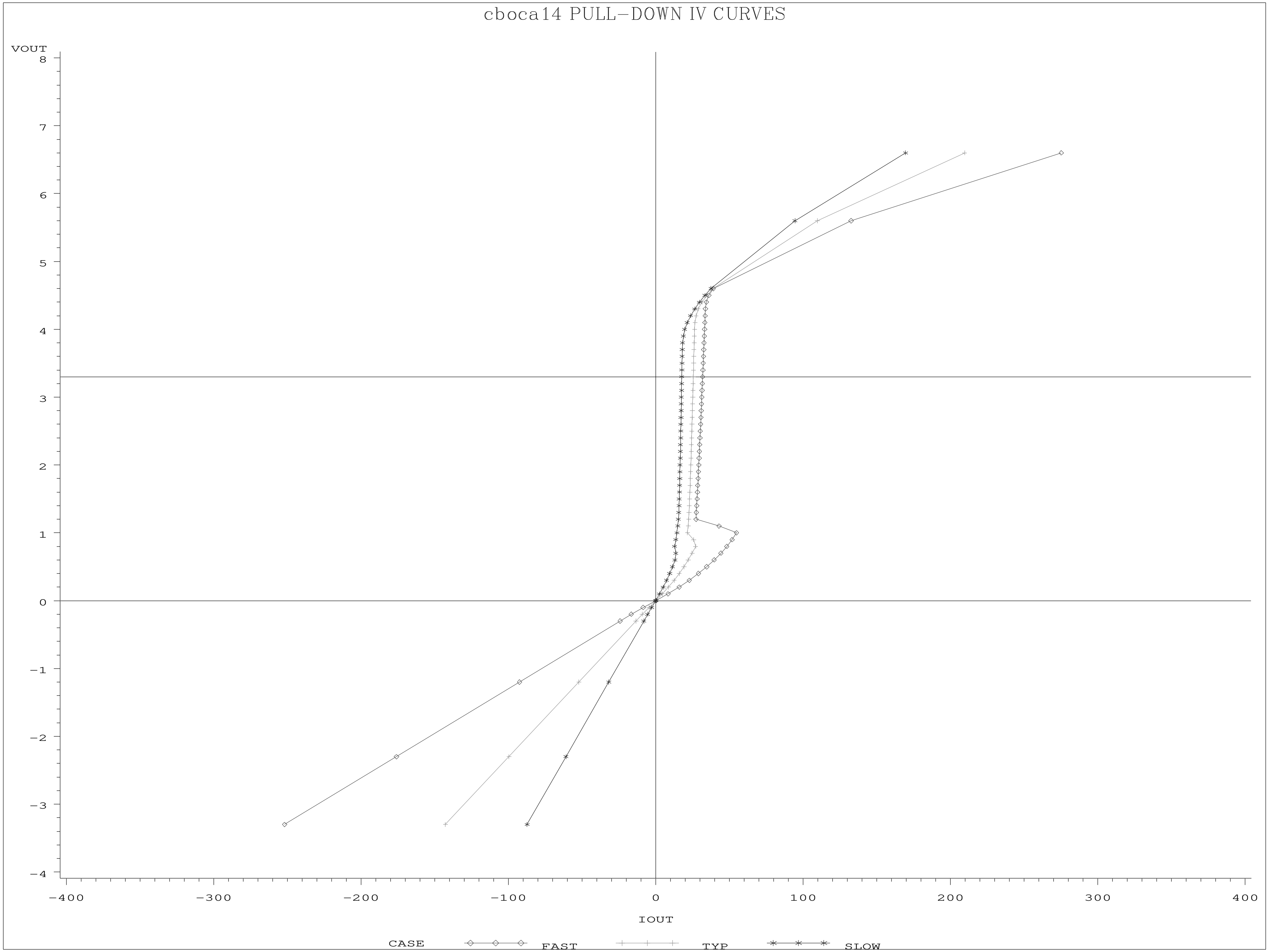

cboca10 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

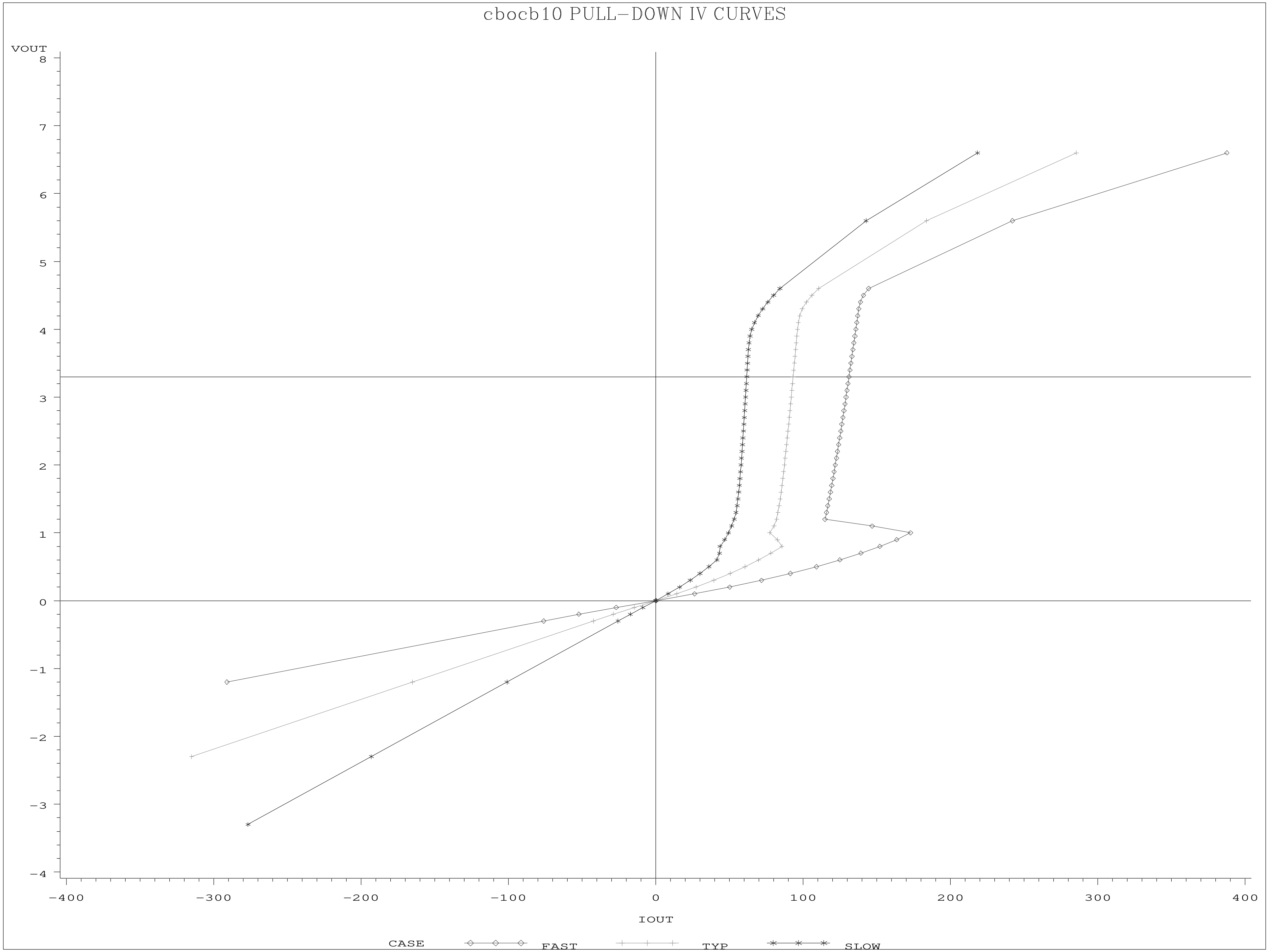

cboca11 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

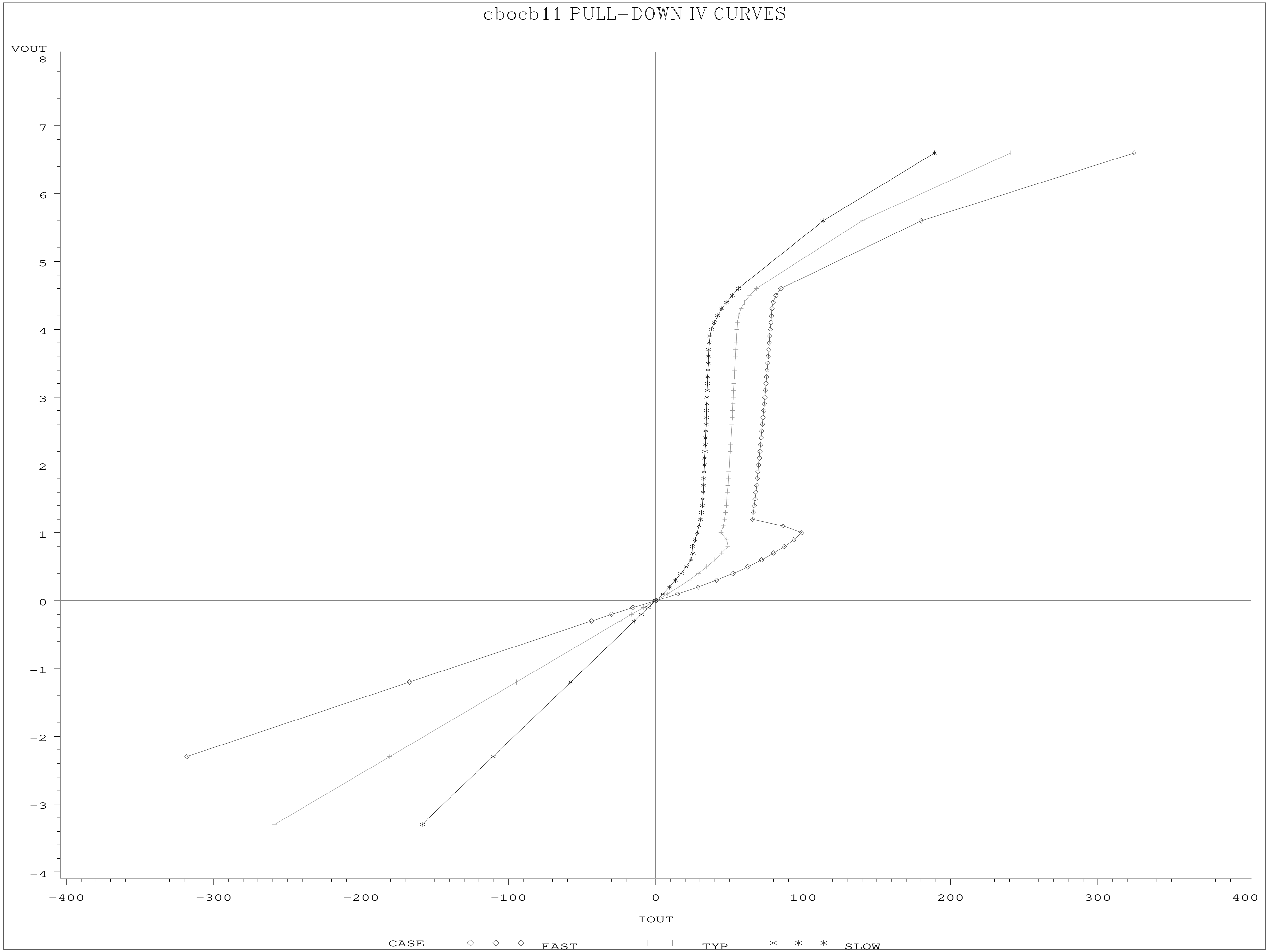

cboca12 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

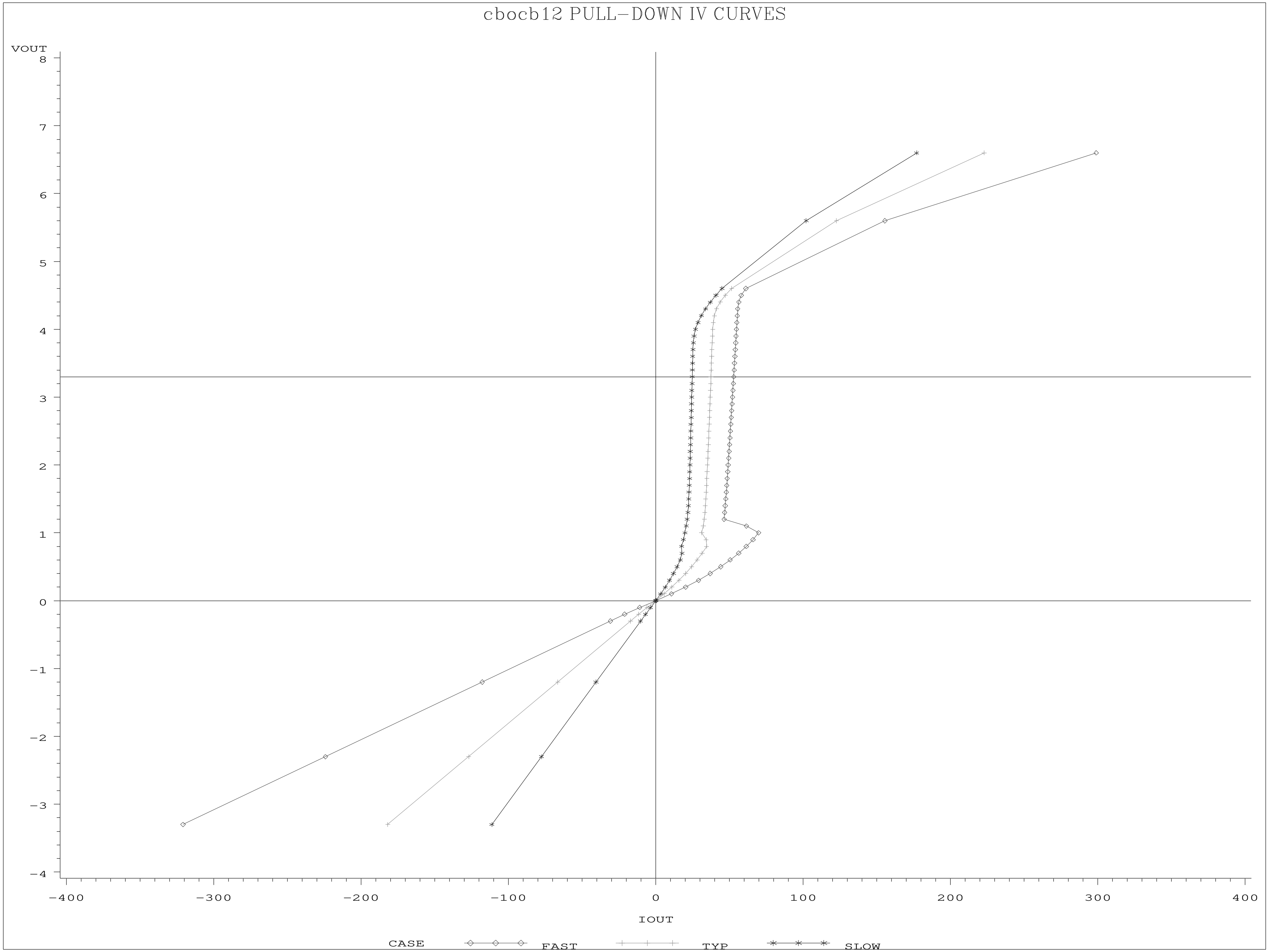


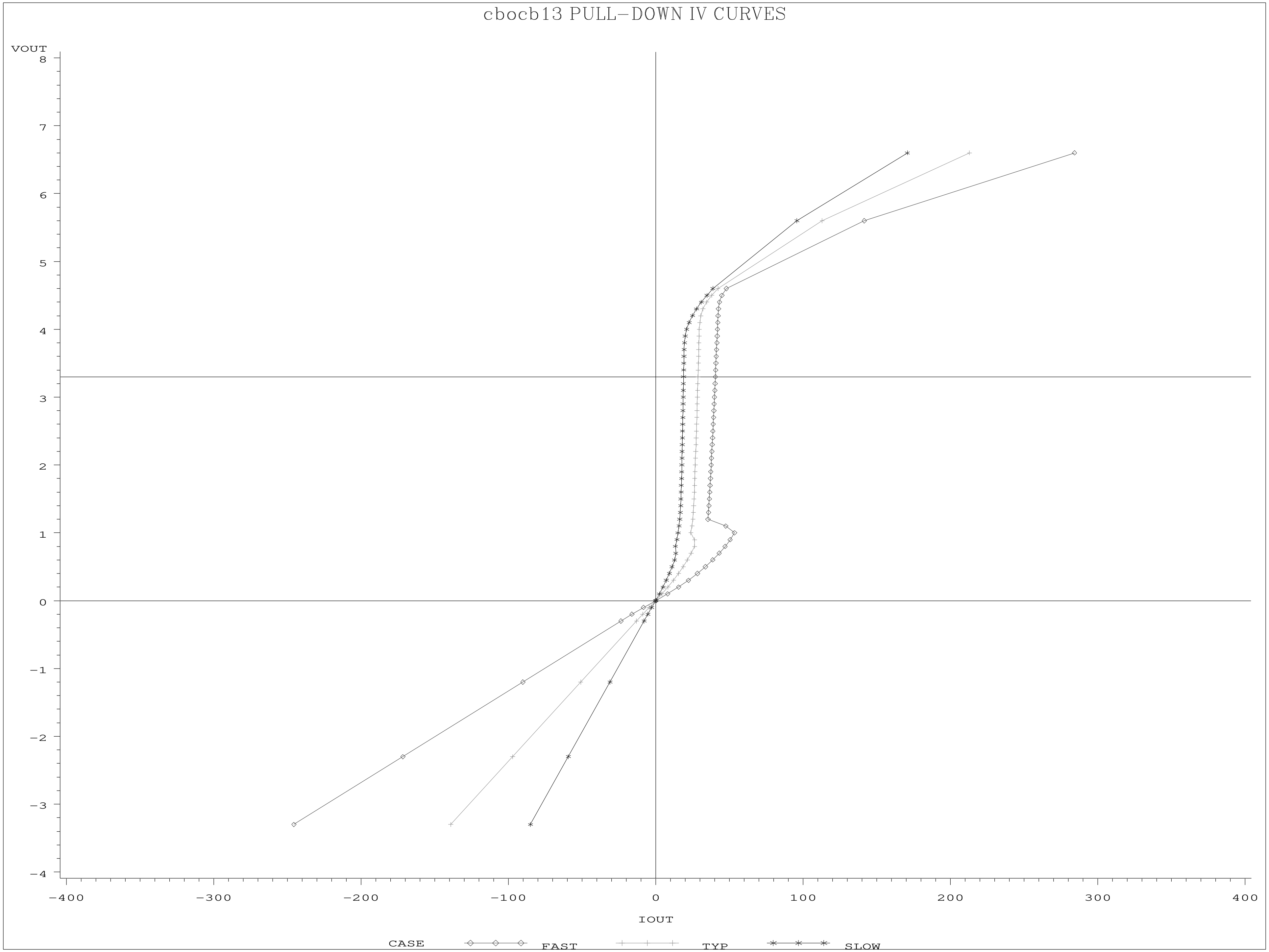
cboca13 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

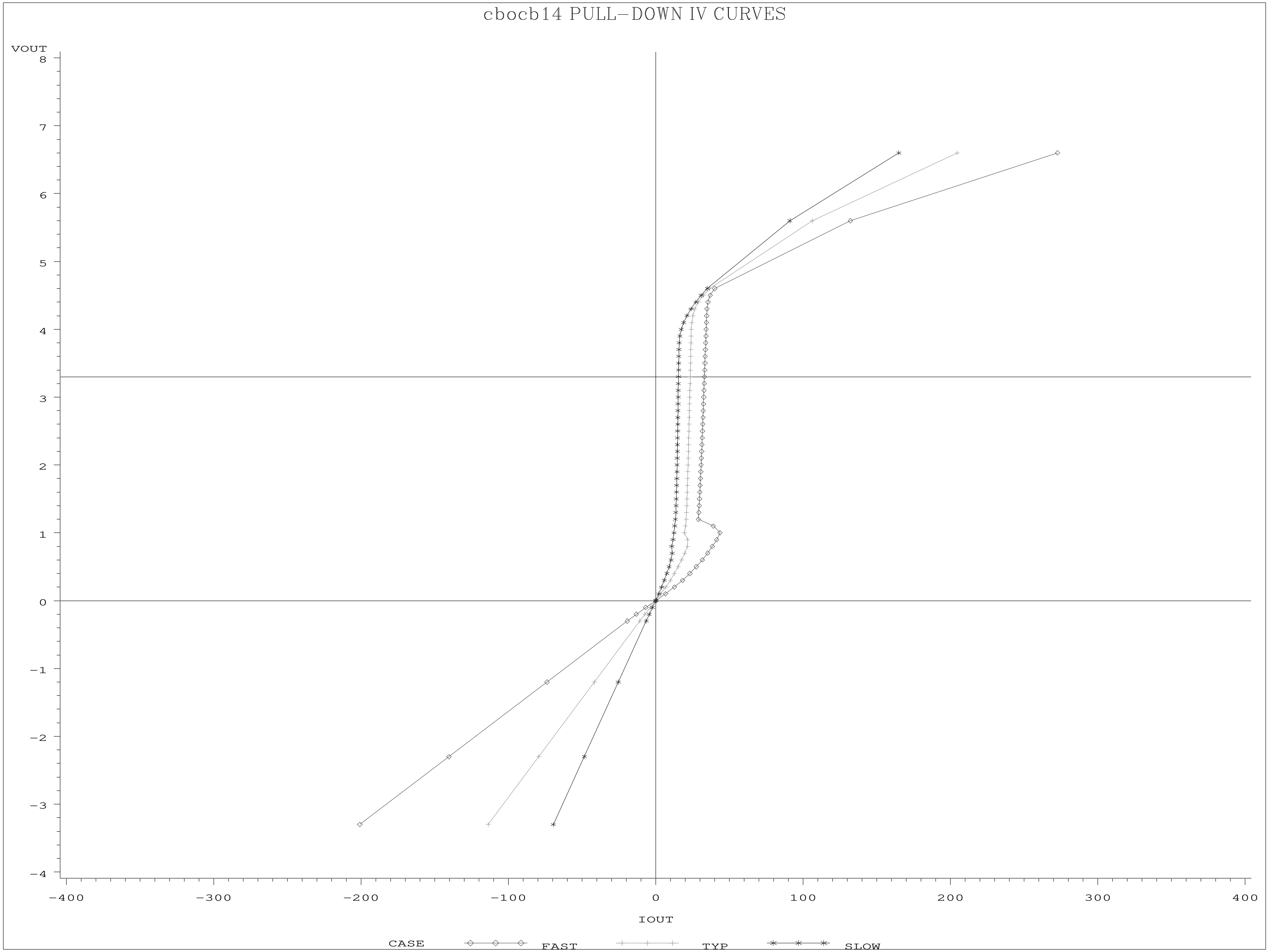

cboca14 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

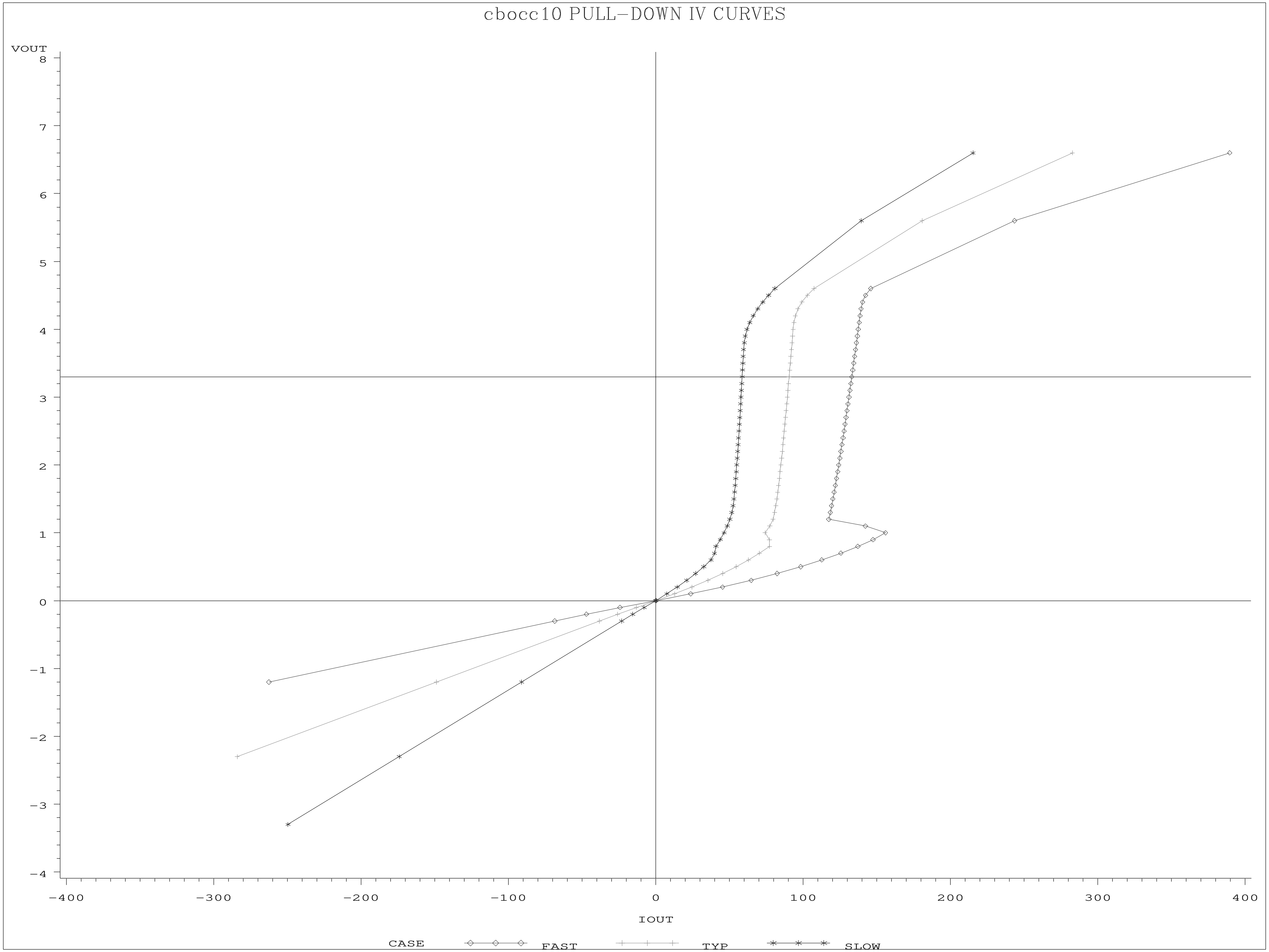

cbocb10 PULL−DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW


cbocb11 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

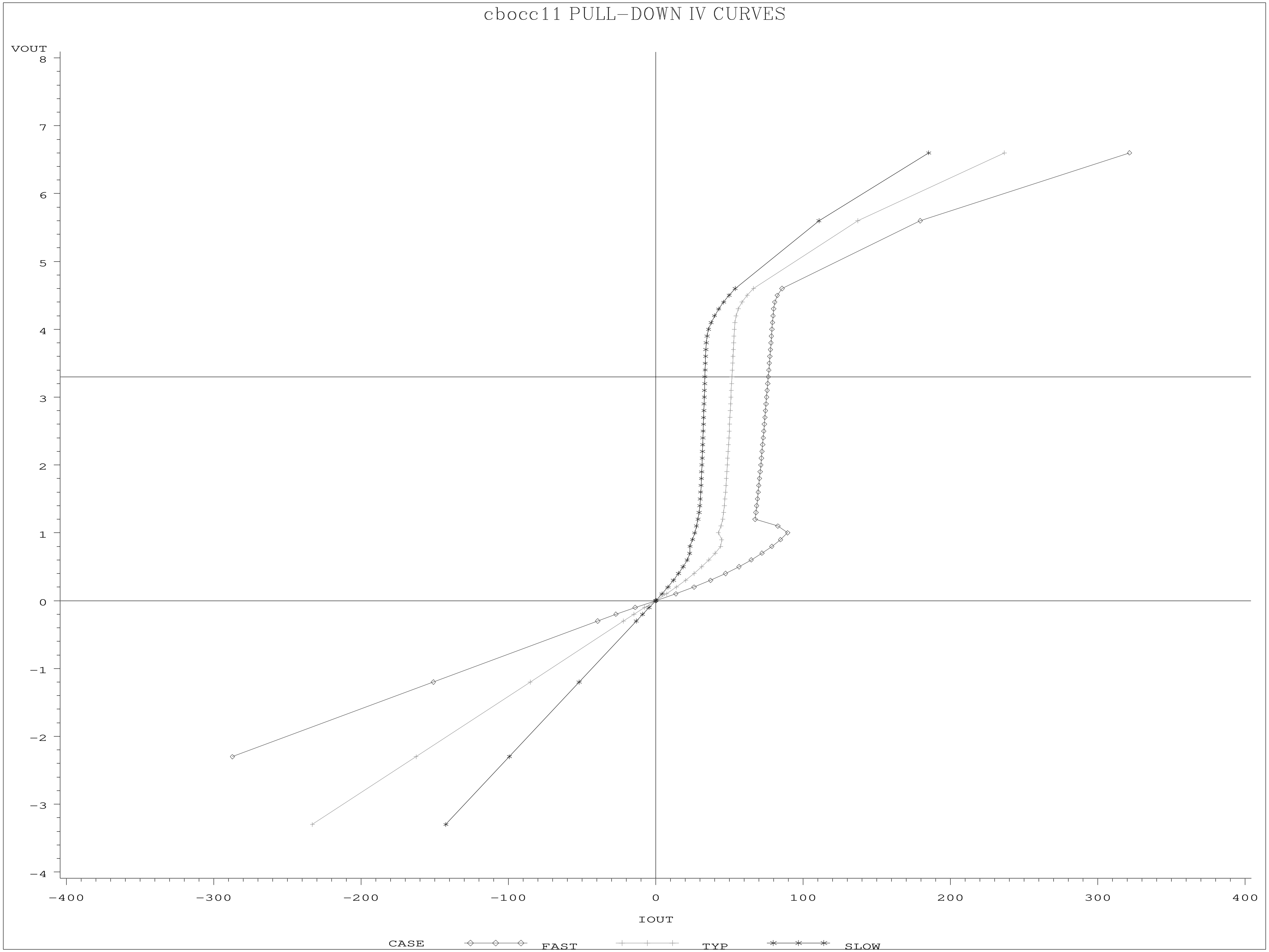

cbocb12 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW


cbocb13 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

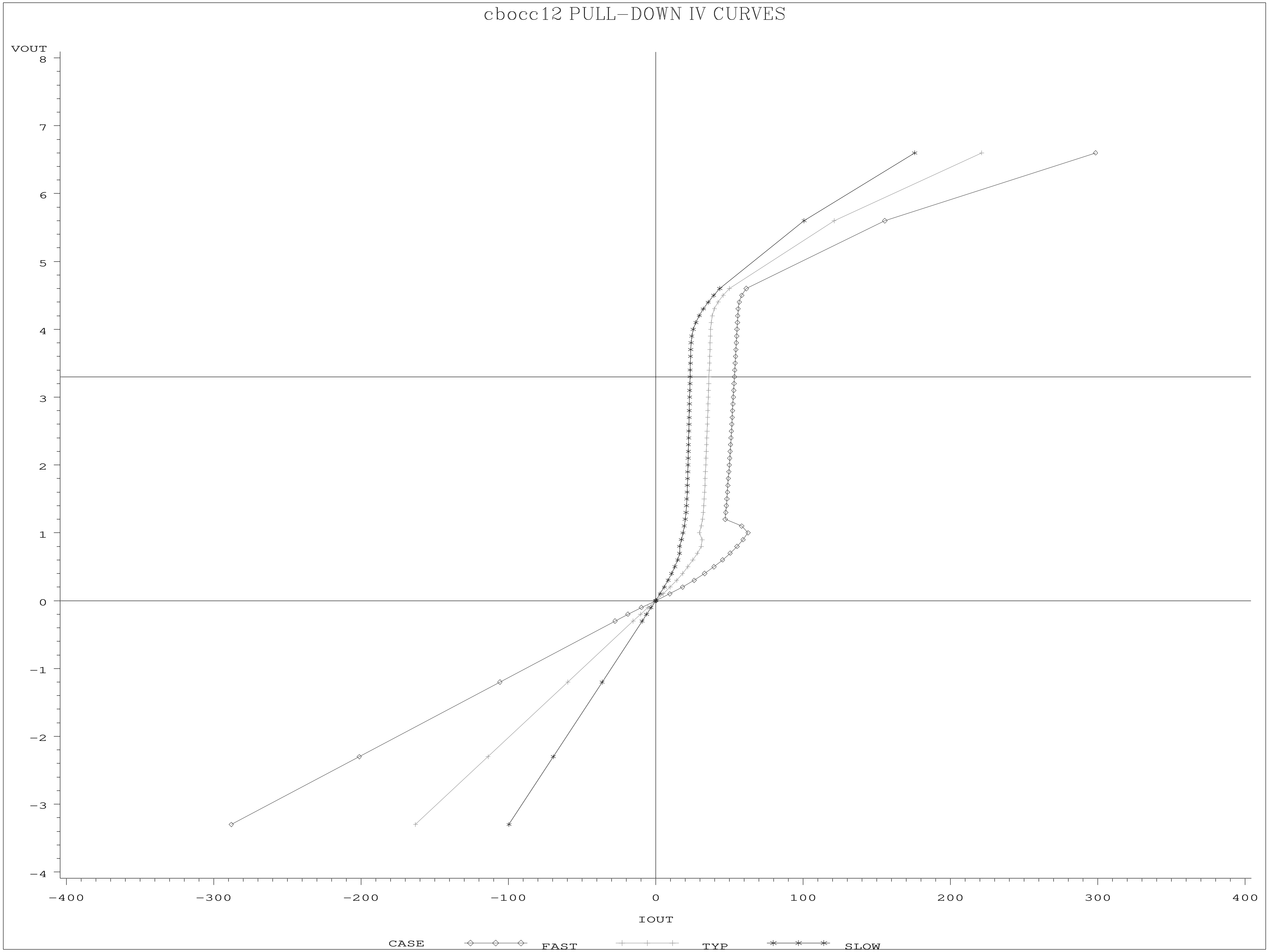

cbocb14 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

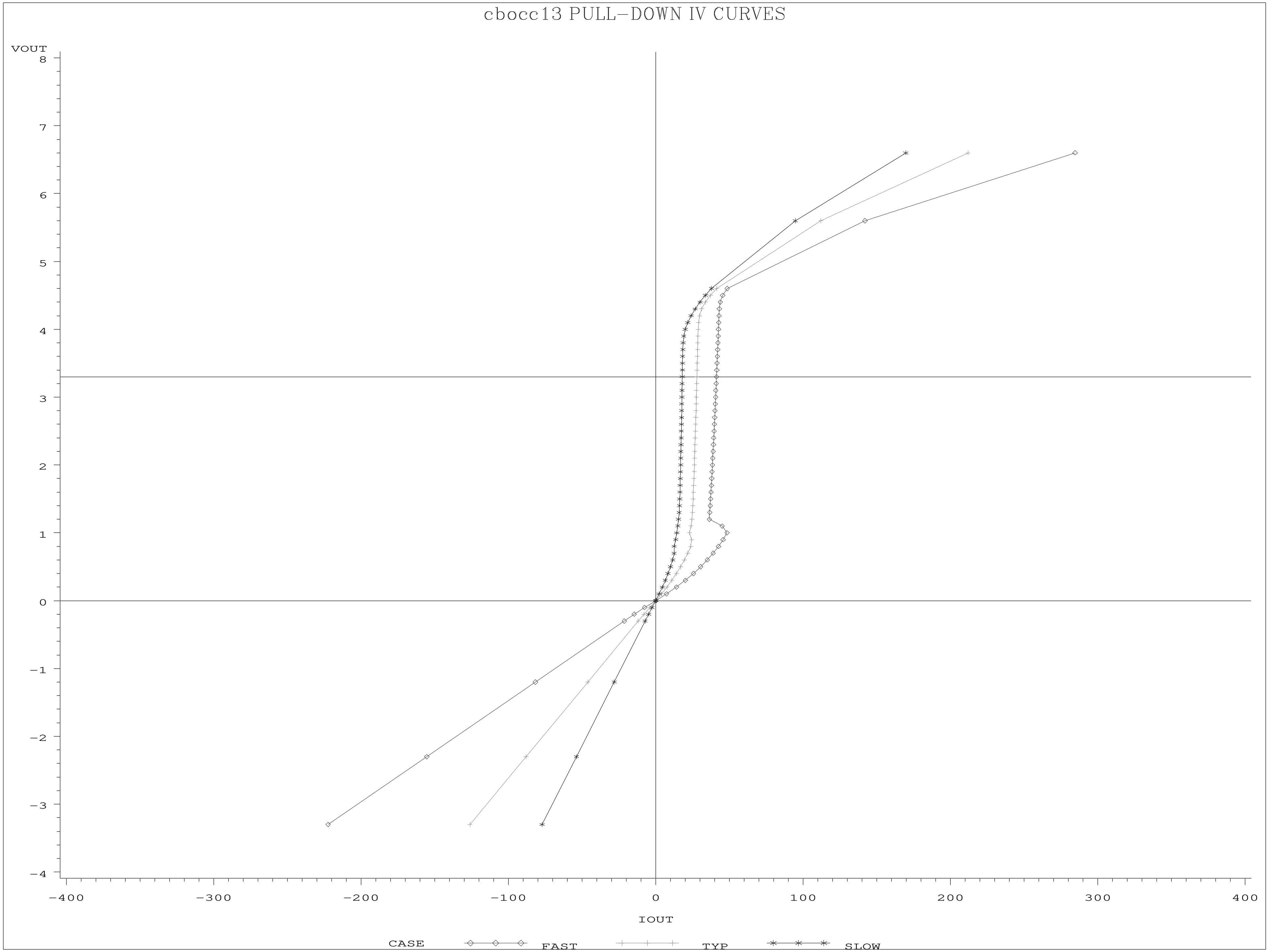


cbocc10 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

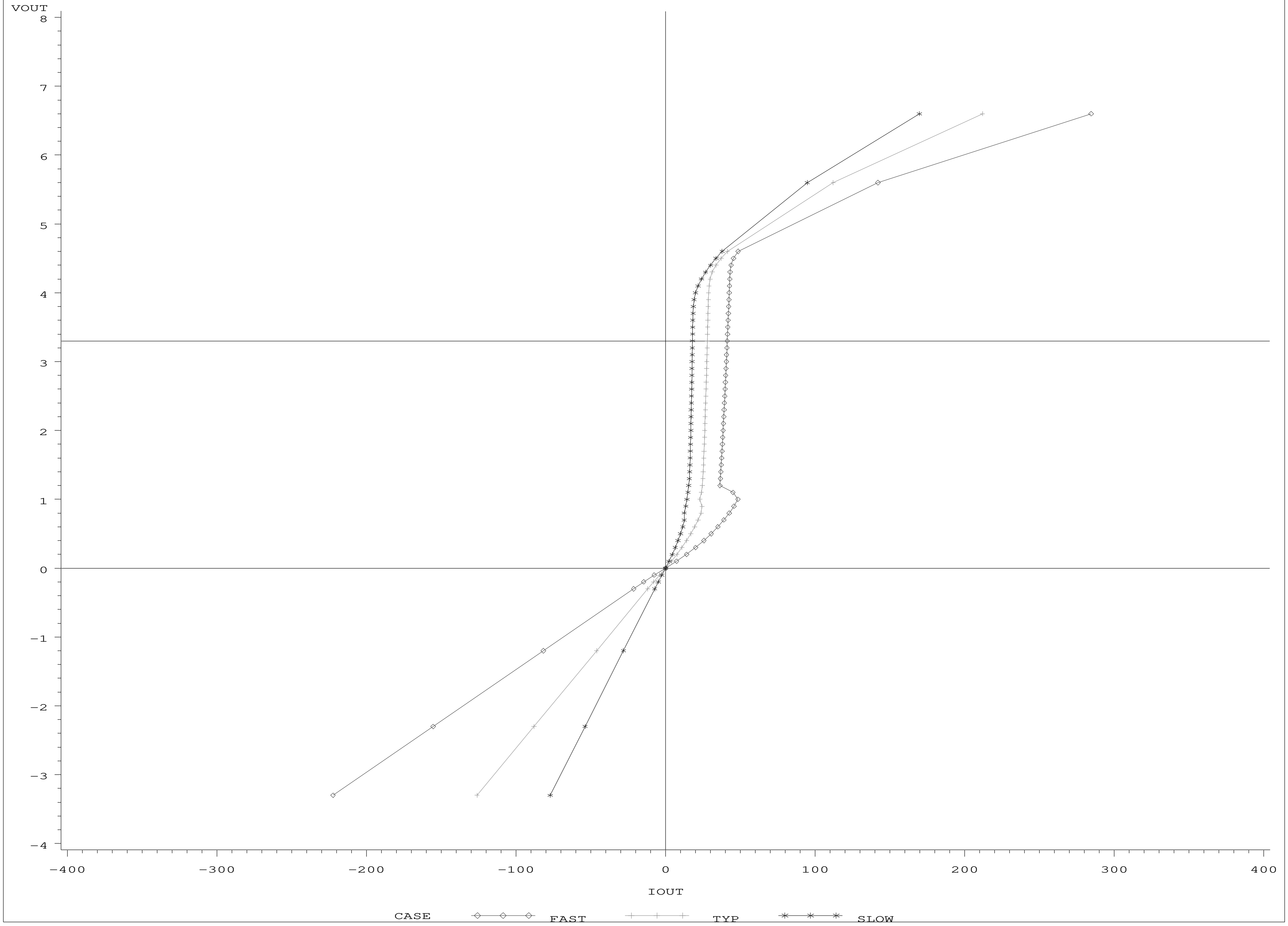


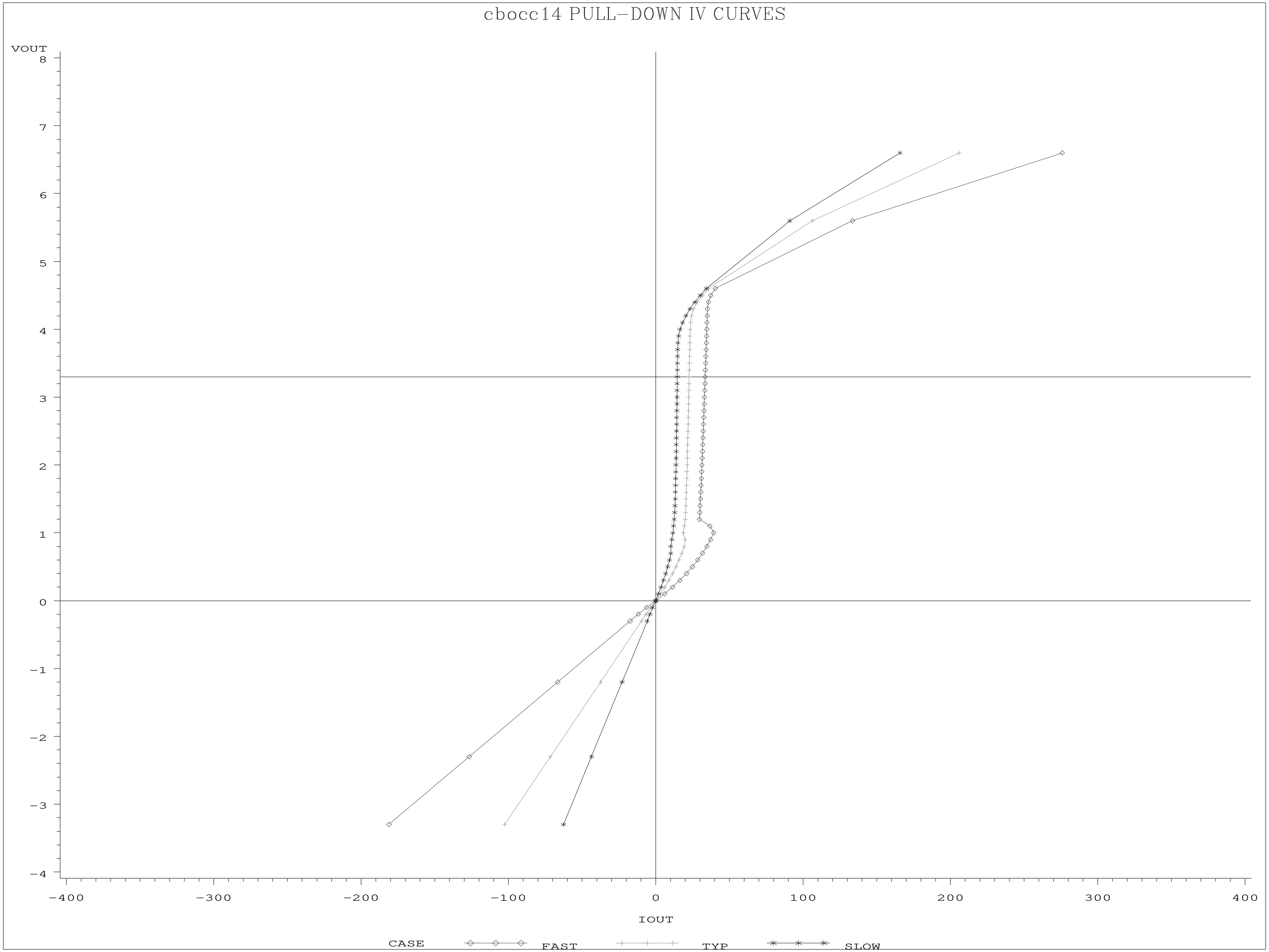
cbocc11 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

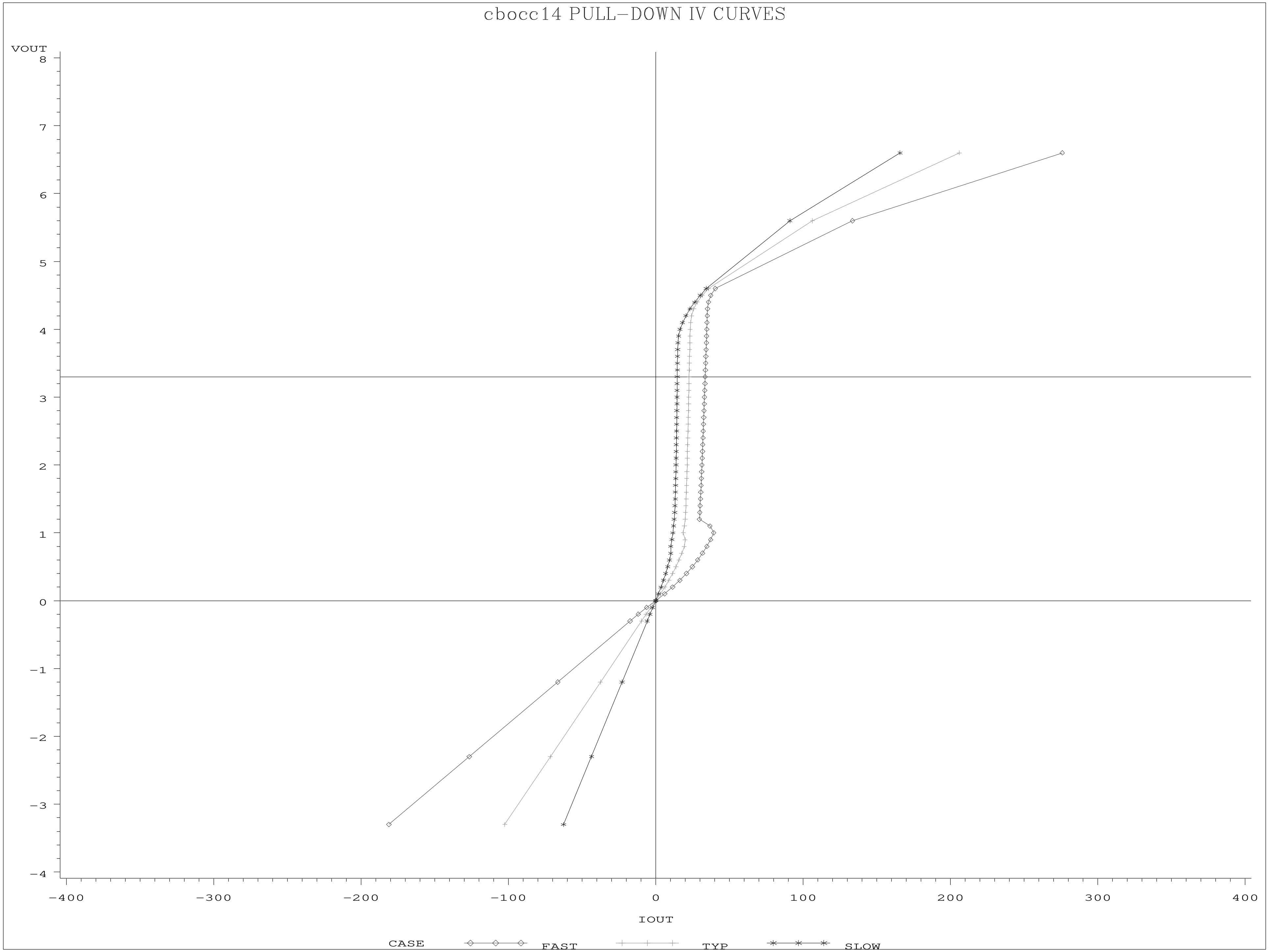

cbocc12 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

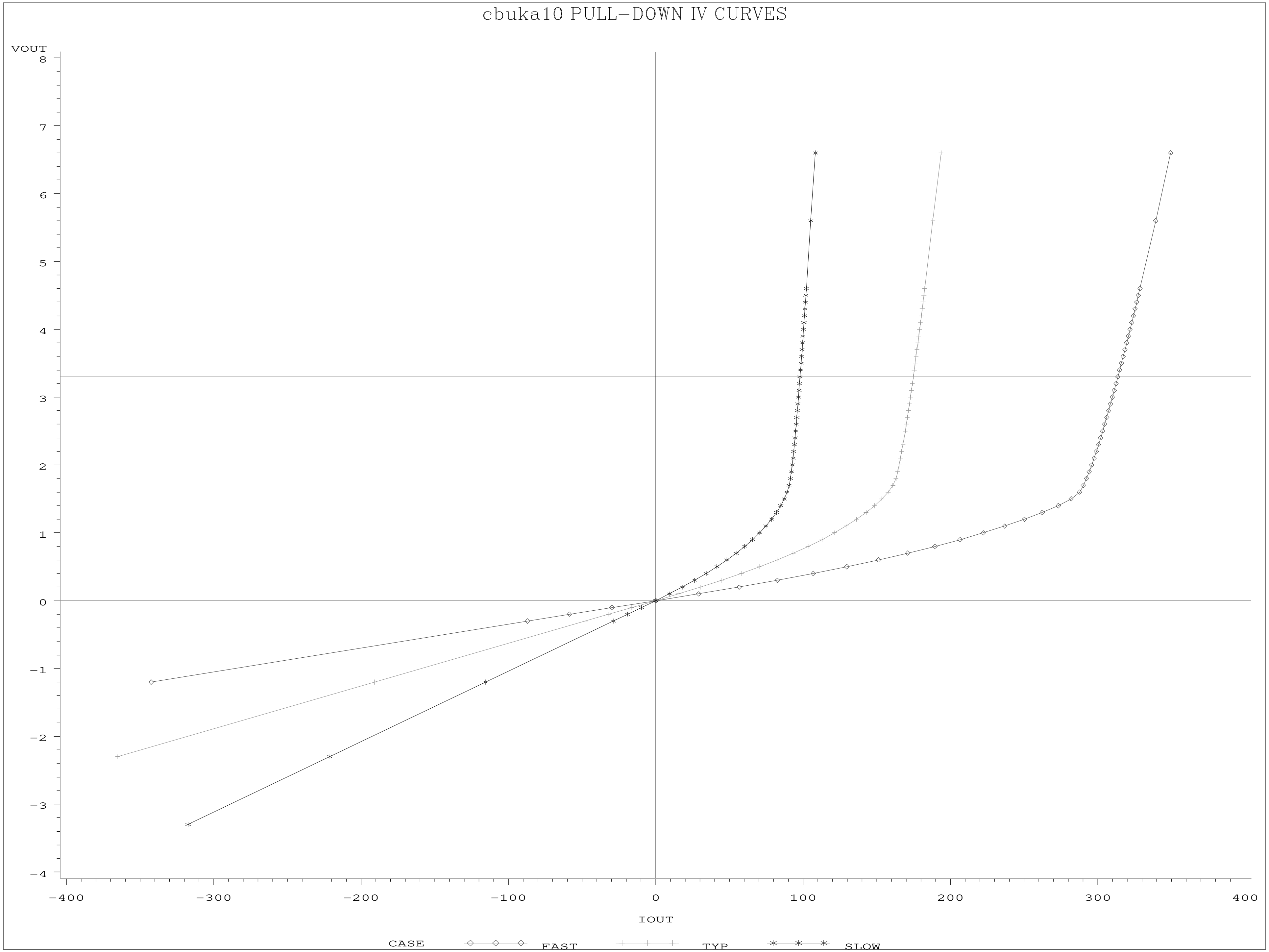

cbocc13 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

cbocc13 PULL-DOWN IV CURVES

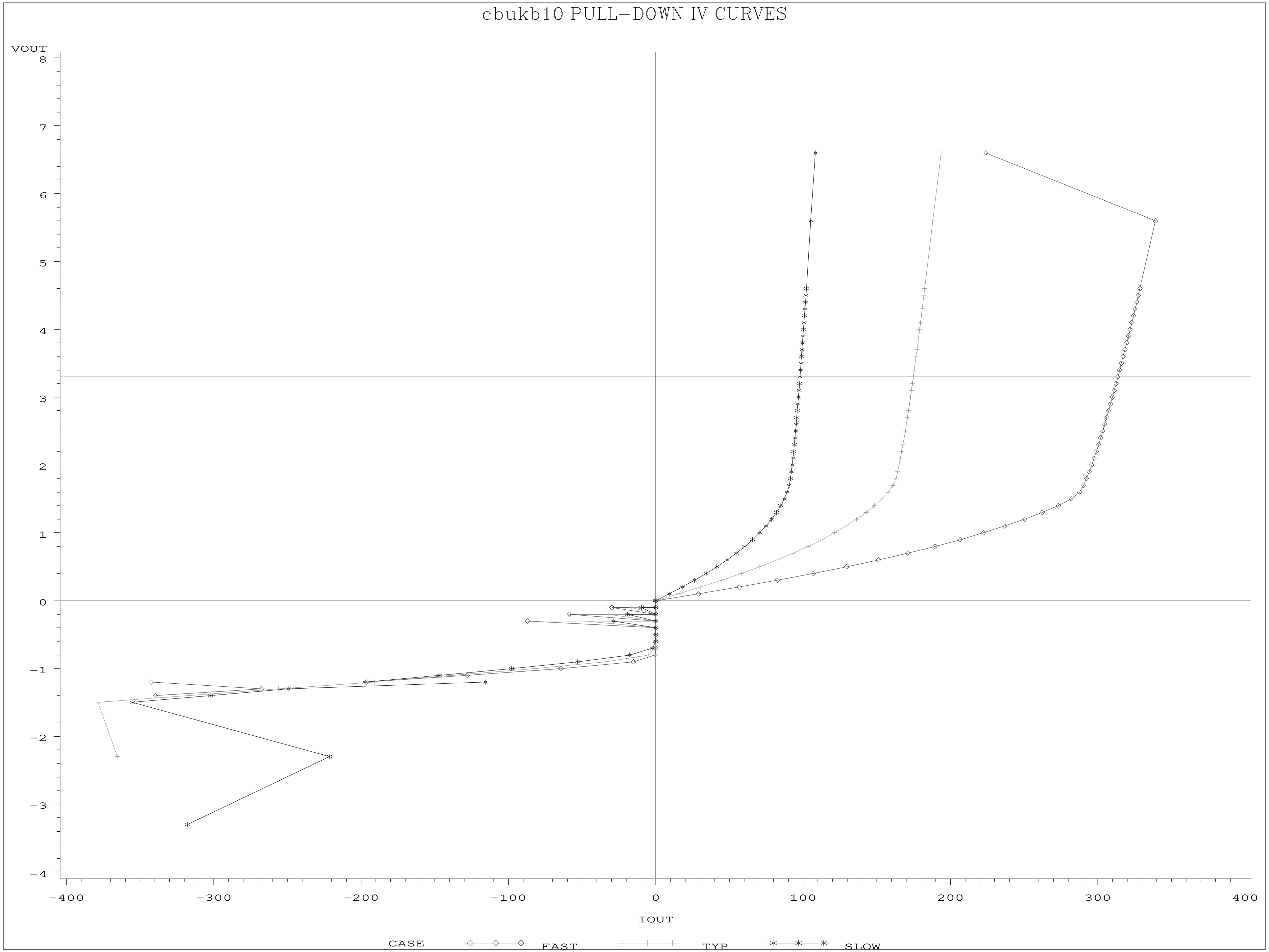



cbocc14 PULL-DOWN IV CURVES
VOUT
IOUT
CASE
FAST
TYP
SLOW

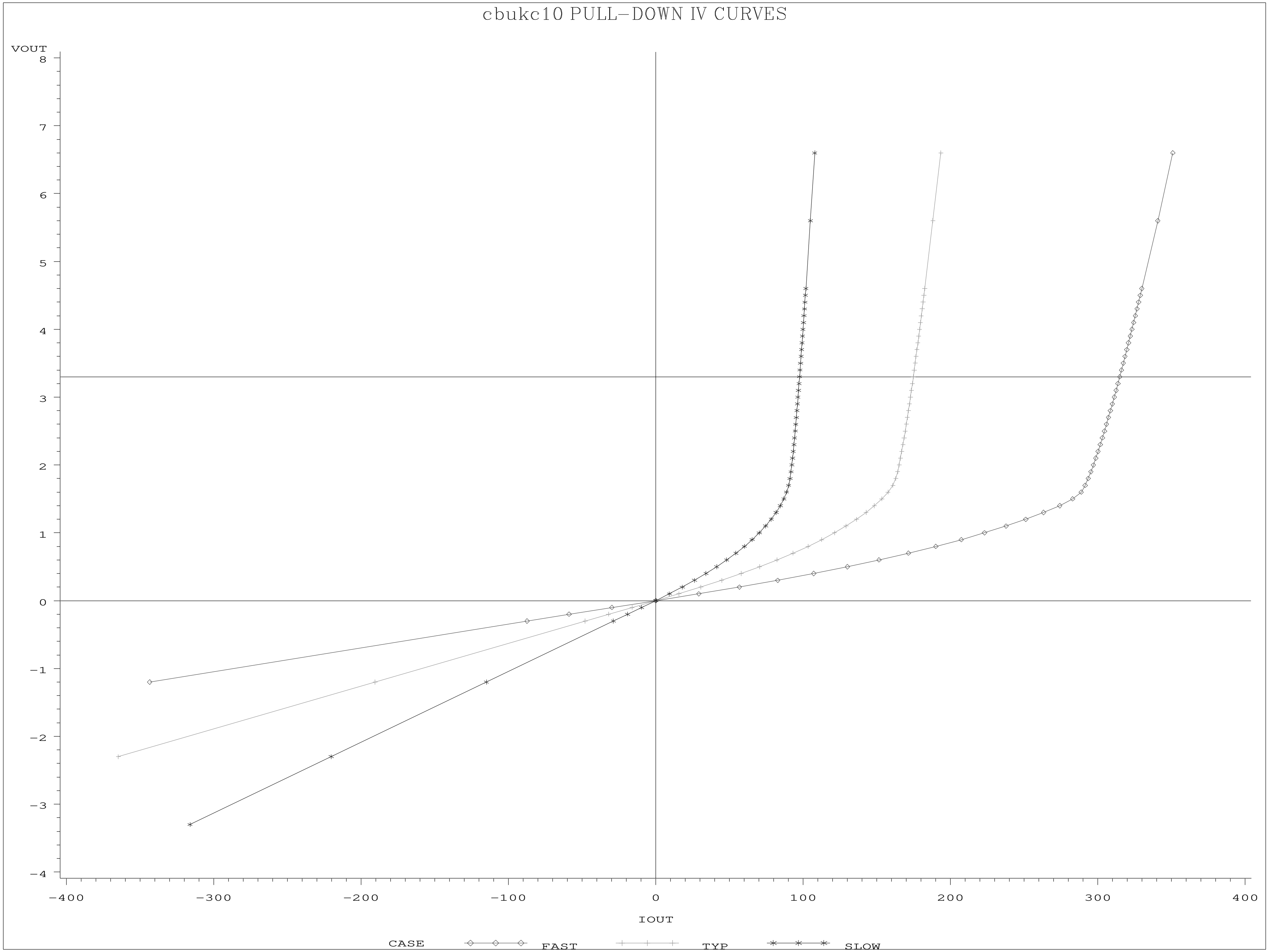

cbocc14 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

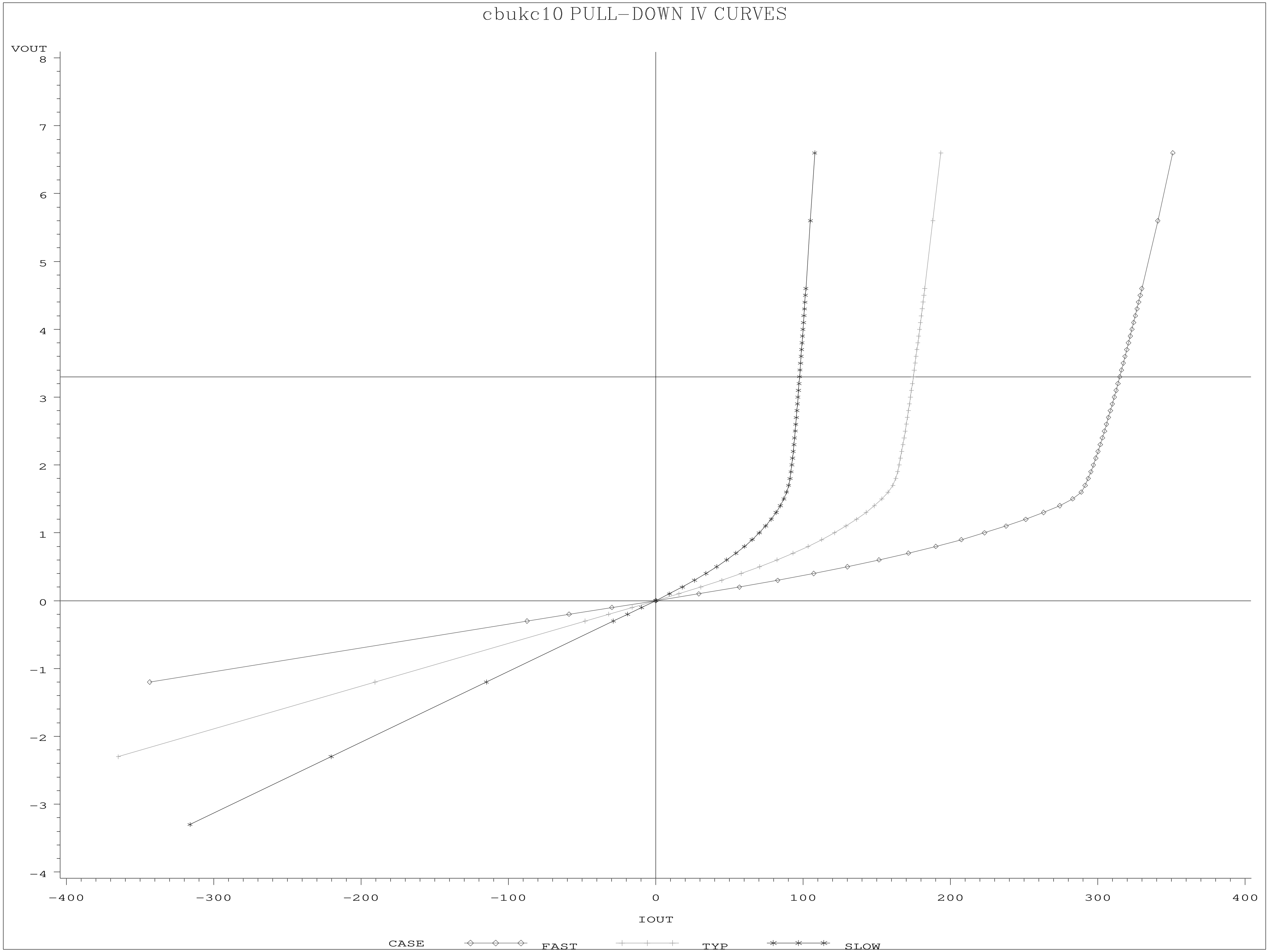


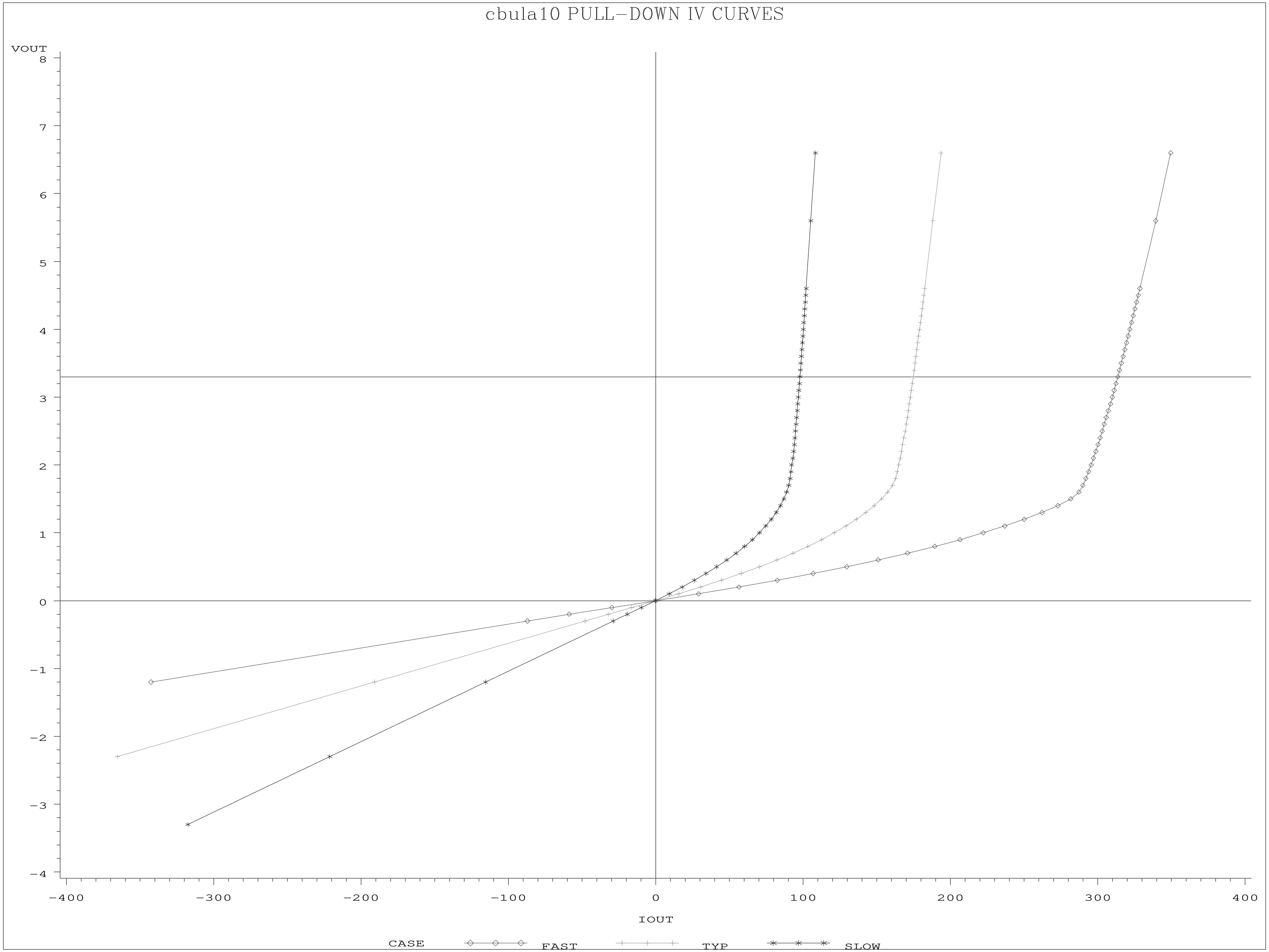
cbuka10 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

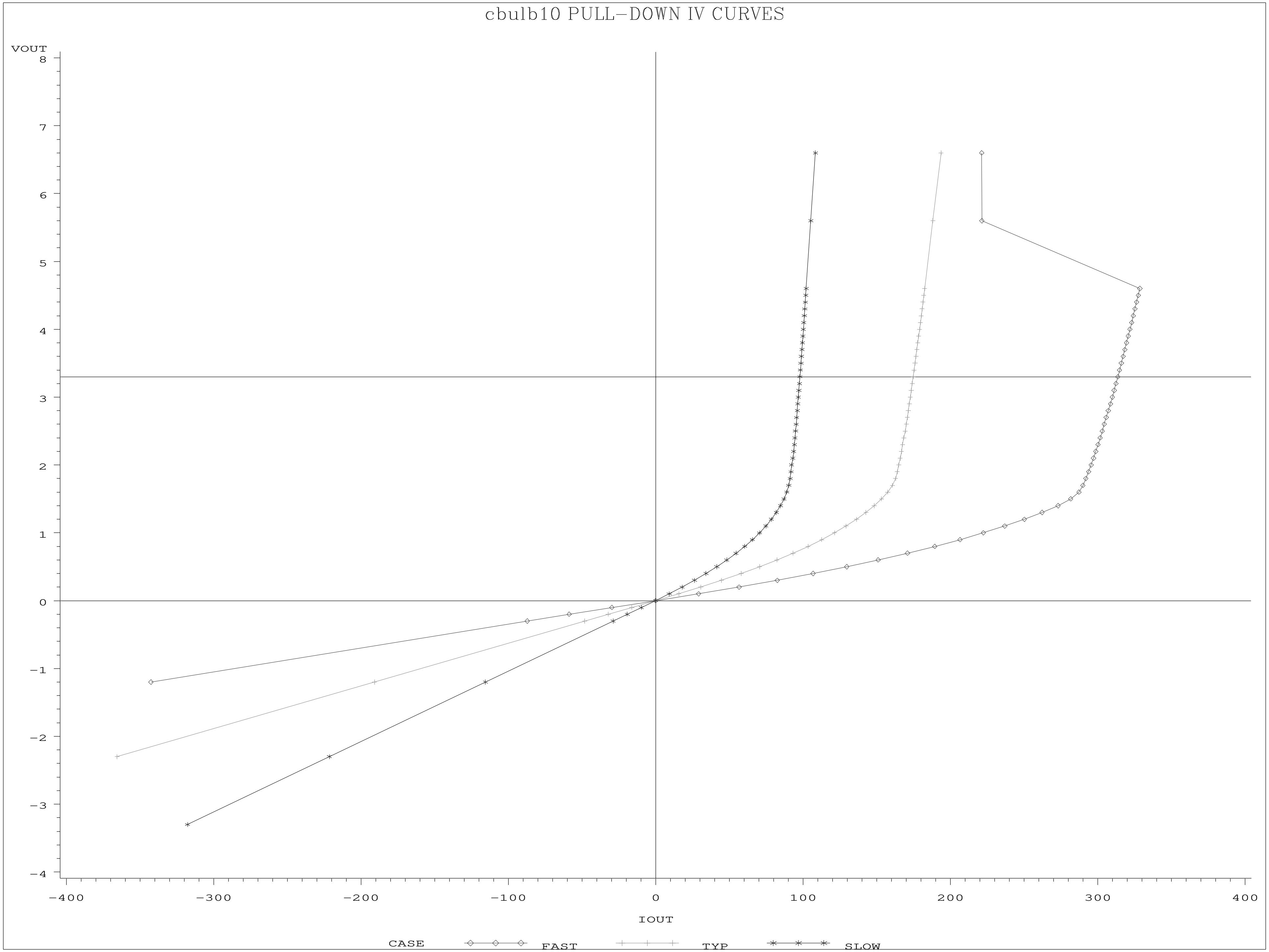


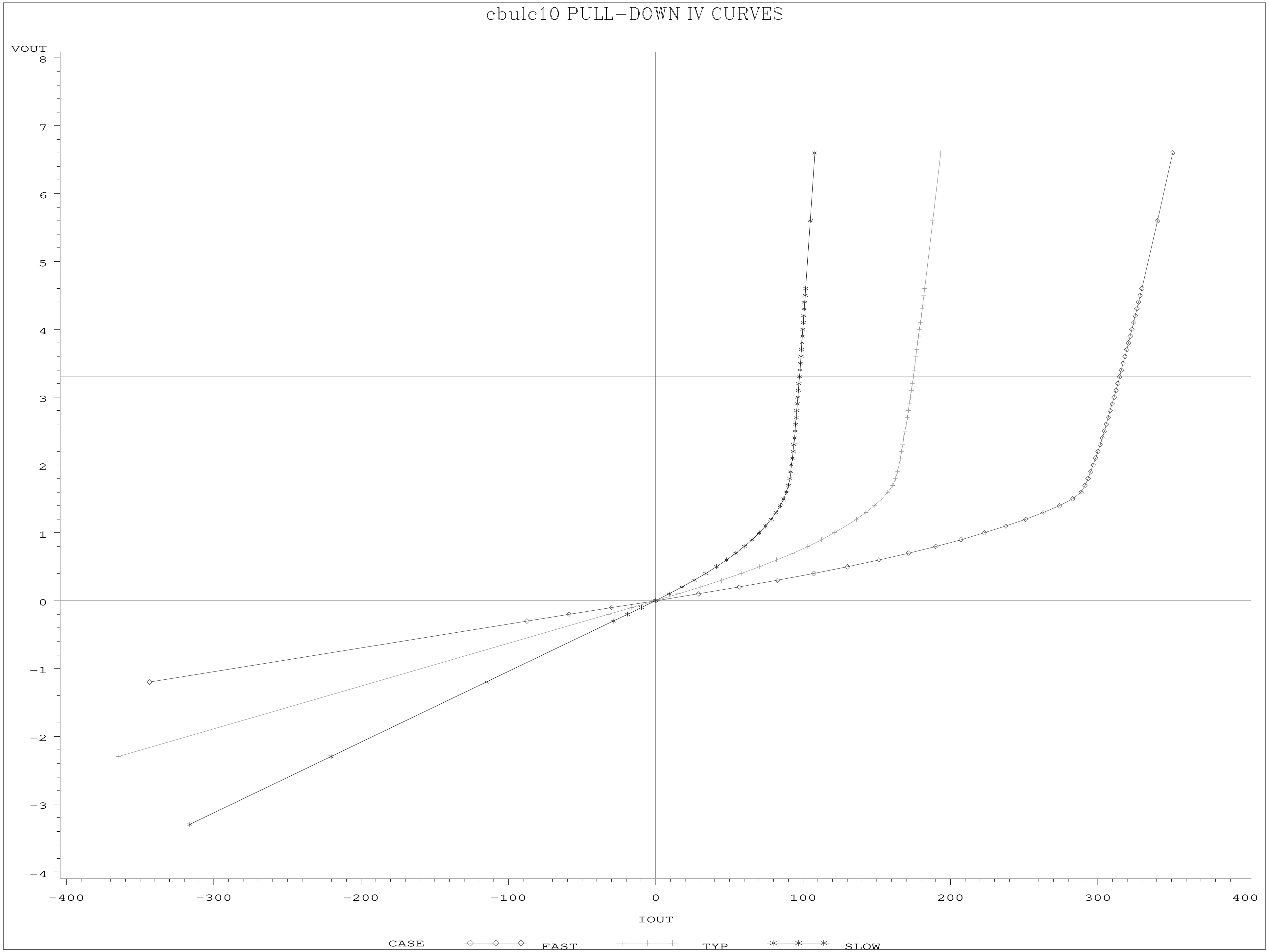
cbukb10 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW


cbukc10 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

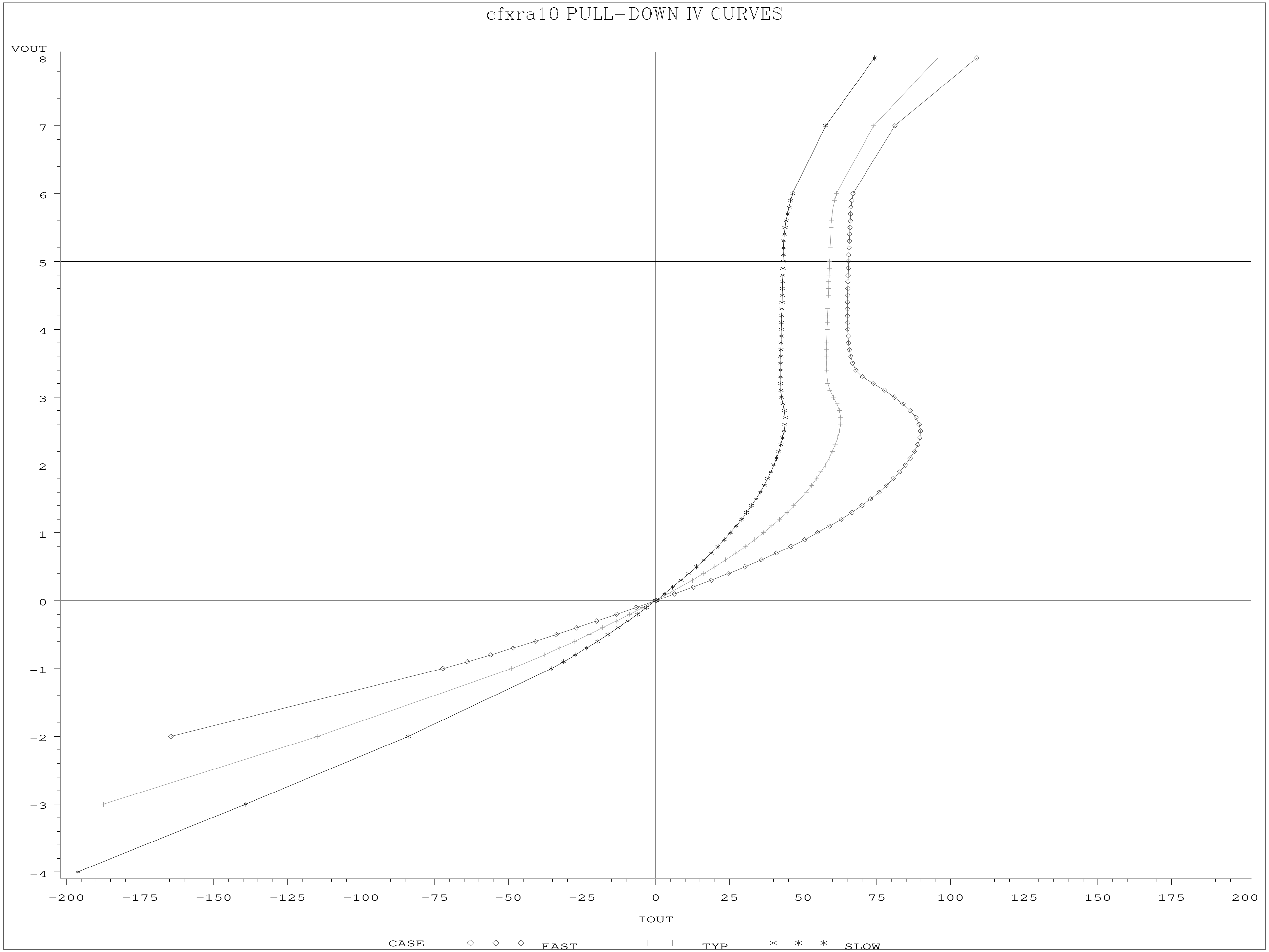

cbukc10 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW


cbula10 PULL-DOWN IV CURVES
VOUT
IOUT
CASE
FAST
TYP
SLOW
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400

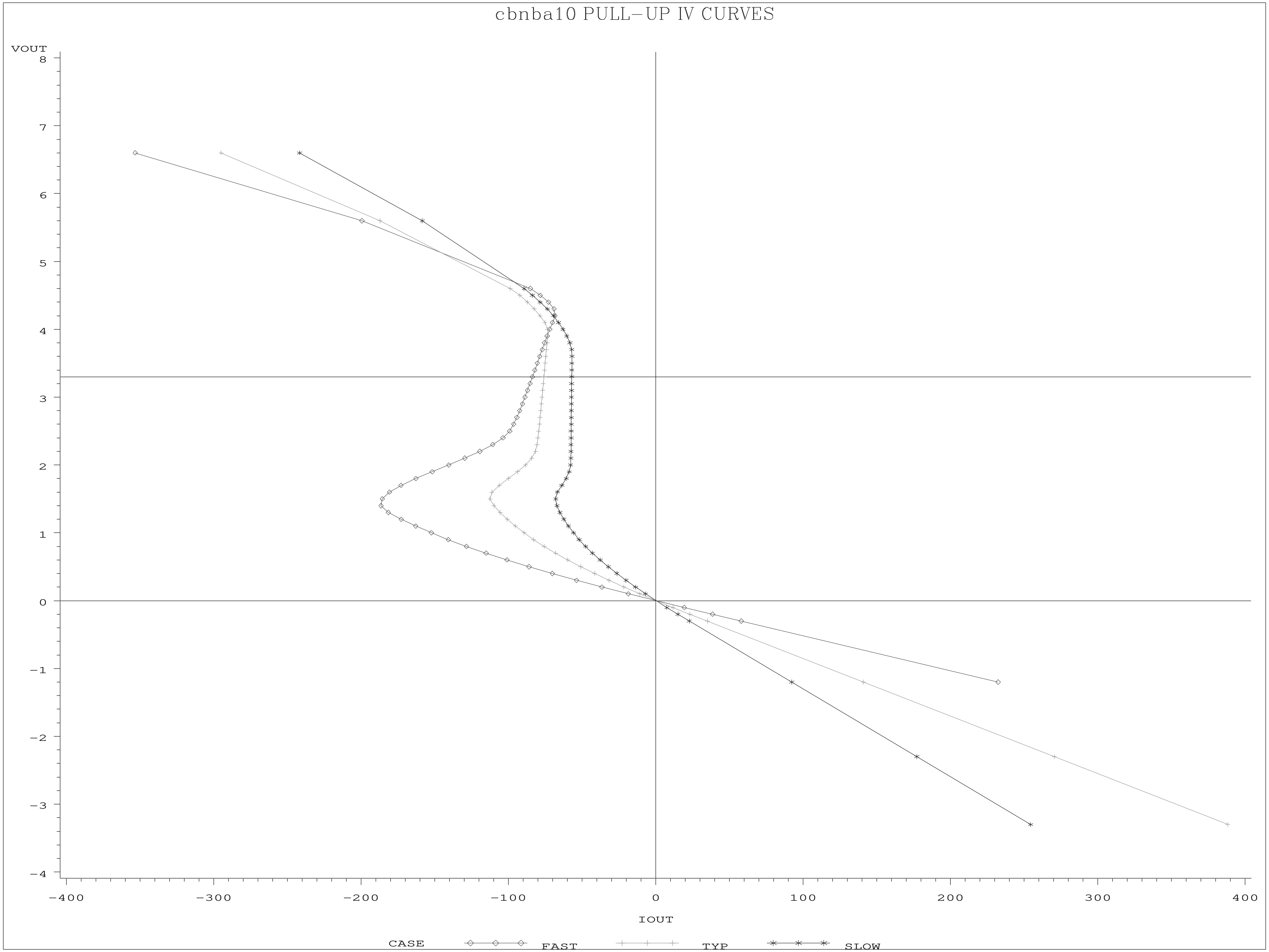

cbulb10 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

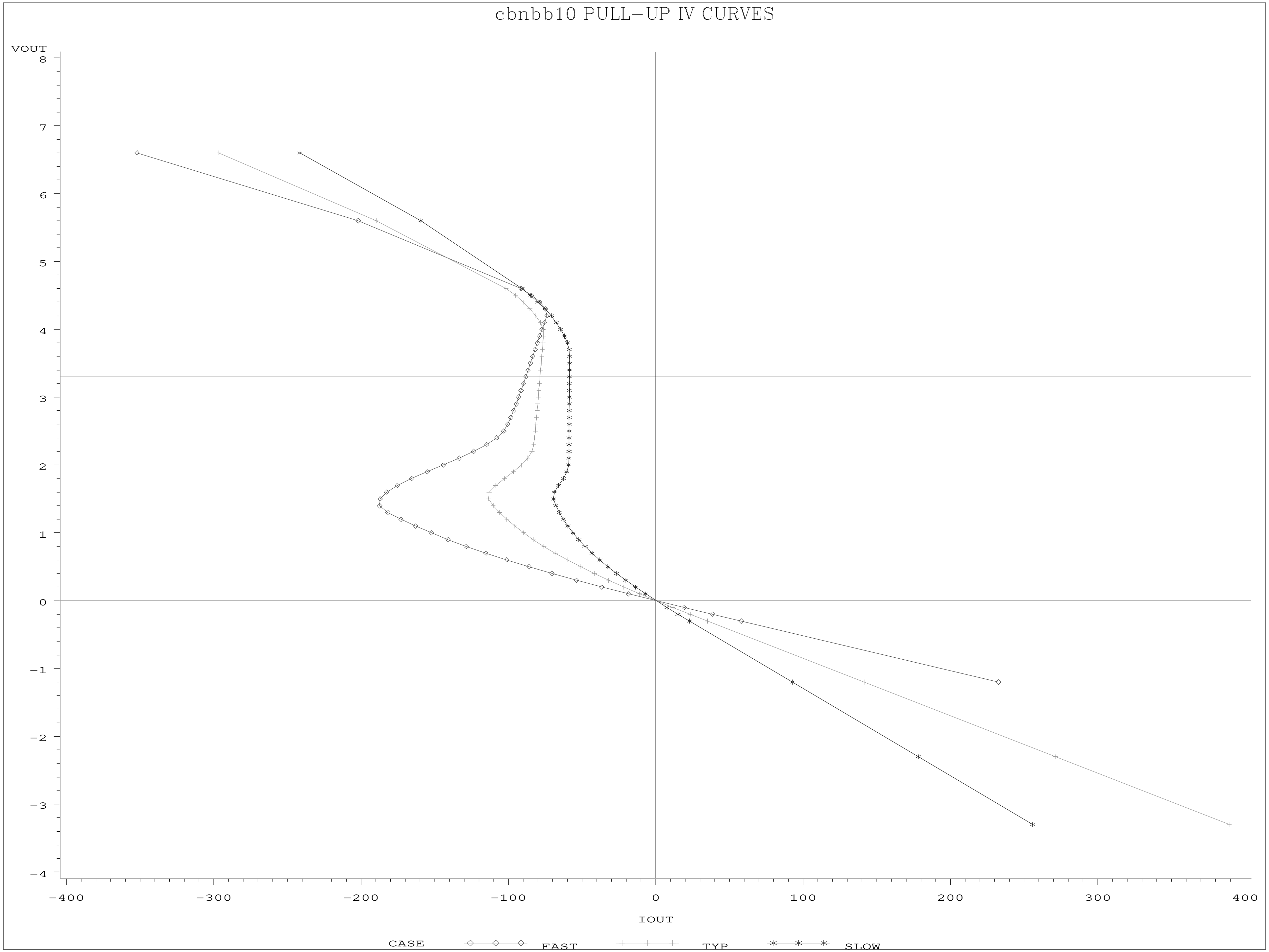


cbulc10 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-400
-300
-200
-100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

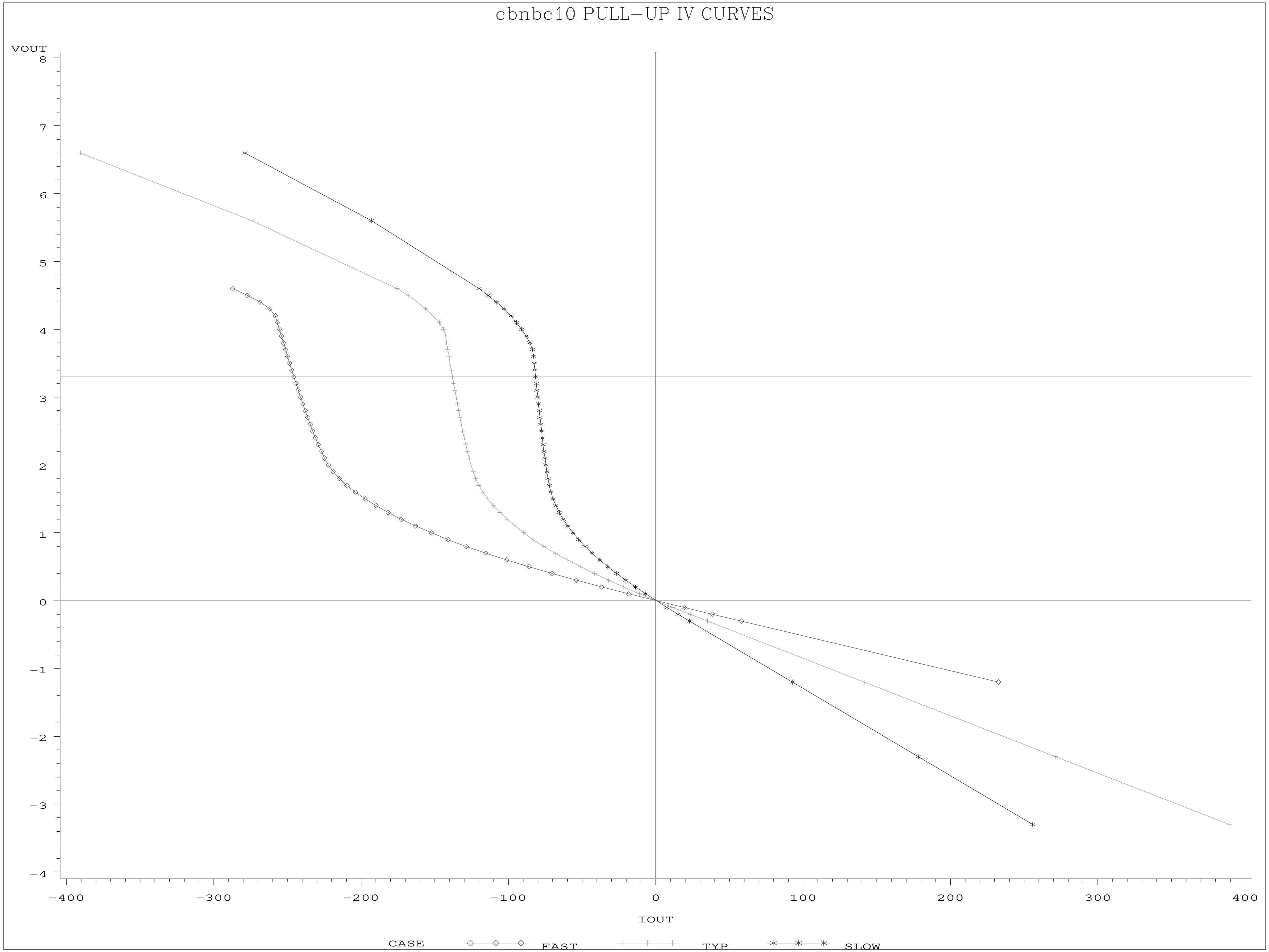


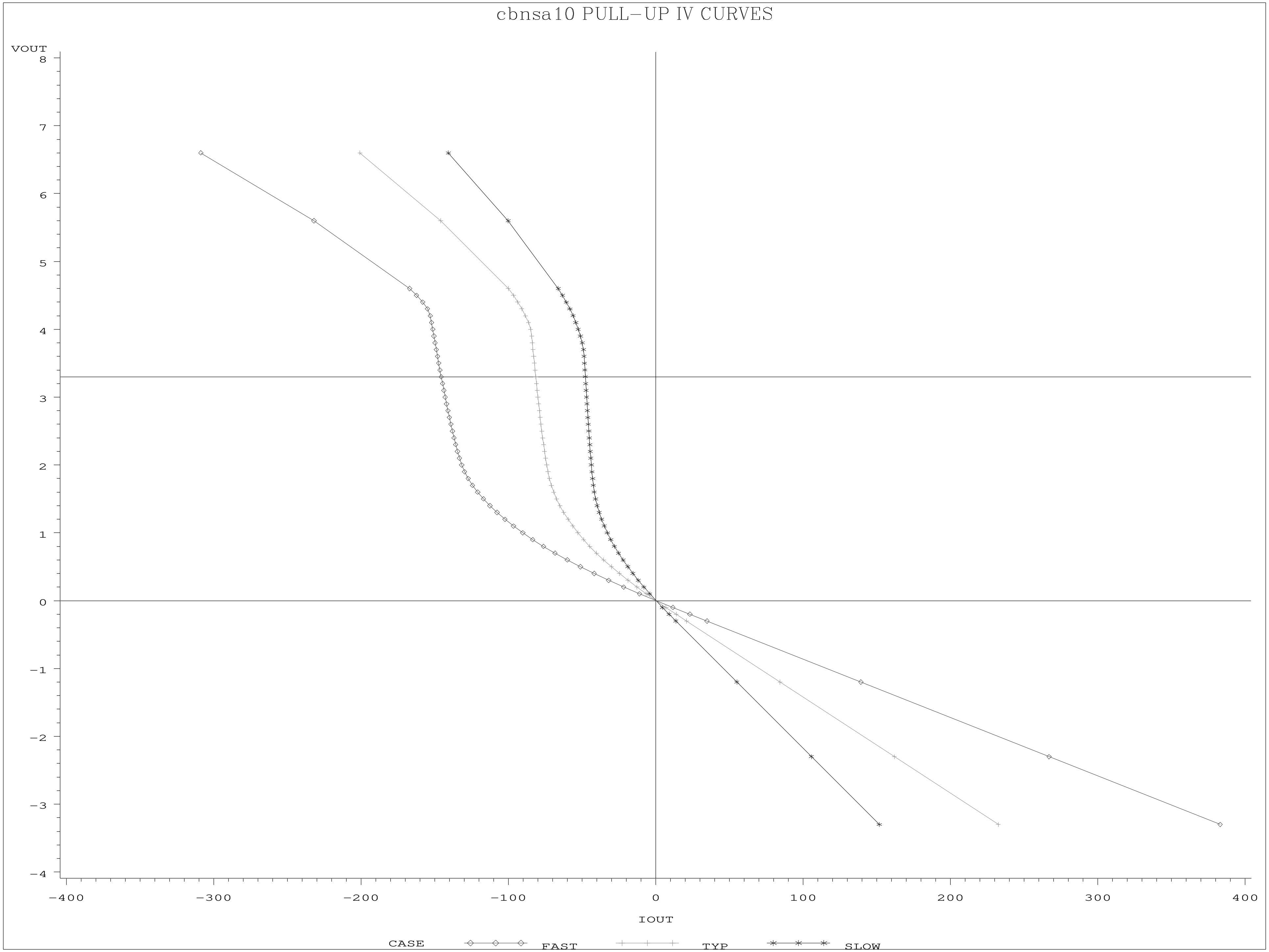
cfxra10 PULL-DOWN IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-200
-175
-150
-125
-100
-75
-50
-25
0
25
50
75
100
125
150
175
200
IOUT
CASE
FAST
TYP
SLOW


cbnba10 PULL-UP IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

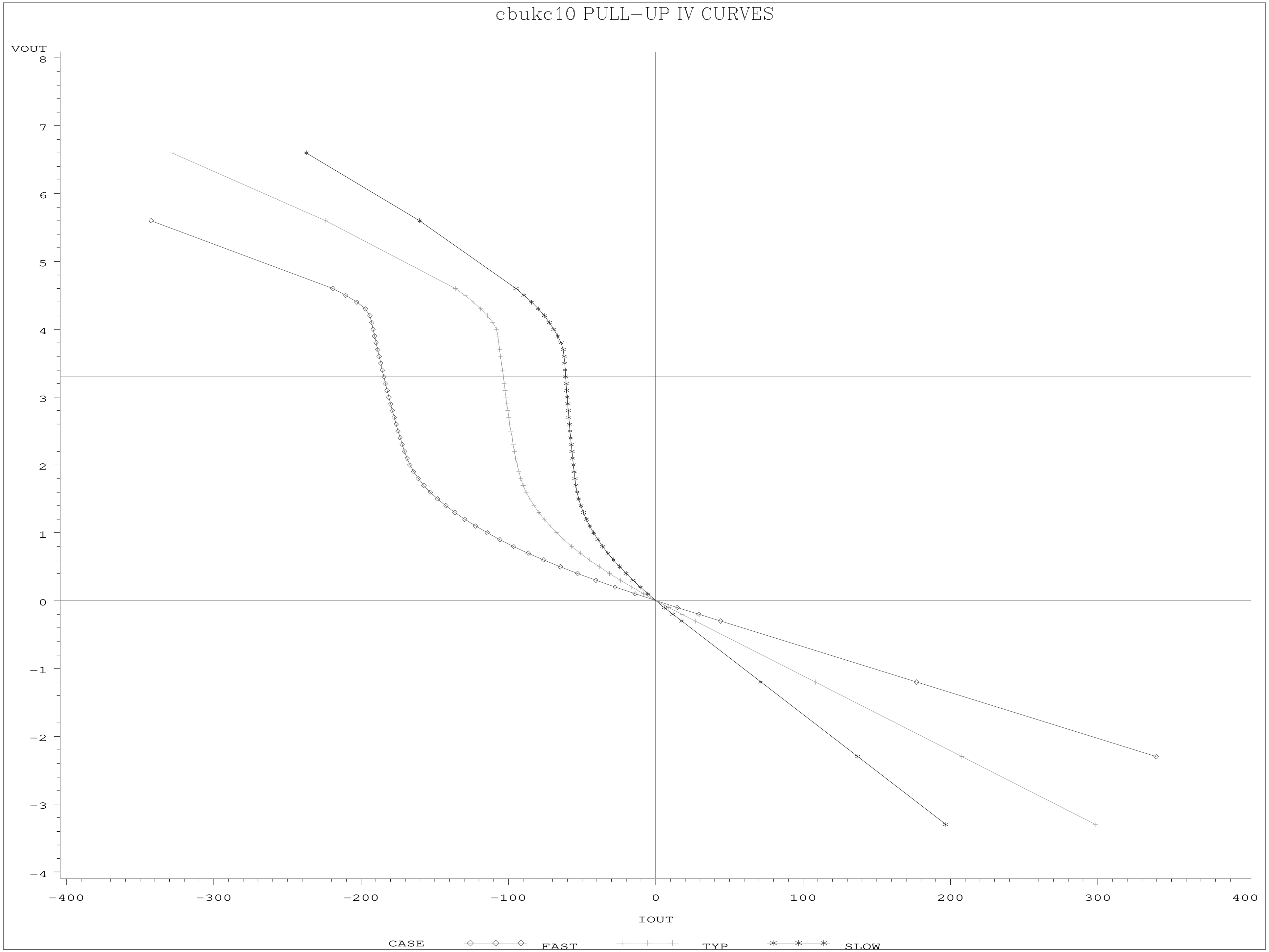


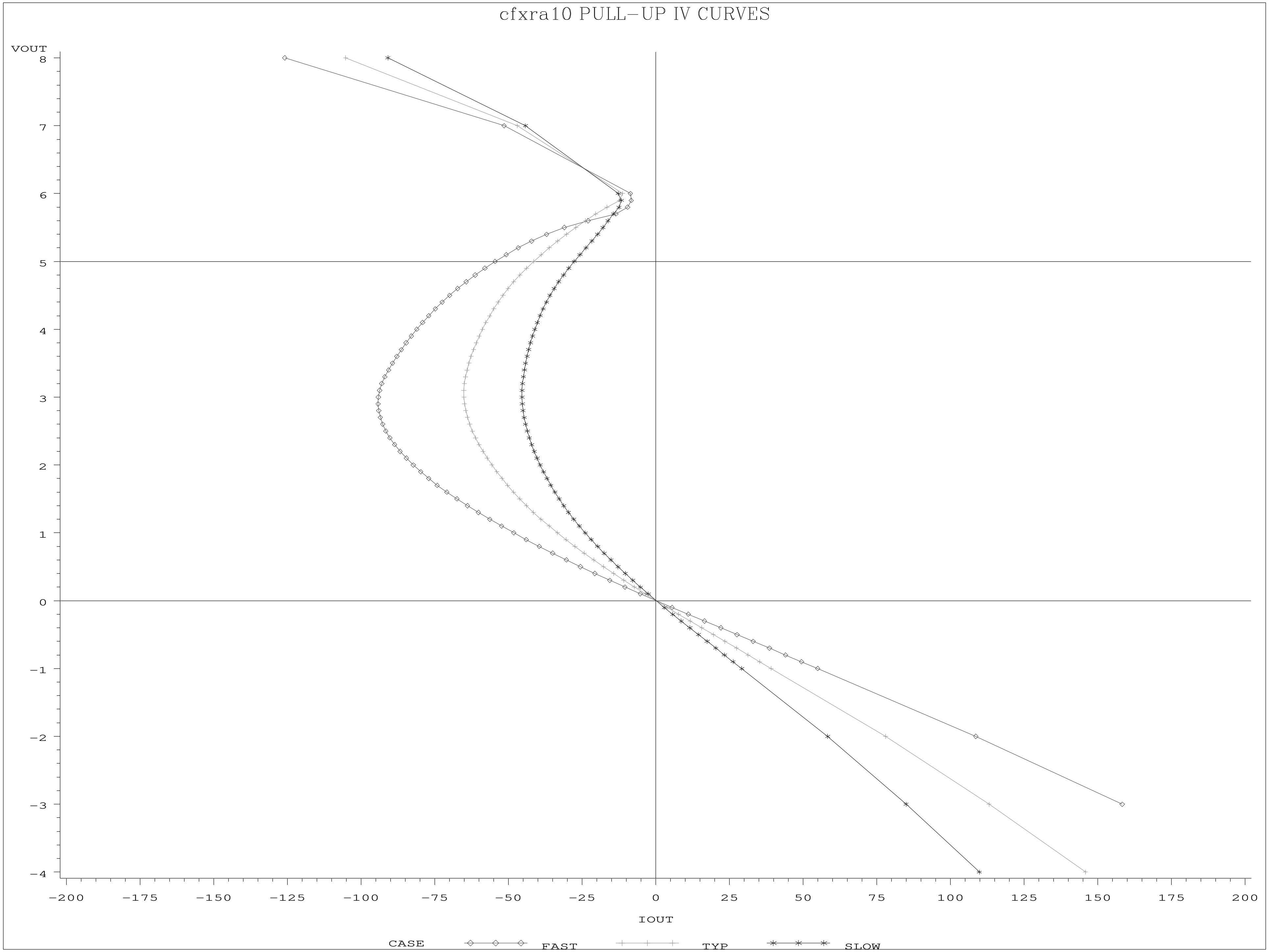
cbnbb10 PULL-UP IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW


cbnbc10 PULL–UP IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW



cbnsa10 PULL−UP IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
−1
−2
−3
−4
−400
−300
−200
−100
0
100
200
300
400
IOUT
CASE
FAST
TYP
SLOW

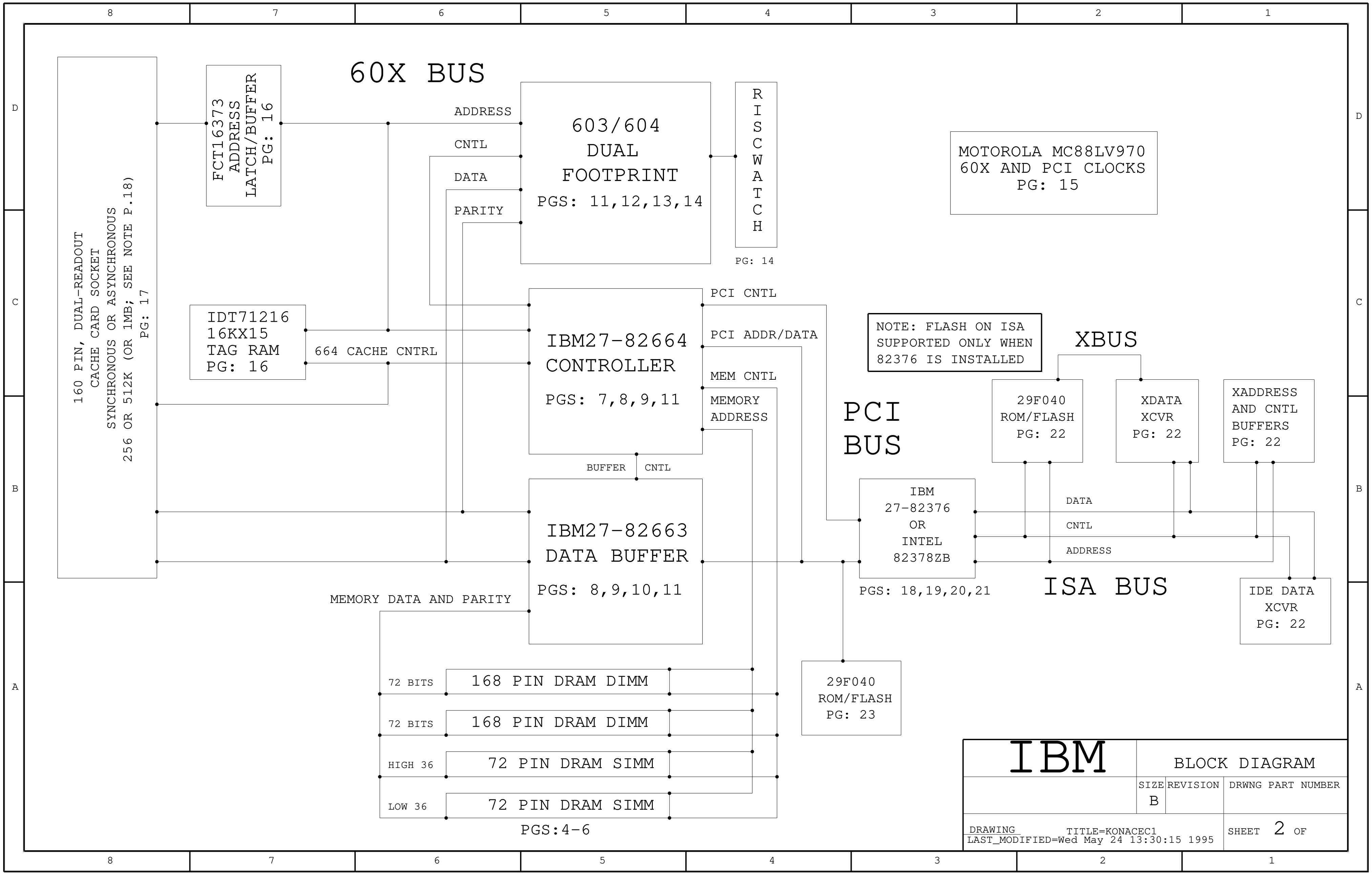

cbukc10 PULL-UP IV CURVES
VOUT
IOUT
CASE
FAST
TYP
SLOW


cfxra10 PULL-UP IV CURVES
VOUT
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-200
-175
-150
-125
-100
-75
-50
-25
0
25
50
75
100
125
150
175
200
IOUT
CASE
FAST
TYP
SLOW

IBM27-82660
EXAMPLE IMPLEMENTATION
FOR MPR660UMU-02

PRELIMINARY

IBM			
	SIZE B	REVISION	DRWNG PART NUMBER
DRAWING TITLE=KONACEC1 LAST_MODIFIED=Mon Jun 5 14:15:24 1995			SHEET 1 OF


60X BUS
FCT16373 ADDRESS LATCH/BUFFER PG: 16
ADDRESS
CNTL
DATA
PARITY
603/604 DUAL FOOTPRINT PGS: 11,12,13,14
RISCWATCH
PG: 14
MOTOROLA MC88LV970 60X AND PCI CLOCKS PG: 15
160 PIN, DUAL-READOUT CACHE CARD SOCKET SYNCHRONOUS OR ASYNCHRONOUS 256 OR 512K (OR 1MB; SEE NOTE P.18) PG: 17
IDT71216 16KX15 TAG RAM PG: 16
664 CACHE CNTRL
IBM27-82664 CONTROLLER PGS: 7,8,9,11
PCI CNTL
PCI ADDR/DATA
MEM CNTL
MEMORY ADDRESS
NOTE: FLASH ON ISA SUPPORTED ONLY WHEN 82376 IS INSTALLED
XBUS
29F040 ROM/FLASH PG: 22
XDATA XCVR PG: 22
XADDRESS AND CNTL BUFFERS PG: 22
PCI BUS
BUFFER
CNTL
IBM27-82663 DATA BUFFER PGS: 8,9,10,11
IBM 27-82376 OR INTEL 82378ZB
PGS: 18,19,20,21
DATA
CNTL
ADDRESS
ISA BUS
IDE DATA XCVR PG: 22
MEMORY DATA AND PARITY
72 BITS
168 PIN DRAM DIMM
72 BITS
168 PIN DRAM DIMM
HIGH 36
72 PIN DRAM SIMM
LOW 36
72 PIN DRAM SIMM
PGS:4-6
29F040 ROM/FLASH PG: 23
IBM
BLOCK DIAGRAM
SIZE B
REVISION
DRWNG PART NUMBER
DRAWING
TITLE=KONACEC1
LAST_MODIFIED=Wed May 24 13:30:15 1995
SHEET 2 OF

CONFIGURATION NOTES

SYSTEM CLOCK CONFIGURATION OPTIONS

MOTO LV970 CONFIG SETTINGS:

J9: LEGAL SETTINGS FOR REFERENCE DESIGN MARKED WITH (*)
MPC601_CLKS = 0; VCO_SEL SET TO 0

LEGAL	PCI DIV1	PCI DIV0	BCLK DIV1	BCLK DIV0	2X_PCLK,PCLK_EN BCLK_EN	ALL OTHER BCLK(X)	PCI_CLK(X)
*	0	0	0	0	FOUT	FOUT	FOUT
	0	0	0	1	FOUT	FOUT/2	FOUT/2
	0	0	1	0	FOUT	FOUT/3	FOUT/3
	0	0	1	1	FOUT	FOUT/4	FOUT/4
*	0	1	0	0	FOUT	FOUT	FOUT/2
	0	1	0	1	FOUT	FOUT/2	FOUT/4
	0	1	1	0	FOUT	FOUT/3	FOUT/6
	0	1	1	1	FOUT	FOUT/4	FOUT/8
*	1	0	0	0	FOUT	FOUT	FOUT/3
	1	0	0	1	FOUT	FOUT/2	FOUT/6
	1	0	1	0	FOUT	FOUT/3	FOUT/9
	1	0	1	1	FOUT	FOUT/4	FOUT/12
*	1	1	0	0	FOUT	FOUT	FOUT
	1	1	0	1	FOUT	FOUT/2	FOUT
	1	1	1	0	FOUT	FOUT/3	FOUT
	1	1	1	1	FOUT	FOUT/4	FOUT

VCO_SEL PIN (PIN 52):

FOUT = 8 X FIN IF VCO_SEL=1
FOUT = 4 X FIN IF VCO_SEL=0
FVCO = 32 X FIN; 300MHZ <= FVCO <= 1000MHZ
IT IS REQUIRED THAT 10MHZ <= FIN <= 25 MHZ

FIN IS DEFINED BY Y1 CRYSTAL. ALTERNATELY, THE PLL CAN BE BYPASSED AND A REFERENCE FREQUENCY (TTL_CLK) IS TRANSFERRED TO THE OUTPUTS.

603/604 PLL CONFIG

J10: USABLE PLLCFG SETTINGS

PLL CFG0	PLL CFG1	PLL CFG2	PLL CFG3	604 PCLK:BCLK RATIO	604 VCO DIV	603 PCLK:BCLK RATIO	603 VCO DIV
0	0	0	0	1:1	/2	1:1	/2
0	0	0	1	1:1	/4	1:1	/4
0	0	1	0	1:1	/8	1:1	/8
0	0	1	1				
0	1	0	0	2:1	/2	2:1	/2
0	1	0	1	2:1	/4	2:1	/4
0	1	1	0				
0	1	1	1				
1	0	0	0	3:1	/2	3:1	/2
1	0	0	1				
1	0	1	0				
1	0	1	1				
1	1	0	0	3:2	/2	4:1	/2
1	1	0	1	3:2	/4		
1	1	1	0				
1	1	1	1				

SET PLLCFG SUCH THAT FVCO / DIV = BCLK X RATIO
AND 100<=FVCO<=200 FOR 603, 112.5<=FVCO,=225 FOR 604

PCI TO ISA BRIDGE CONFIGURATION OPTIONS

THIS REFERENCE IMPLEMENTATION ILLUSTRATES THE USAGE OF ONE OF TWO PCI/ISA BRIDGE CHIPS, THE IBM27-82376 OR THE INTEL SIO 82378ZB.

THESE TWO CHIPS SHARE 171 PINS WITH IDENTICAL FUNCTION AND LOCATION. THERE ARE AN ADDITIONAL 19 PINS THAT HAVE SLIGHTLY DIFFERENT FUNCTION OR DEFINITION AND 18 PINS THAT HAVE COMPLETELY UNIQUE FUNCTIONS.

THE READER SHOULD REFERENCE PAGES 18-21 FOR NOTES REGARDING THE CONNECTION OF THESE DEVICES.

60X PROCESSOR OPERATING MODES

THE 604 PROCESSOR HAS TWO DRIVER MODE CONFIGURATION PINS THAT CAN BE CONFIGURED TO ALLOW THE USER TO CHOOSE THE DRIVE STRENGTH FOR A SUBSET OF THE 60X BUS CONTROL PINS. THESE DRIVER MODE PINS (DRVMOD<1:0>) ARE SET BY PULL UP/DOWN RESISTORS ON KONA.

THE DRVMOD PINS SHOULD BE SET ACCORDING TO THE TABLE SHOWN BELOW.

DRVMOD<0> = 0, R110 = 1K, R137 = NOPOP
= 1, R110 = NOPOP, R137 = 10K
DRVMOD<1> = 0, R109 = 1K, R128 = NOPOP
= 1, R109 = NOPOP, R128 = 10K

MODE	DRVMOD<0>	DRVMOD<1>
DISABLED	0	0
NORMAL	0	1
STRONG	1	0
HERCULEAN	1	1

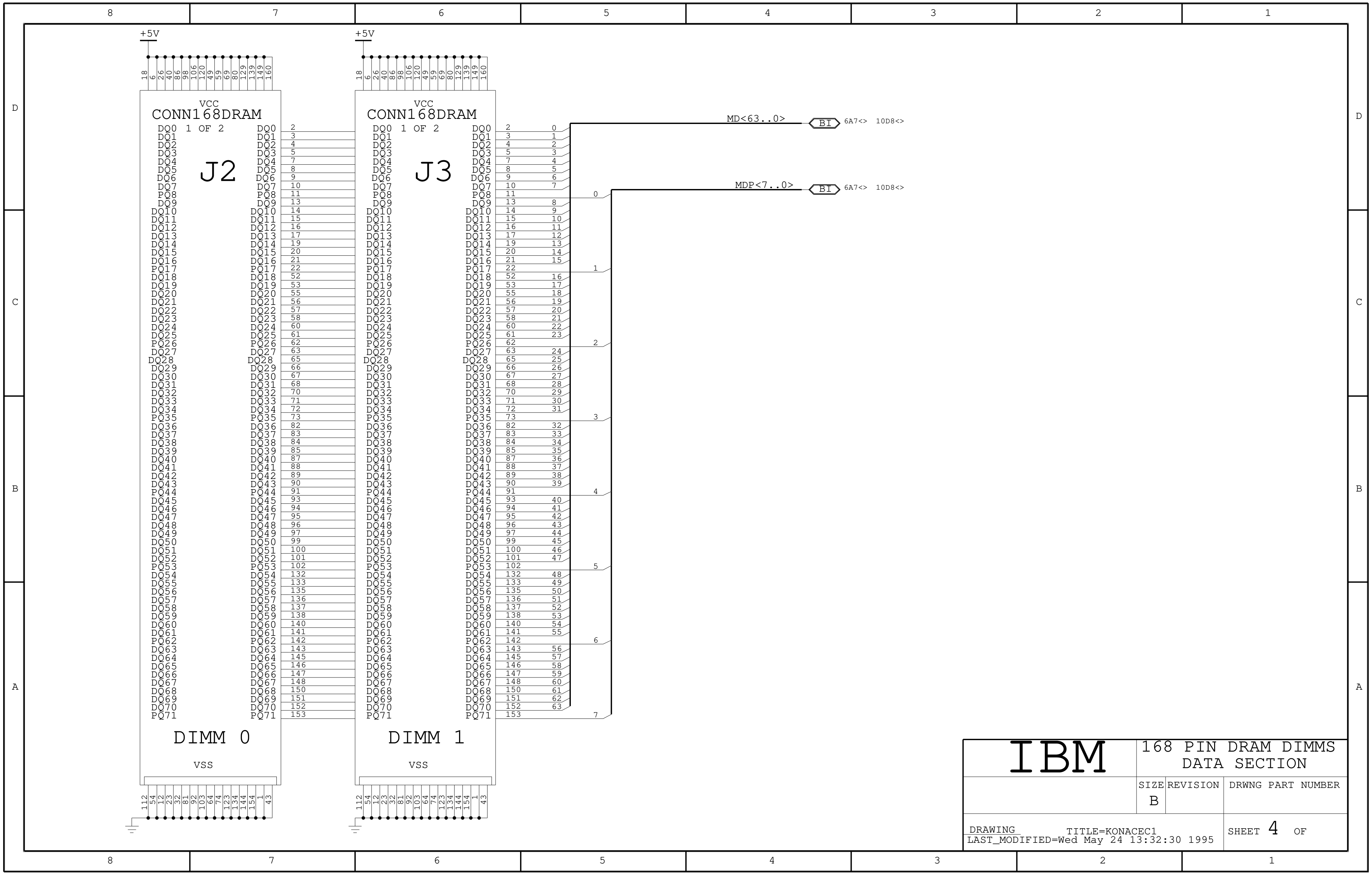
664 CONTROLLER OPERATING MODES

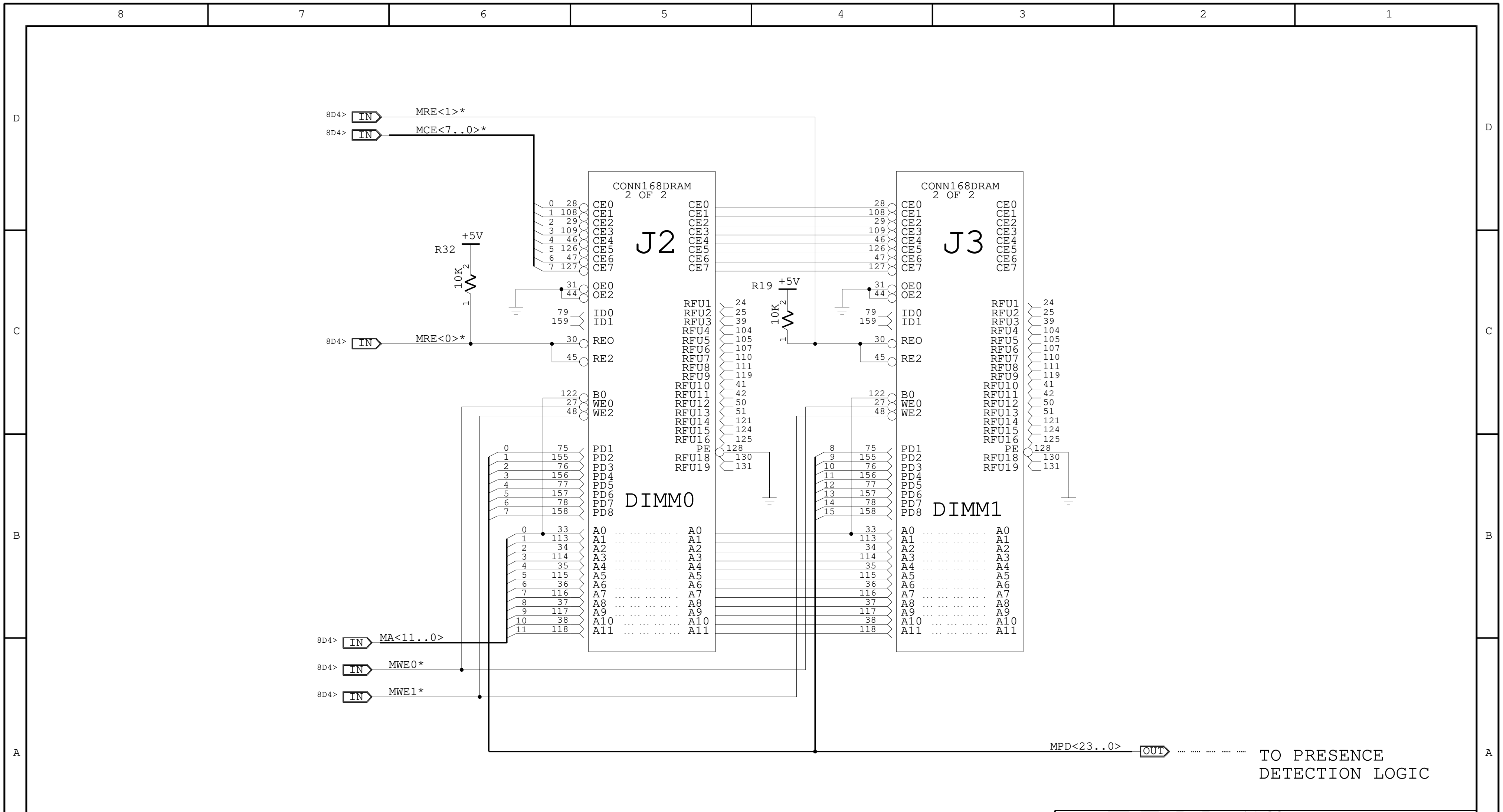
THE 664 SYSTEM CONTROLLER HAS TWO HARDWARE STRAPPING OPTIONS THAT MUST BE CONFIGURED USING JUMPERS AT J6.

J6.B1 IS CONNECTED TO MWS_P2MRXS. THIS PIN SHOULD BE PULLED UP IF THE SYSTEM PROCESSOR IS A 603 RUNNING 1:1 CLOCKING MODE. OTHERWISE THIS PIN SHOULD BE PULLED DOWN.

J6.B2 IS CONNECTED TO CRS_C2PWXS. THIS PIN SHOULD BE PULLED UP IF THE SYSTEM IPL ROM IS LOCATED BEHIND THE PCI I/O BRIDGE. OTHERWISE IT SHOULD BE PULLED DOWN TO INDICATE THAT THE SYSTEM IPL ROM IS CONNECTED DIRECTLY TO THE 664 ROM INTERFACE.

IBM	CONFIG NOTES		
SIZE B	REVISION	DRWNG PART NUMBER	
DRAWING TITLE=KONACEC1 LAST_MODIFIED=Wed May 24 13:30:19 1995		SHEET 3 OF	


+5V
VCC
CONN168DRAM
1 OF 2
J2
DIMM 0
VSS
J3
DIMM 1
MD<63..0>
BI
6A7<> 10D8<>
MDP<7..0>
IBM
168 PIN DRAM DIMMS
DATA SECTION
SIZE B
REVISION
DRWNG PART NUMBER
DRAWING
TITLE=KONACEC1
LAST_MODIFIED=Wed May 24 13:32:30 1995
SHEET 4 OF



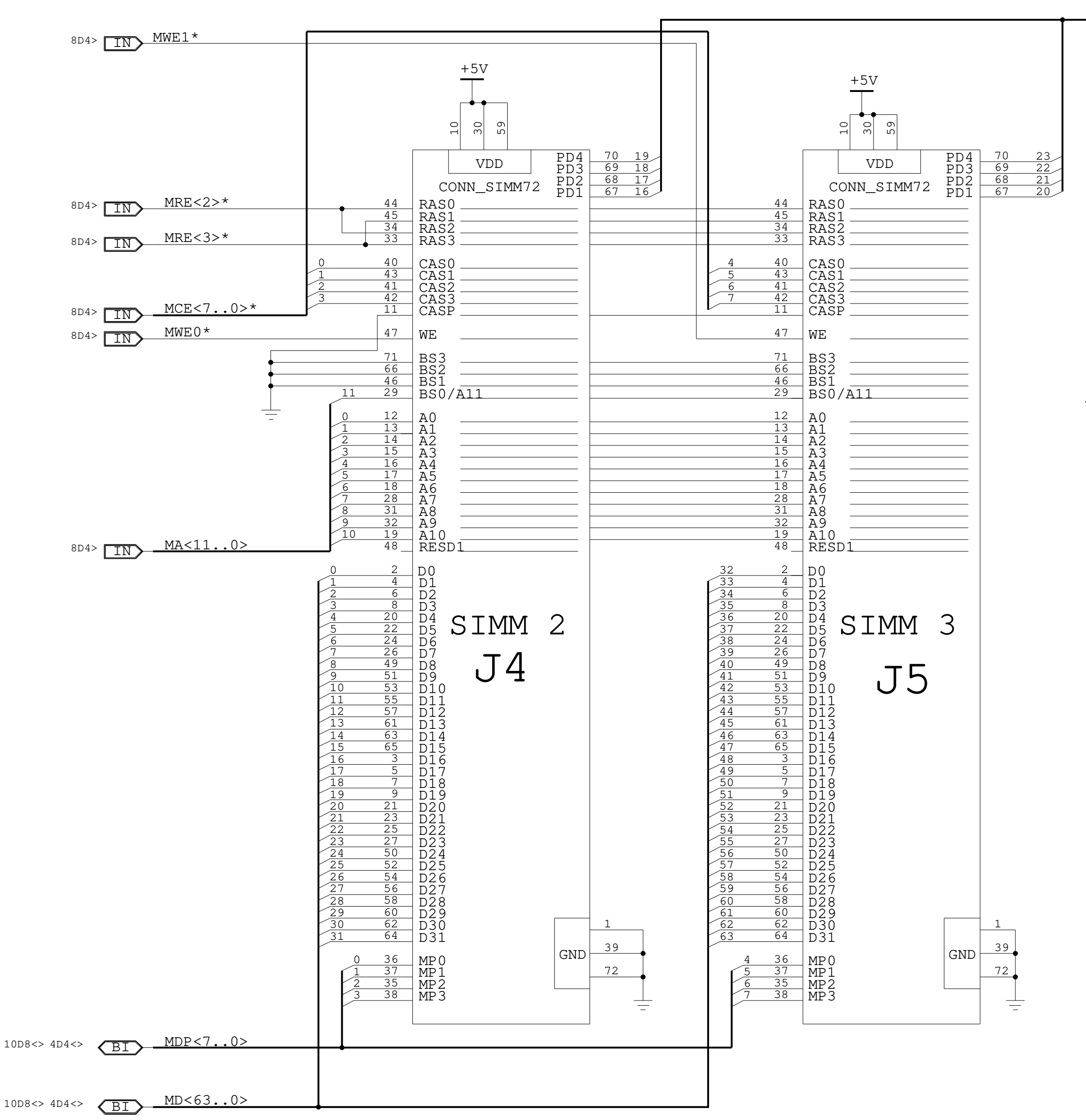
IBM

168 PIN DRAM DIMMS CONTROL SECTION

SIZE	REVISION	DRWNG PART NUMBER
B		

DRAWING TITLE=KONACEC1
LAST_MODIFIED=Wed May 24 13:32:32 1995

SHEET 5 OF



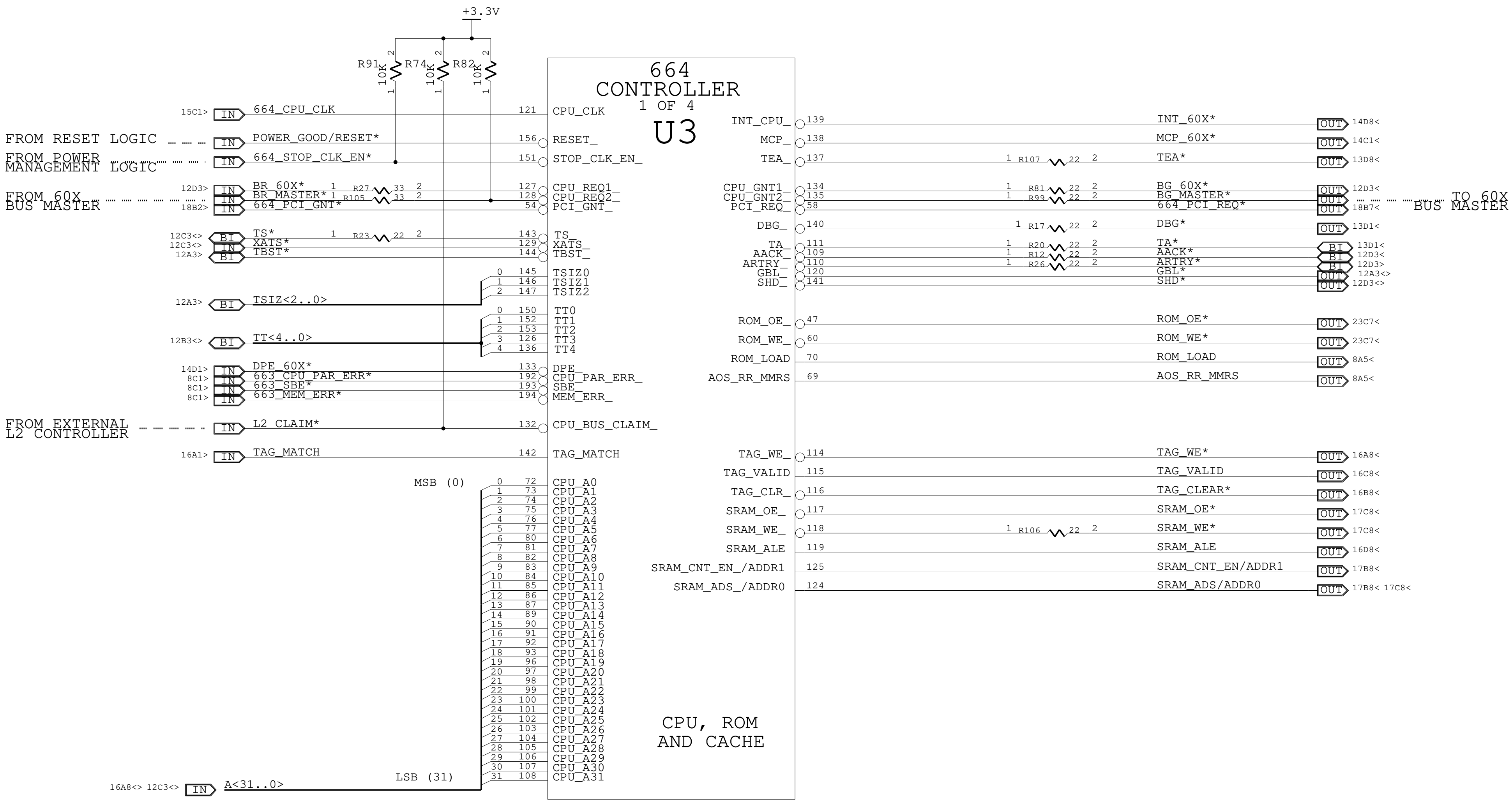
BUFFERING MA,WE,RAS, AND CAS

DESIGNERS SHOULD CONSIDER BUFFERING THE 82664 MEMORY DRIVERS (MA,WE,RAS,CAS) WHEN THE LOAD ON AN INDIVIDUAL DRIVER APPROACHES 350-400PF.

BUFFERING WILL MOST LIKELY NOT BE NECESSARY FOR DESIGNS USING 1-8 168 PIN X72 DUAL-SIDED IN-LINE MEMORY MODULES.

SOME BUFFERING MAY BE NECESSARY FOR DESIGNS USING MORE THAN TWO 72 PIN X36 SINGLE-SIDED IN-LINE MEMORY MODULES.

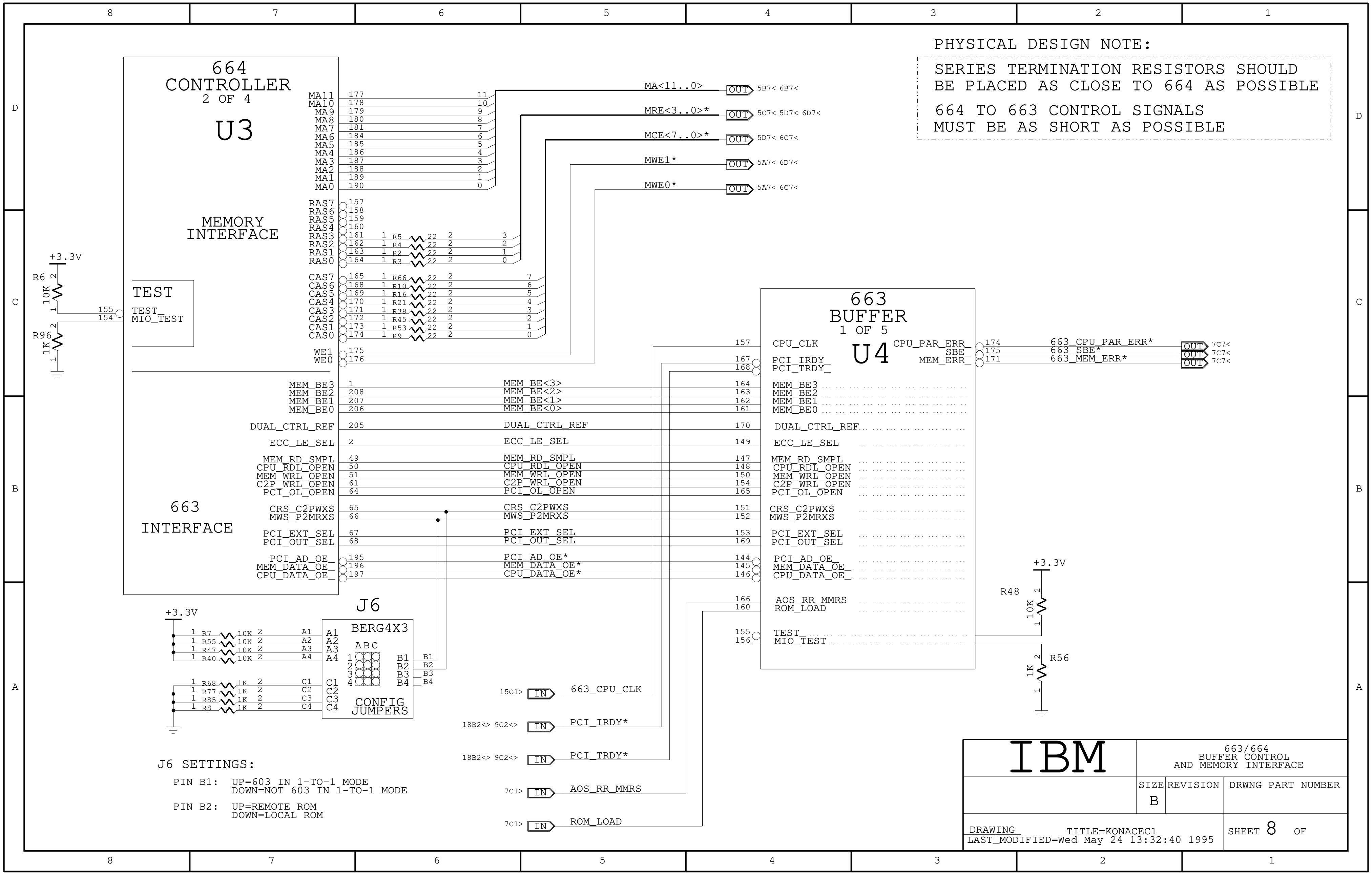
IBM	72 PIN DRAM DIMMS		
	SIZE B	REVISION	DRWNG PART NUMBER
DRAWING TITLE=KONACEC1 LAST_MODIFIED=Wed May 24 13:32:35 1995			SHEET 6 OF



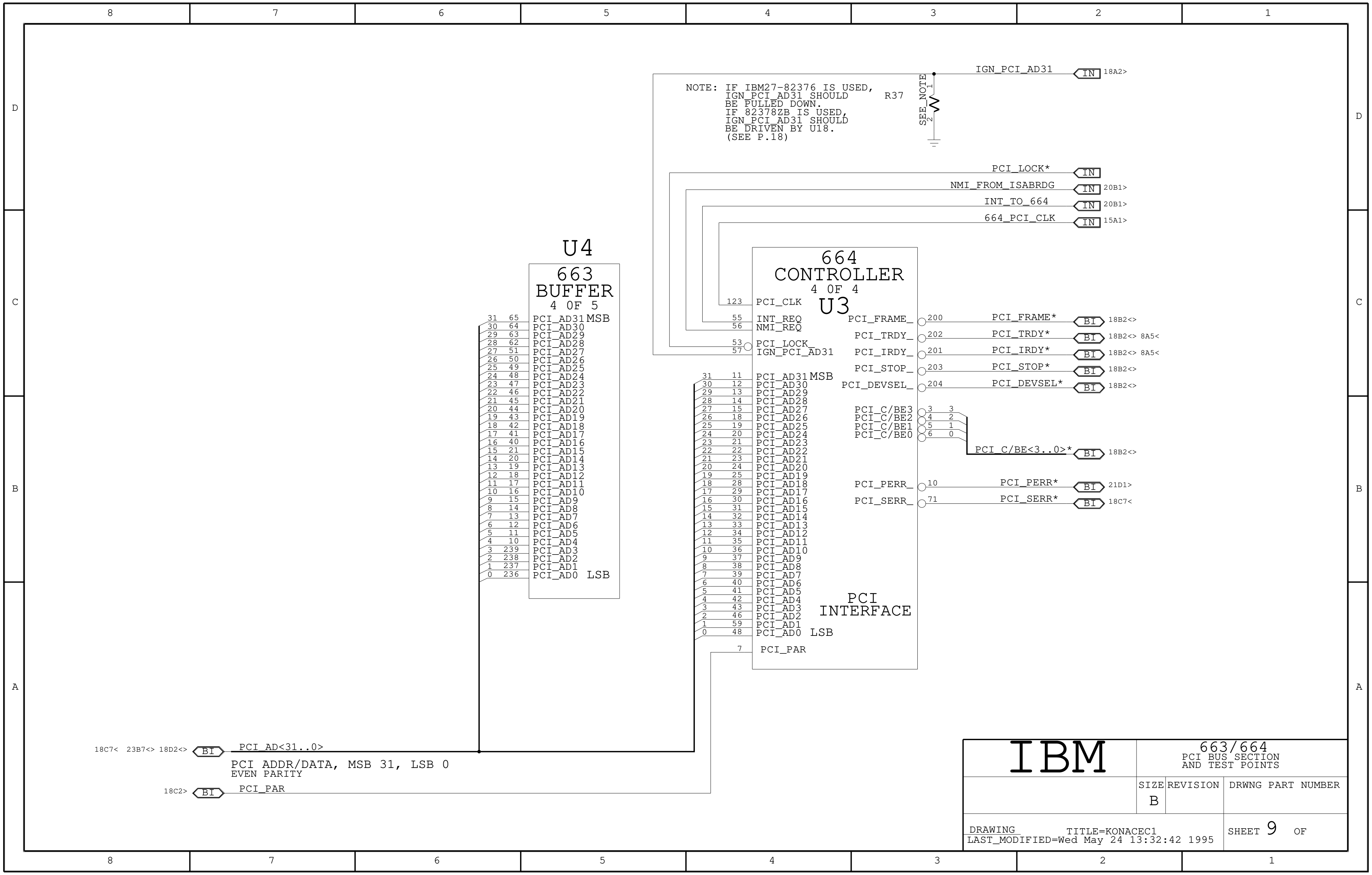
PHYSICAL DESIGN NOTE:

ALL SERIES RESISTORS MUST BE PLACED AS CLOSE TO 664 AS POSSIBLE.

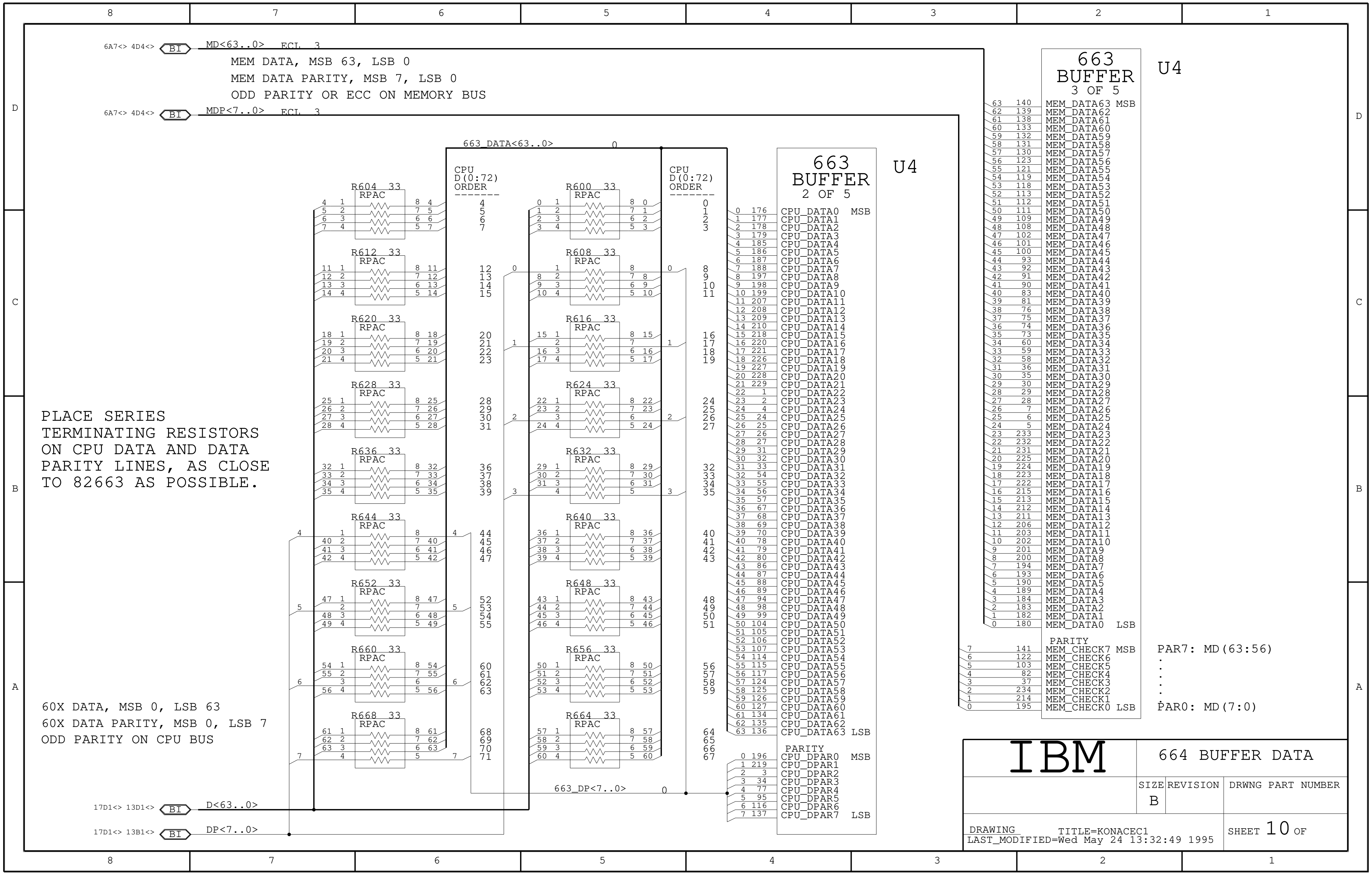
IBM	664 CONTROLLER CENTRAL CONTROL PORTION	
SIZE B	REVISION	DRWNG PART NUMBER
DRAWING TITLE=KONACEC1 LAST_MODIFIED=Wed May 24 13:32:38 1995		SHEET 7 OF



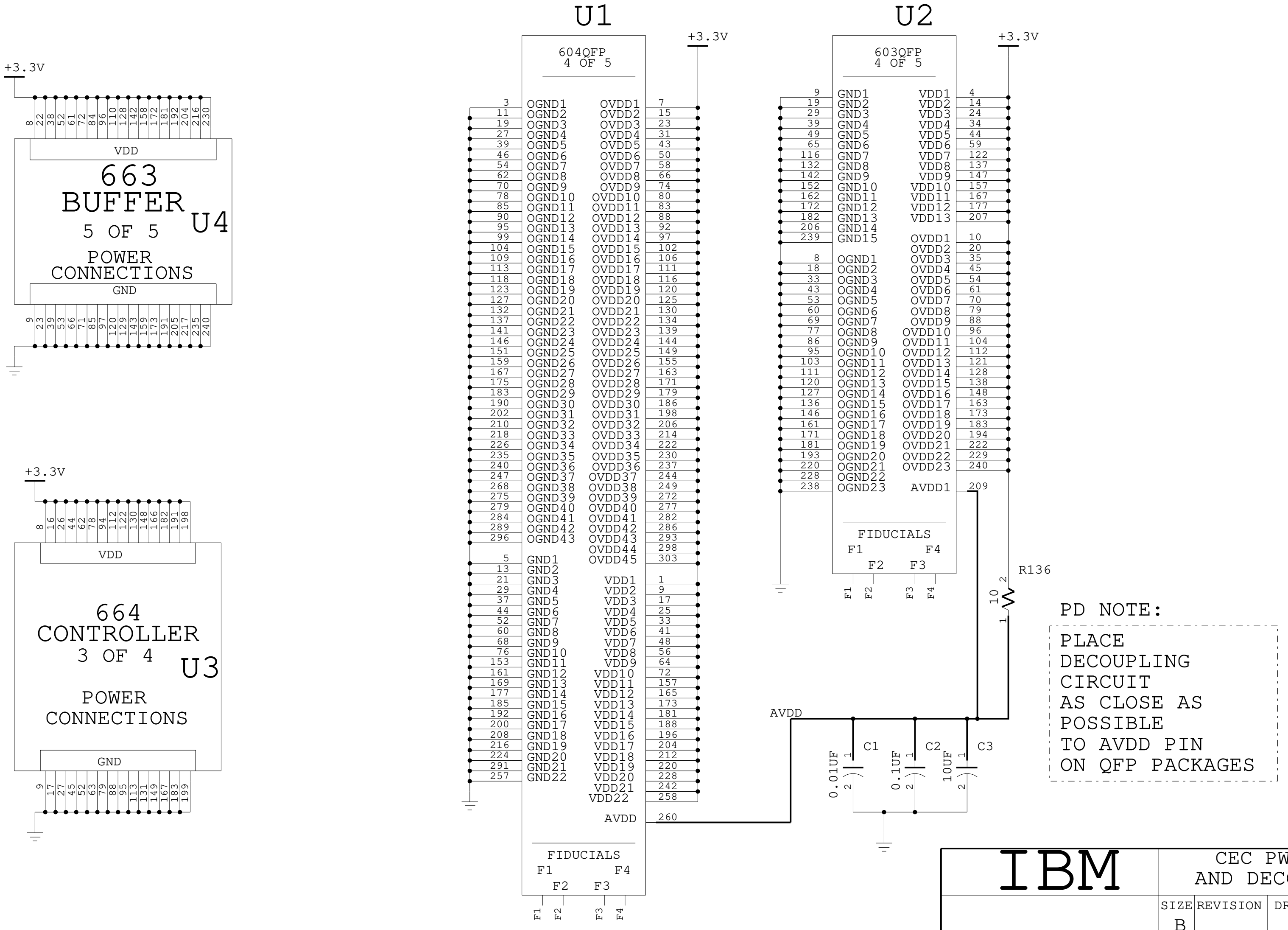
PHYSICAL DESIGN NOTE:
SERIES TERMINATION RESISTORS SHOULD BE PLACED AS CLOSE TO 664 AS POSSIBLE
664 TO 663 CONTROL SIGNALS MUST BE AS SHORT AS POSSIBLE
664 CONTROLLER 2 OF 4
U3
MEMORY INTERFACE
663 INTERFACE
TEST
663 BUFFER 1 OF 5
U4
J6
BERG4X3
CONFIG JUMPERS
MA<11..0>
MRE<3..0>*
MCE<7..0>*
MWE1*
MWE0*
663_CPU_PAR_ERR*
663_SBE*
663_MEM_ERR*
663_CPU_CLK
PCI_IRDY*
PCI_TRDY*
AOS_RR_MMRS
ROM_LOAD
J6 SETTINGS:
PIN B1: UP=603 IN 1-TO-1 MODE
DOWN=NOT 603 IN 1-TO-1 MODE
PIN B2: UP=REMOTE ROM
DOWN=LOCAL ROM
IBM
663/664
BUFFER CONTROL
AND MEMORY INTERFACE
SIZE B
REVISION
DRWNG PART NUMBER
DRAWING
TITLE=KONACEC1
LAST_MODIFIED=Wed May 24 13:32:40 1995
SHEET 8 OF



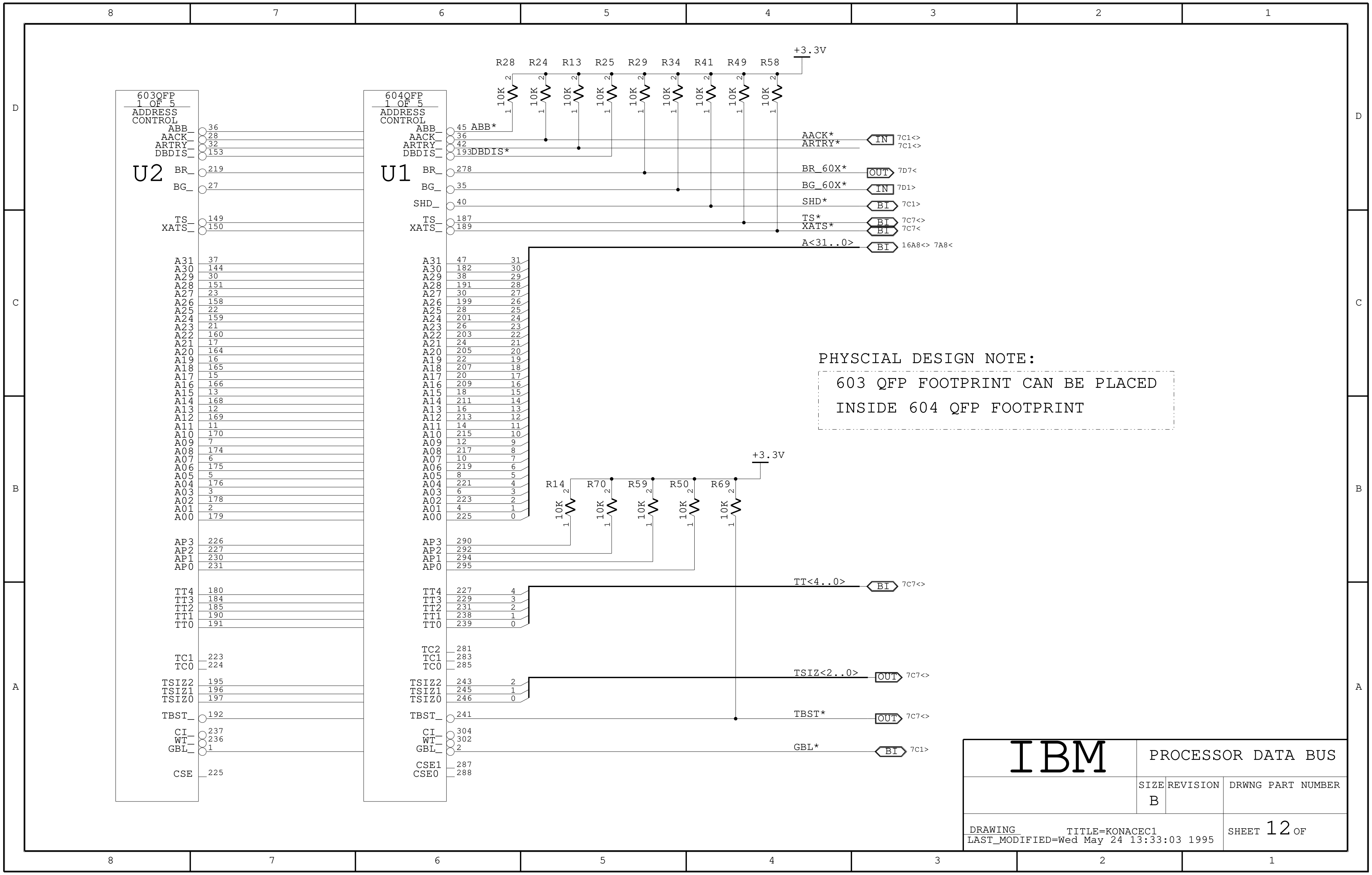
NOTE: IF IBM27-82376 IS USED, IGN_PCI_AD31 SHOULD BE PULLED DOWN.
IF 82378ZB IS USED, IGN_PCI_AD31 SHOULD BE DRIVEN BY U18.
(SEE P.18)
R37
SEE_NOTE
IGN_PCI_AD31
IN 18A2>
PCI_LOCK*
IN
NMI_FROM_ISABRDG
IN 20B1>
INT_TO_664
IN 20B1>
664_PCI_CLK
IN 15A1>
U4
663
BUFFER
4 OF 5
PCI_AD31 MSB
PCI_AD0 LSB
664
CONTROLLER
4 OF 4
U3
PCI_CLK
INT_REQ
NMI_REQ
PCI_LOCK_
IGN_PCI_AD31
PCI_AD31 MSB
PCI_AD0 LSB
PCI_PAR
PCI
INTERFACE
PCI_FRAME_
PCI_TRDY_
PCI_IRDY_
PCI_STOP_
PCI_DEVSEL_
PCI_C/BE3
PCI_C/BE2
PCI_C/BE1
PCI_C/BE0
PCI_PERR_
PCI_SERR_
PCI_FRAME*
BI 18B2<>
PCI_TRDY*
BI 18B2<> 8A5<
PCI_IRDY*
BI 18B2<> 8A5<
PCI_STOP*
BI 18B2<>
PCI_DEVSEL*
BI 18B2<>
PCI_C/BE<3..0>*
BI 18B2<>
PCI_PERR*
BI 21D1>
PCI_SERR*
BI 18C7<
18C7< 23B7<> 18D2<>
BI
PCI_AD<31..0>
PCI ADDR/DATA, MSB 31, LSB 0
EVEN PARITY
18C2>
BI
PCI_PAR
IBM
663/664
PCI BUS SECTION
AND TEST POINTS
SIZE B
REVISION
DRWNG PART NUMBER
DRAWING
TITLE=KONACEC1
LAST_MODIFIED=Wed May 24 13:32:42 1995
SHEET 9 OF



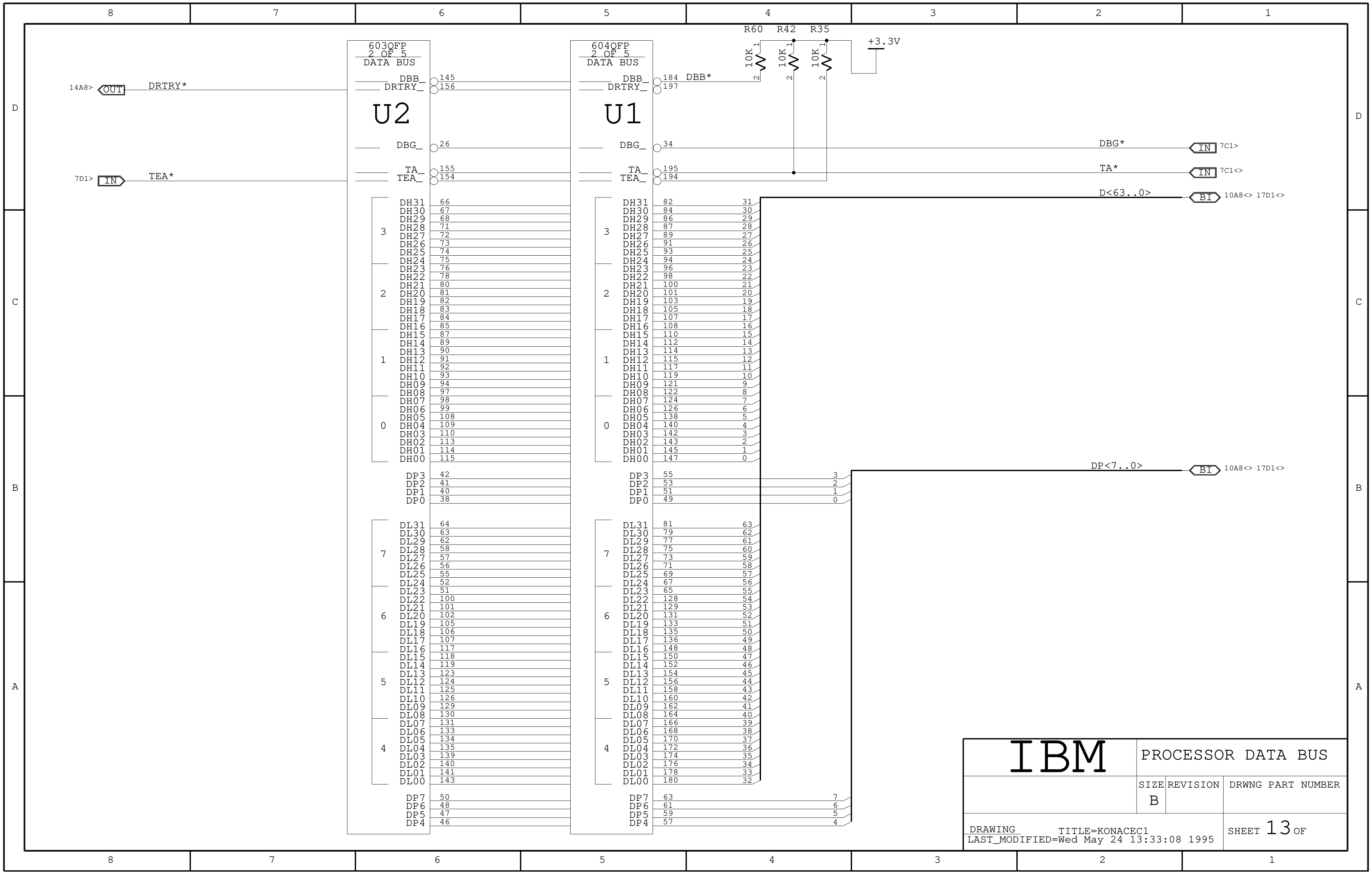
MD<63..0> ECL 3
MEM DATA, MSB 63, LSB 0
MEM DATA PARITY, MSB 7, LSB 0
ODD PARITY OR ECC ON MEMORY BUS
MDP<7..0> ECL 3
663_DATA<63..0>
CPU D(0:72) ORDER
663 BUFFER U4 2 OF 5
663 BUFFER U4 3 OF 5
PLACE SERIES TERMINATING RESISTORS ON CPU DATA AND DATA PARITY LINES, AS CLOSE TO 82663 AS POSSIBLE.
60X DATA, MSB 0, LSB 63
60X DATA PARITY, MSB 0, LSB 7
ODD PARITY ON CPU BUS
663_DP<7..0>
D<63..0>
DP<7..0>
PARITY
PAR7: MD(63:56)
PAR0: MD(7:0)
IBM
664 BUFFER DATA
SIZE B
REVISION
DRWNG PART NUMBER
DRAWING
TITLE=KONACEC1
LAST_MODIFIED=Wed May 24 13:32:49 1995
SHEET 10 OF

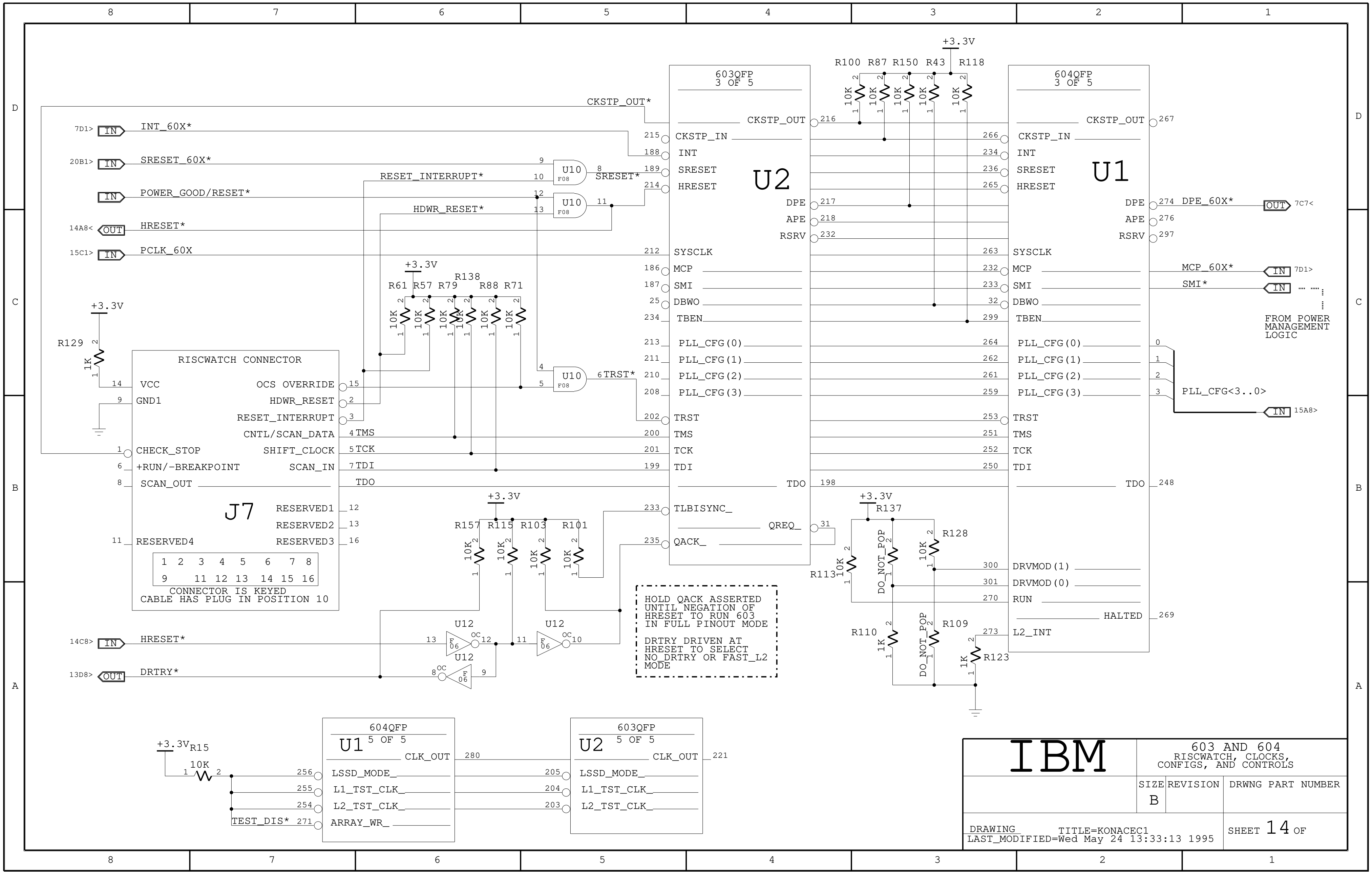


IBM	CEC PWR/GNDS AND DECOUPLING		
	SIZE B	REVISION	DRWNG PART NUMBER
DRAWING TITLE=KONACEC1 LAST_MODIFIED=Wed May 24 13:32:59 1995			SHEET 11 OF

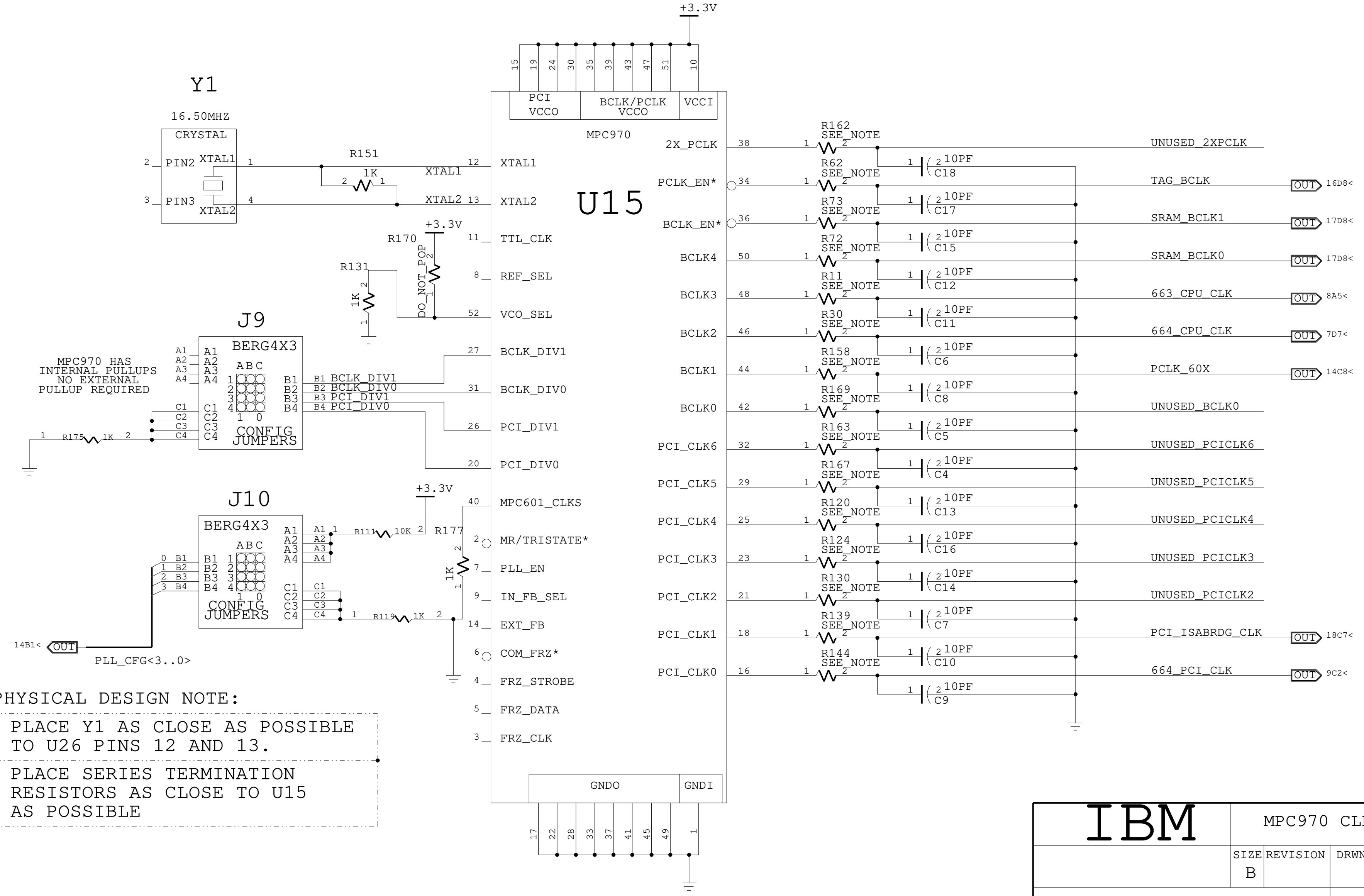


603QFP 1 OF 5 ADDRESS CONTROL
U2
604QFP 1 OF 5 ADDRESS CONTROL
U1
R28 R24 R13 R25 R29 R34 R41 R49 R58
10K
+3.3V
ABB* DBDIS*
ABB_ AACK_ ARTRY_ DBDIS_ BR_ BG_ SHD_ TS_ XATS_
AACK* 7C1<>
ARTRY* 7C1<>
BR_60X* 7D7<
BG_60X* 7D1>
SHD* 7C1>
TS* 7C7<>
XATS* 7C7<
A<31..0> 16A8<> 7A8<
A31 … A00
AP3 AP2 AP1 AP0
R14 R70 R59 R50 R69
TT4 TT3 TT2 TT1 TT0
TT<4..0> 7C7<>
TC2 TC1 TC0
TSIZ2 TSIZ1 TSIZ0
TSIZ<2..0> 7C7<>
TBST_
TBST* 7C7<>
CI_ WT_ GBL_
GBL* 7C1>
CSE CSE1 CSE0
PHYSCIAL DESIGN NOTE:
603 QFP FOOTPRINT CAN BE PLACED
INSIDE 604 QFP FOOTPRINT
IBM
PROCESSOR DATA BUS
SIZE B
REVISION
DRWNG PART NUMBER
DRAWING
TITLE=KONACEC1
LAST_MODIFIED=Wed May 24 13:33:03 1995
SHEET 12 OF


603QFP
2 OF 5
DATA BUS
U2
604QFP
2 OF 5
DATA BUS
U1
DRTRY*
TEA*
DBB*
R60
R42
R35
10K
+3.3V
DBG*
TA*
D<63..0>
DP<7..0>
IN
OUT
BI
IBM
PROCESSOR DATA BUS
SIZE
REVISION
DRWNG PART NUMBER
B
DRAWING
TITLE=KONACEC1
LAST_MODIFIED=Wed May 24 13:33:08 1995
SHEET 13 OF



603QFP 3 OF 5
U2
604QFP 3 OF 5
U1
CKSTP_OUT*
INT_60X*
SRESET_60X*
POWER_GOOD/RESET*
HRESET*
PCLK_60X
RESET_INTERRUPT*
HDWR_RESET*
SRESET*
TRST*
U10
F08
CKSTP_OUT
CKSTP_IN
INT
SRESET
HRESET
DPE
APE
RSRV
SYSCLK
MCP
SMI
DBWO
TBEN
PLL_CFG(0)
PLL_CFG(1)
PLL_CFG(2)
PLL_CFG(3)
TRST
TMS
TCK
TDI
TDO
TLBISYNC_
QREQ_
QACK_
DRVMOD(1)
DRVMOD(0)
RUN
HALTED
L2_INT
DPE_60X*
MCP_60X*
SMI*
FROM POWER MANAGEMENT LOGIC
PLL_CFG<3..0>
+3.3V
R100 R87 R150 R43 R118
R61 R57 R79 R138 R88 R71
R129
R157 R115 R103 R101
R137 R128 R113 R110 R109 R123
DO_NOT_POP
10K
1K
RISCWATCH CONNECTOR
VCC
GND1
CHECK_STOP
+RUN/-BREAKPOINT
SCAN_OUT
RESERVED4
OCS OVERRIDE
HDWR_RESET
RESET_INTERRUPT
CNTL/SCAN_DATA
SHIFT_CLOCK
SCAN_IN
RESERVED1
RESERVED2
RESERVED3
J7
1 2 3 4 5 6 7 8
9 11 12 13 14 15 16
CONNECTOR IS KEYED
CABLE HAS PLUG IN POSITION 10
U12
DRTRY*
HOLD QACK ASSERTED UNTIL NEGATION OF HRESET TO RUN 603 IN FULL PINOUT MODE
DRTRY DRIVEN AT HRESET TO SELECT NO_DRTRY OR FAST_L2 MODE
604QFP 5 OF 5
603QFP 5 OF 5
R15
CLK_OUT
LSSD_MODE_
L1_TST_CLK_
L2_TST_CLK_
ARRAY_WR_
TEST_DIS*
IBM
603 AND 604
RISCWATCH, CLOCKS, CONFIGS, AND CONTROLS
SIZE B
REVISION
DRWNG PART NUMBER
DRAWING
TITLE=KONACEC1
LAST_MODIFIED=Wed May 24 13:33:13 1995
SHEET 14 OF

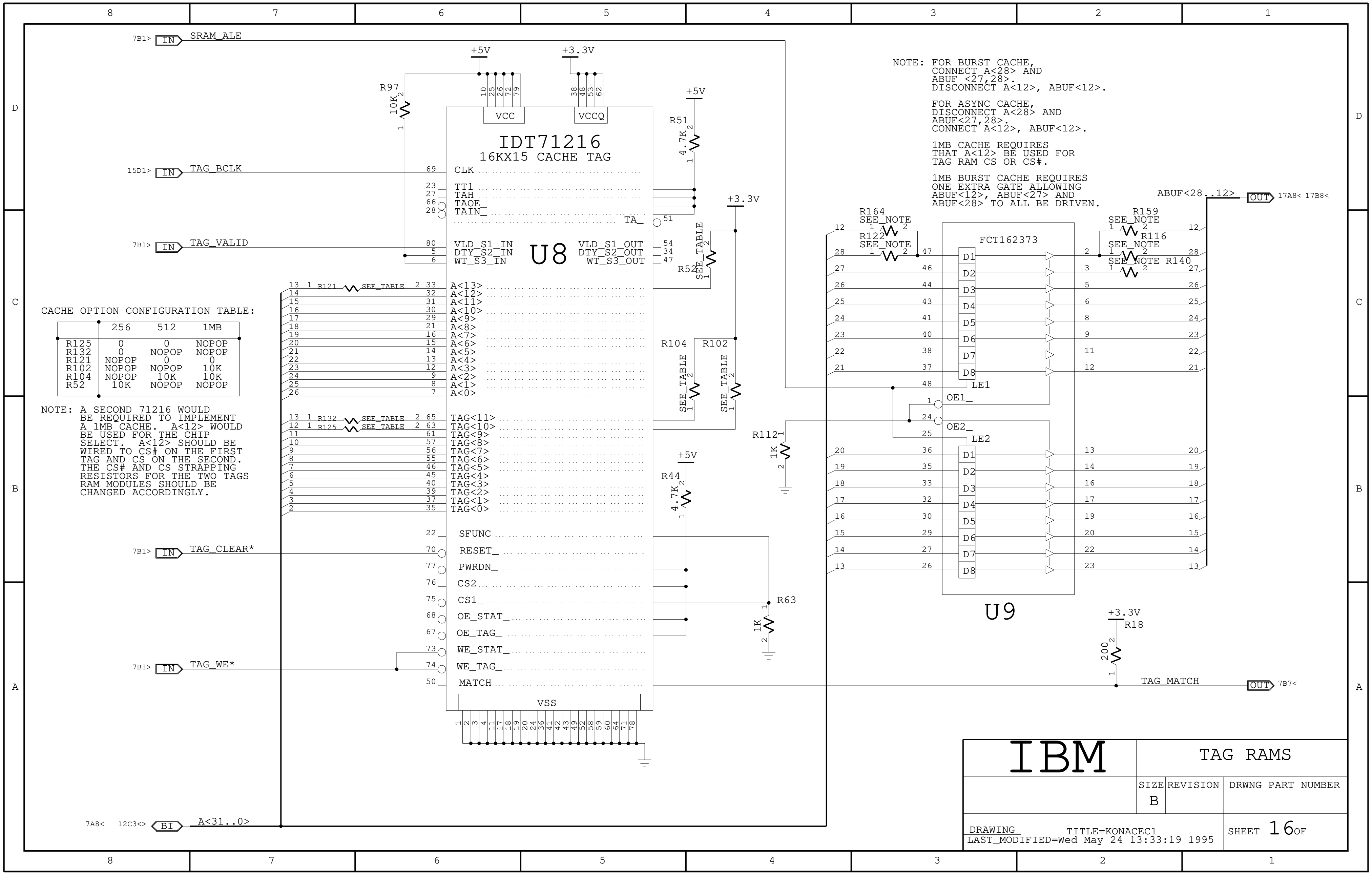


PHYSICAL DESIGN NOTE:

PLACE Y1 AS CLOSE AS POSSIBLE TO U26 PINS 12 AND 13.

PLACE SERIES TERMINATION RESISTORS AS CLOSE TO U15 AS POSSIBLE

IBM	MPC970 CLK CHIP		
	SIZE B	REVISION	DRWNG PART NUMBER
DRAWING TITLE=KONACEC1 LAST_MODIFIED=Wed May 24 13:33:17 1995			SHEET 15 OF



CACHE OPTION CONFIGURATION TABLE:

	256	512	1MB
R125	0	0	NOPOP
R132	0	NOPOP	NOPOP
R121	NOPOP	0	0
R102	NOPOP	NOPOP	10K
R104	NOPOP	10K	10K
R52	10K	NOPOP	NOPOP

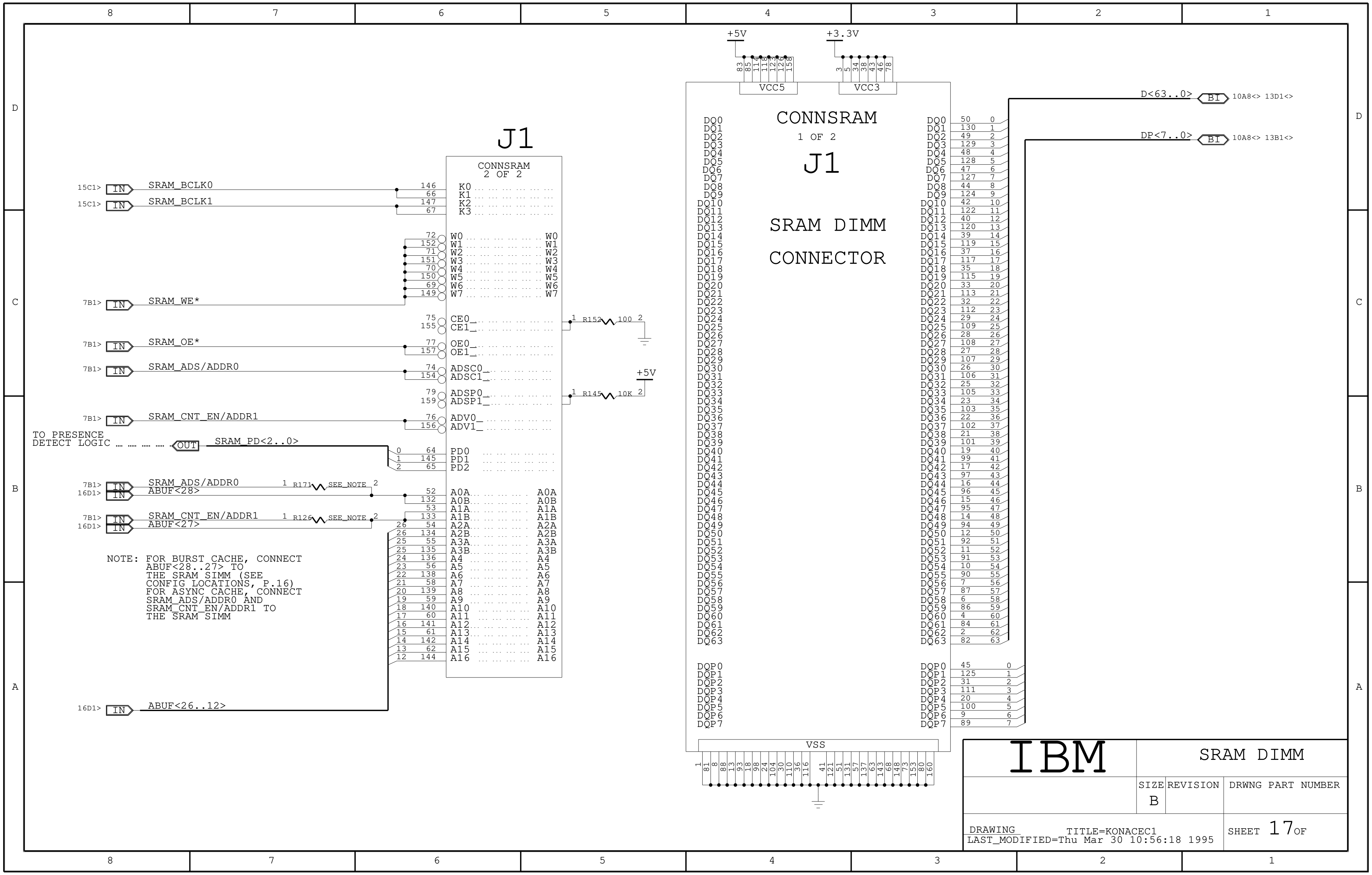
IBM

TAG RAMS

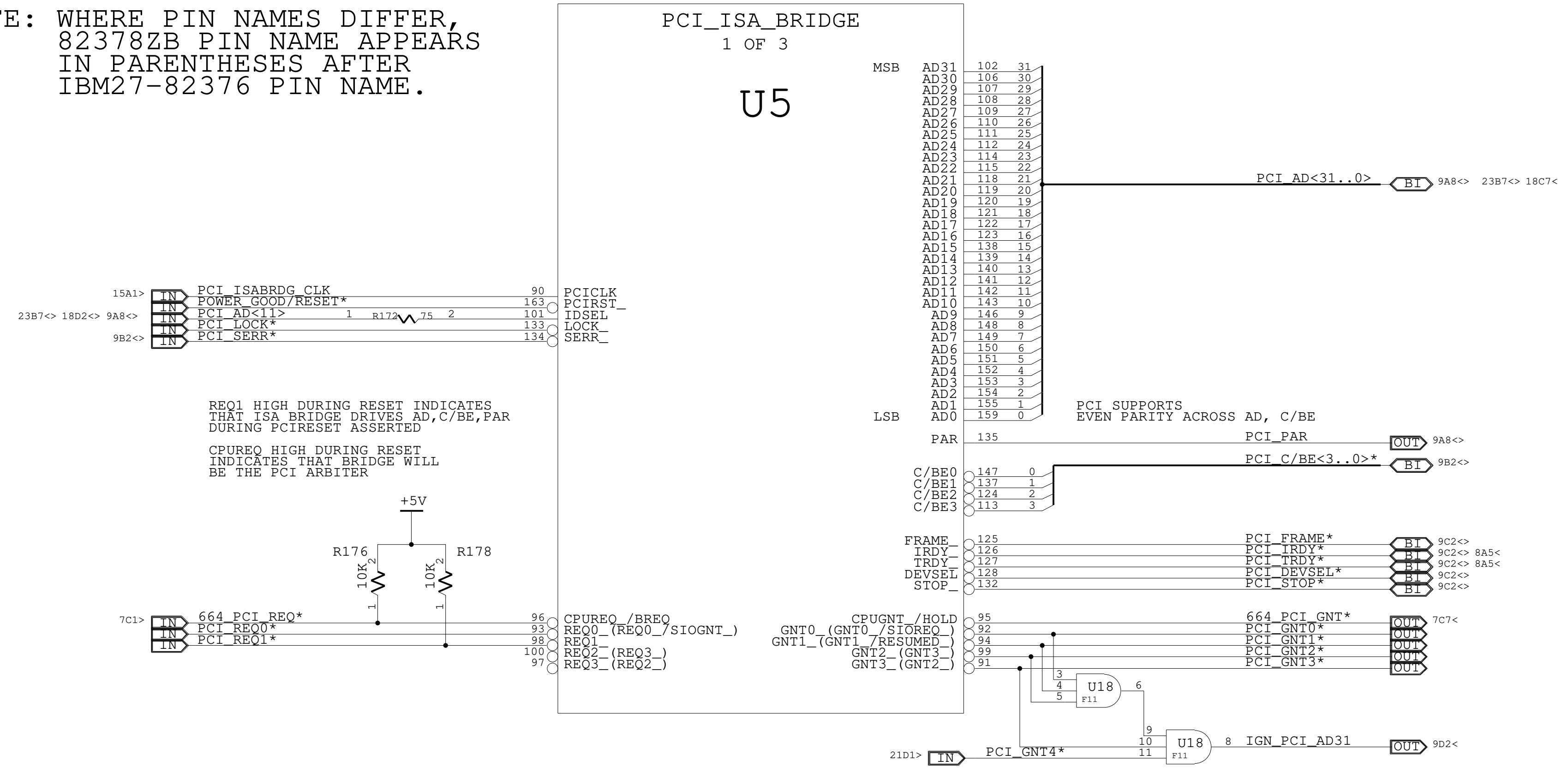
SIZE B | REVISION | DRWNG PART NUMBER

DRAWING TITLE=KONACEC1
LAST_MODIFIED=Wed May 24 13:33:19 1995

SHEET 16 OF


J1
CONNSRAM
2 OF 2
SRAM_BCLK0
SRAM_BCLK1
SRAM_WE*
SRAM_OE*
SRAM_ADS/ADDR0
SRAM_CNT_EN/ADDR1
TO PRESENCE DETECT LOGIC
SRAM_PD<2..0>
ABUF<28>
ABUF<27>
ABUF<26..12>
R152 100
R145 10K
R171 SEE NOTE
R126 SEE NOTE
+5V
+3.3V
VCC5
VCC3
VSS
CONNSRAM
1 OF 2
J1
SRAM DIMM
CONNECTOR
D<63..0>
DP<7..0>
10A8<> 13D1<>
10A8<> 13B1<>
NOTE: FOR BURST CACHE, CONNECT ABUF<28..27> TO THE SRAM SIMM (SEE CONFIG LOCATIONS, P.16) FOR ASYNC CACHE, CONNECT SRAM_ADS/ADDR0 AND SRAM_CNT_EN/ADDR1 TO THE SRAM SIMM
IBM
SRAM DIMM
SIZE B
REVISION
DRWNG PART NUMBER
DRAWING
TITLE=KONACEC1
LAST_MODIFIED=Thu Mar 30 10:56:18 1995
SHEET 17 OF

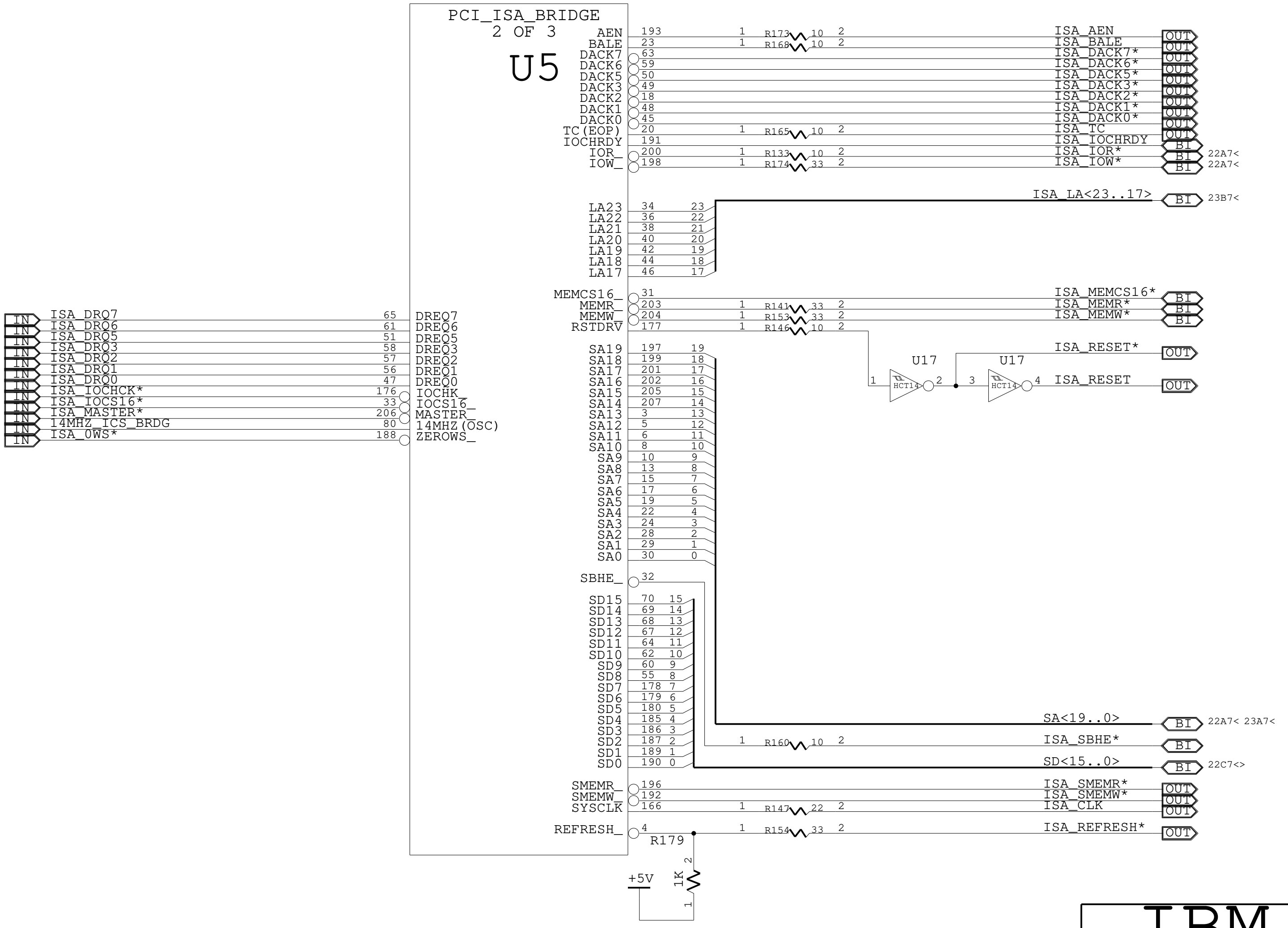
NOTE: WHERE PIN NAMES DIFFER,
82378ZB PIN NAME APPEARS
IN PARENTHESES AFTER
IBM27-82376 PIN NAME.



NOTE: CASCADING 'AND' GATES NECESSARY ONLY
IF 82378ZB IS INSTALLED. IGN_PCI_AD31
INDICATES TO 664 THAT THE PCI/ISA BRIDGE
IS THE PCI MASTER AND THAT THE BRIDGE
MAY BE RUNNING A CYCLE FOR AN ISA
MASTER.

THE USER SHOULD TAKE INTO CONSIDERATION
THE TIMING CONSTRAINTS OF THE 664
IGN_PCI_AD31 INPUT IN DESIGNING THIS
CASCADE CIRCUIT.

IBM	IBM27-82376, 82378ZB COMMON PINS-PCI PORTION		
	SIZE B	REVISION	DRWNG PART NUMBER
DRAWING TITLE=KONACEC1 LAST_MODIFIED=Thu Mar 30 10:56:59 1995			SHEET 18 OF



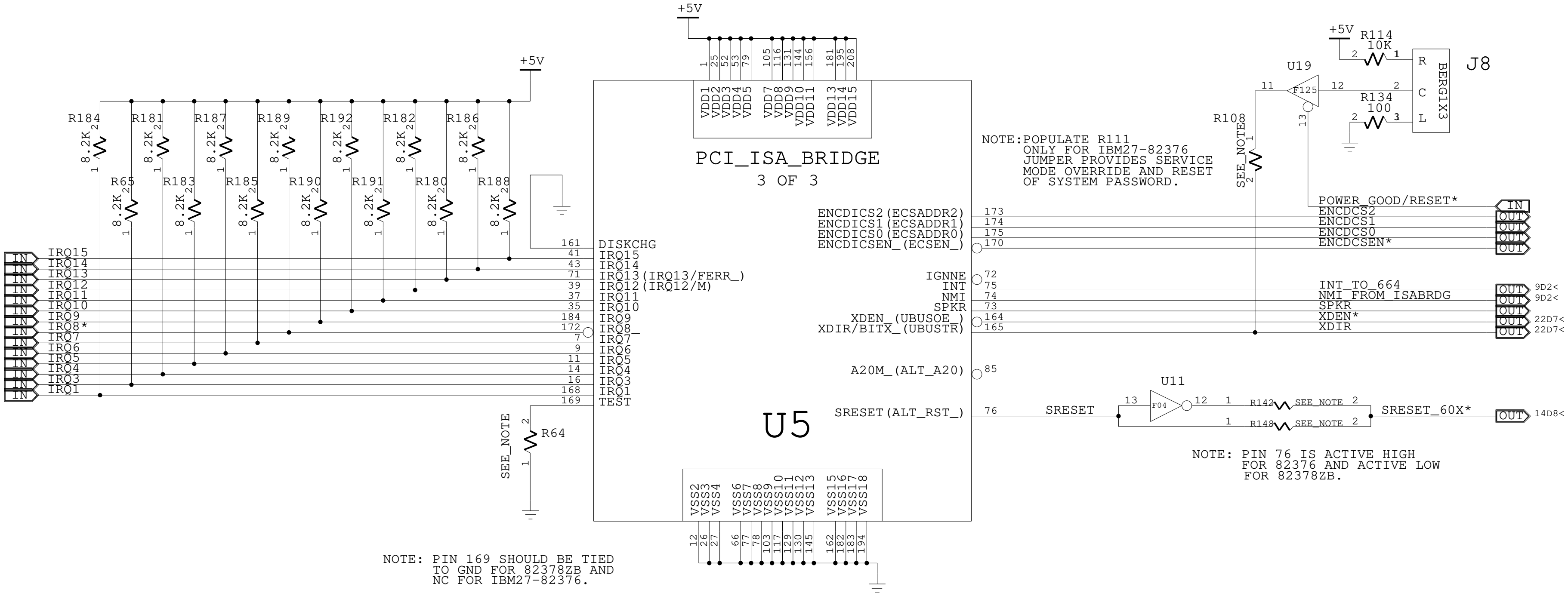
IBM

IBM27-82376, 82378ZB
COMMON PINS-ISA PORTION

SIZE	REVISION	DRWNG PART NUMBER
B		

DRAWING
TITLE=KONACEC1
LAST_MODIFIED=Thu Mar 30 10:57:13 1995

SHEET 19 OF



IBM

IBM27-82376, 82378ZB
COMMON PINS

SIZE	REVISION	DRWNG PART NUMBER
B		

DRAWING TITLE=KONACEC1
LAST_MODIFIED=Wed May 24 13:33:01 1995

SHEET 20 OF

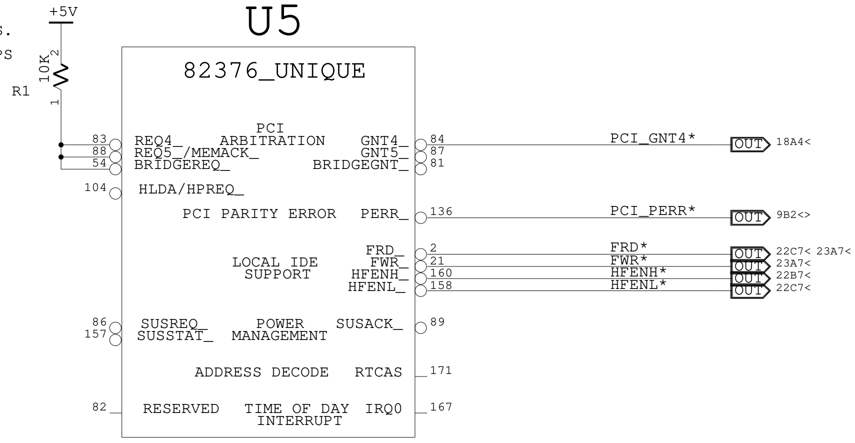
IBM27-82376/82378ZB COMMON PINS WITH SPECIAL CONSIDERATIONS

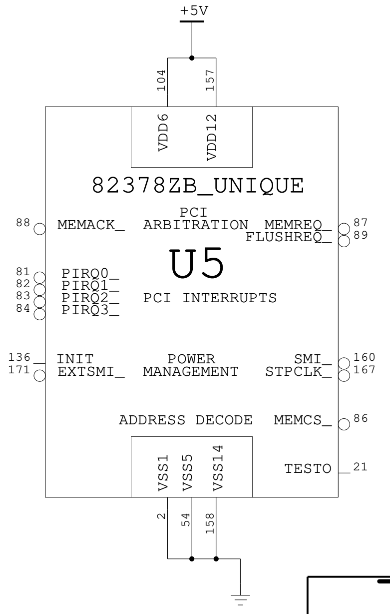
PIN #	82378ZB		IBM27-82376	
42	LA19	(I/O)	LA19/ROMSEC	(O)
44	LA18	(I/O)	LA18/ROMBANK	(O)
46	LA17	(I/O)	LA17	(O)
76	ALT_RST_	(O)	SRESET	(O)
85	ALT_A20_	(O)	A20M_	(O)
91	GNT2_	(O)	GNT3_	(O)
92	GNT0_/SIOREQ_	(O)	GNT0_	(O)
93	REQ0_/SIOGNT_	(I)	REQ0_	(I)
94	GNT1_/RESUME_	(O)	GNT1_	(O)
95	CPUGNT_	(O)	CPUGNT_/HOLD	(O)
96	CPUREQ_	(I)	CPUREQ_/BREQ	(I)
97	REQ2_	(I)	REQ3_	(I)
99	GNT3_	(O)	GNT2_	(O)
100	REQ3_	(I)	REQ2_	(I)
134	SERR_	(I)	SERR_	(I/O)
135	PAR	(O)	PAR	(I/O)
165	UBUSTR	(O)	XDIR/BITX_	(I/O)
169	TEST (GND)	(I)	TEST(NC)	(I)
176	IOCHK_	(I)	IOCHK_	(I/O)

IBM27-82376/82378ZB UNIQUE PINOUT

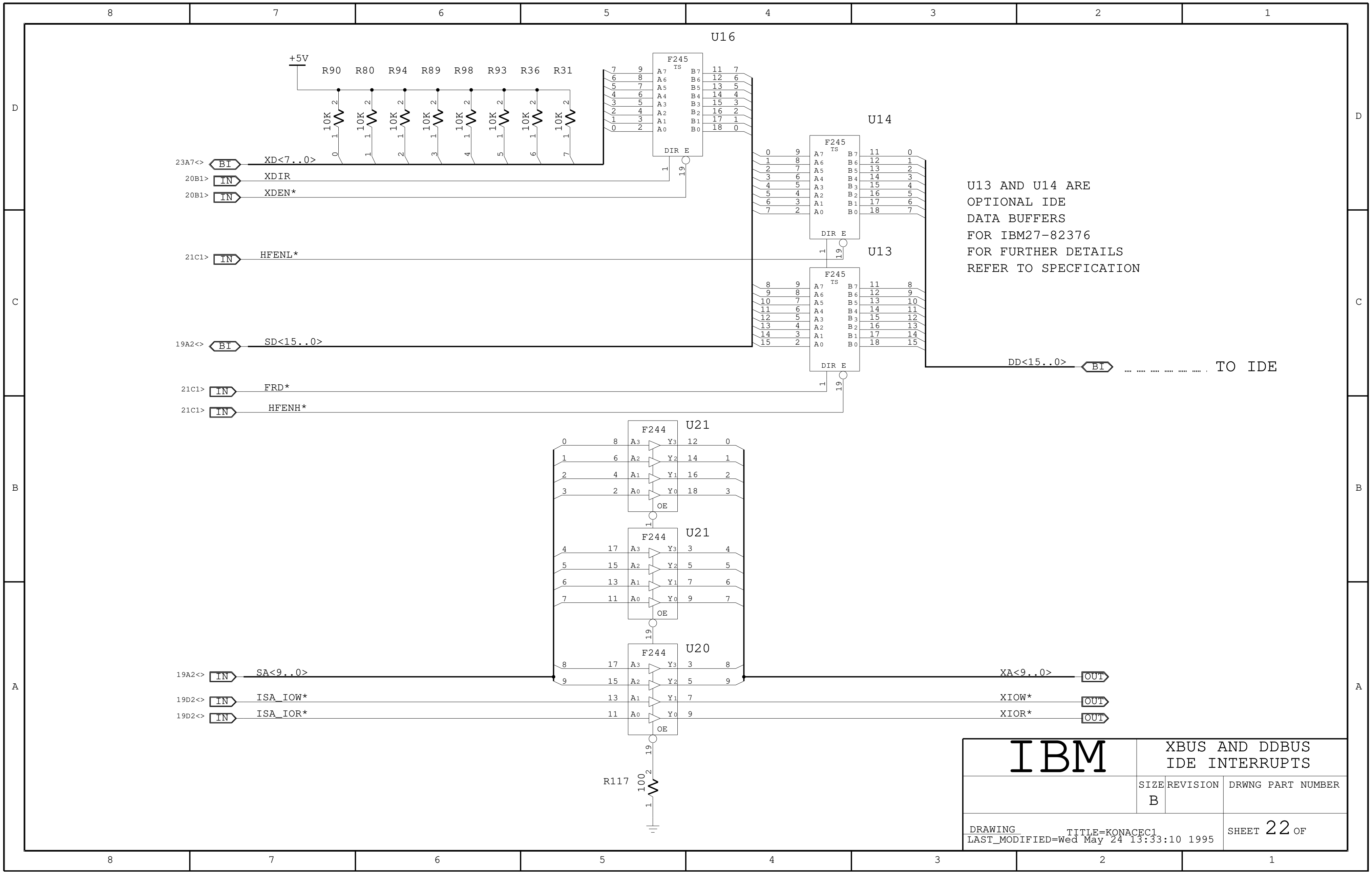
PIN #	82378ZB	IBM27-82376	PIN #	82378ZB	IBM27-82376
2	VSS	FRD_	88	MEMACK_	REQ5_/MEMACK_
21	TESTO	FWR_	89	FLUSHREQ_	SUSACK_
54	VSS	BRIDGE_REQ_	104	VDD	HLDA/HPREQ_
81	PIRQ0_	BRIDGE_GNT_	136	INIT	PERR_
82	PIRQ1_	RESERVED	157	VDD	SUSSTAT_
83	PIRQ2_	REQ4_	158	VSS	HFENL_
84	PIRQ3_	GNT4_	160	SMI_	HFENH_
86	MEMCS_	SUSREQ_	167	STPCLK_	IRQ0
87	MEMREQ_	GNT5_/MEMREQ_	171	EXTSMI_	RTCAS

NOTE: REQUEST PINS HAVE WEAK INTERNAL PULL UPS. THE USER MAY WISH TO PLACE EXTERNAL PULL UPS ON THESE PINS.





IBM	IBM27-82376 AND 82378ZB UNIQUE PINS		
	SIZE B	REVISION	DRWNG PART NUMBER
DRAWING TITLE=KONACEC1 LAST_MODIFIED=Wed May 24 13:33:05 1995			SHEET 21 OF



IBM

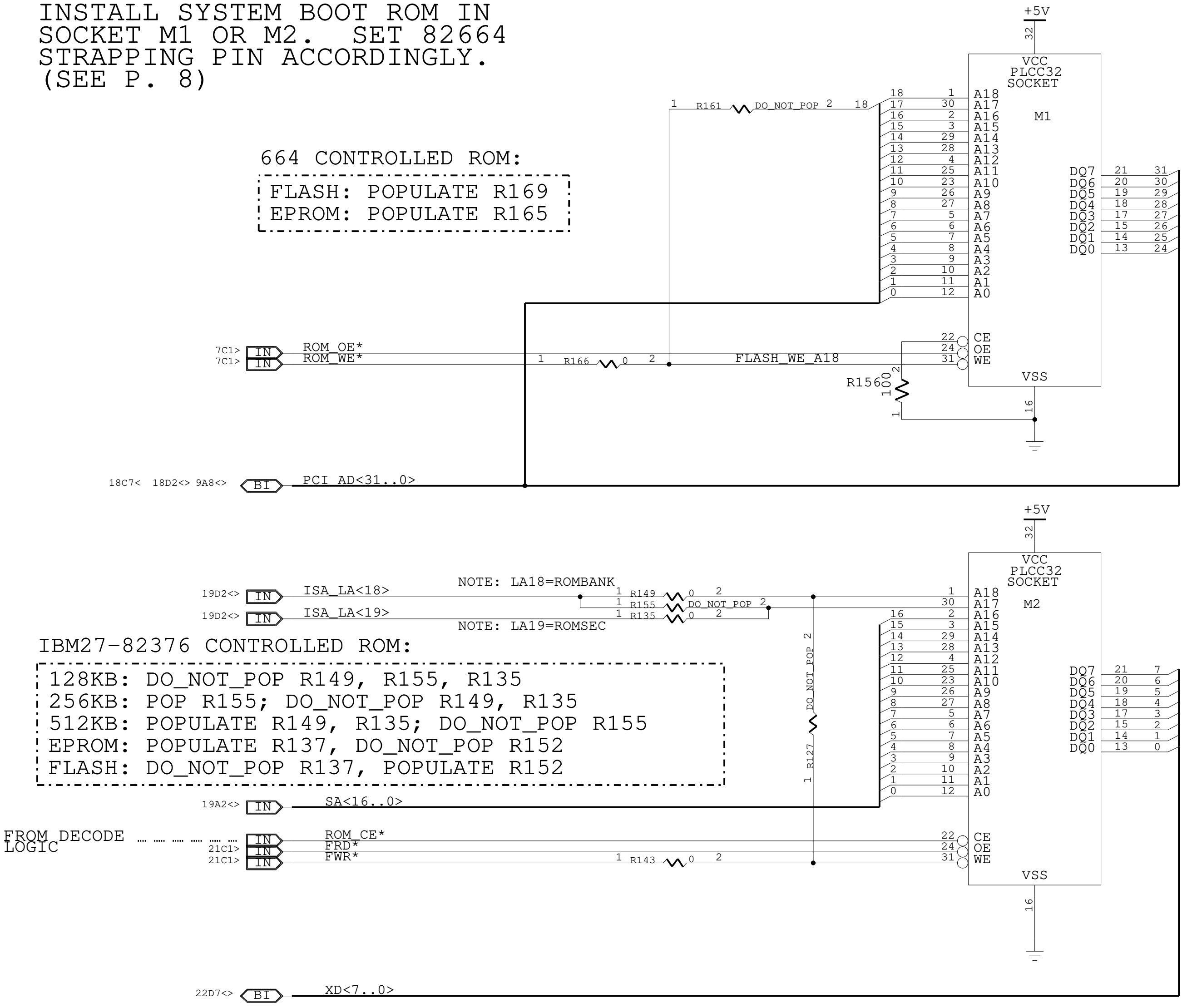
XBUS AND DDBUS
IDE INTERRUPTS

SIZE B | REVISION | DRWNG PART NUMBER

DRAWING TITLE=KONACEC1
LAST_MODIFIED=Wed May 24 13:33:10 1995

SHEET 22 OF

INSTALL SYSTEM BOOT ROM IN
SOCKET M1 OR M2. SET 82664
STRAPPING PIN ACCORDINGLY.
(SEE P. 8)



NOTE: 664 HAS A LOCAL ROM INTERFACE THAT ALLOWS THE SYSTEM TO ACCESS A BOOT ROM WIRED TO PCI_AD<31..0>. SEE P.8 FOR THE PROPER SETTING OF JUMPER J6 TO CONFIGURE 664 FOR LOCAL ROM ACCESSES.

NOTE: 664 CAN ACCESS A BOOT ROM LOCATED ON ISA (OR SOME OTHER TERTIARY BUS) PROVIDED THAT THE PCI BRIDGE WILL DECODE ROM CYCLES TO THE UPPER 2MB OF THE 4GB ADDRESS SPACE. IBM27-82376 WILL DECODE ROM CYCLES IN THIS ADDRESS RANGE. SEE P.8 FOR THE PROPER SETTING OF JUMPER J6 TO CONFIGURE 664 FOR REMOTE ROM ACCESSES.

IBM	664 LOCAL BOOT ROM IBM27-82376 ISA BOOT ROM	
	SIZE B / REVISION	DRWNG PART NUMBER
DRAWING TITLE=KONACEC1 LAST_MODIFIED=Wed May 24 13:33:15 1995		SHEET 23 OF

```
      *** Signal Cross-Reference ***
         --- for the entire design --

14MHZ_ICS_BRDG  19C8<
663_CPU_CLK     15C1> 8A5<
663_CPU_PAR_ERR * 8C1> 7C7<
663_DATA <63..0> 10D6
663_DP <7..0>   10A5
663_MEM_ERR *   8C1> 7C7<
663_SBE *       8C1> 7C7<
664_CPU_CLK     15C1> 7D7<
664_PCI_CLK     15A1> 9C2<
664_PCI_GNT *   18B2> 7C7<
664_PCI_REQ *   7C1> 18B7<
664_STOP_CLK_EN * 7D7<
A <31..0>       12C3<> 16A8<> 7A8<
AACK *          7C1<> 12D3<
ABB *           12D6
ABUF <28..12>   16D1> 17A8< 17B8<
AOS_RR_MMRS     7C1> 8A5<
ARTRY *         7C1<> 12D3>
AVDD            11A4
BCLK_DIV0       15C7
BCLK_DIV1       15C7
BG_60X *        7D1> 12D3<
BG_MASTER *     7D1>
BR_60X *        12D3> 7D7<
BR_MASTER *     7D7<
C2P_WRL_OPEN    8B6
CKSTP_OUT *     14D5
CPU_DATA_OE *   8B6
CPU_RDL_OPEN    8B6
CRS_C2PWXS      8B6
D <63..0>       10A8<> 13D1<> 17D1<>
DBB *           13D5
DBDIS *         12D6
DBG *           7C1> 13D1<
DD <15..0>      22C2<>
DP <7..0>       10A8<> 13B1<> 17D1<>
DPE_60X *       14D1> 7C7<
DRTRY *         13D8> 14A8>
DUAL_CTRL_REF   8B6
ECC_LE_SEL      8B6
ENCDCS0         20C1>
ENCDCS1         20C1>
ENCDCS2         20C1>
ENCDCSEN *      20C1>
FLASH_WE_A18    23C5
FRD *           21C1> 22C7< 23A7<
FWR *           21C1> 23A7<
GBL *           7C1> 12A3<>
HDWR_RESET *    14C6
HFENH *         21C1> 22B7<
HFENL *         21C1> 22C7<
HRESET *        14C8> 14A8<
IGN_PCI_AD31    18A2> 9D2<
INT_60X *       7D1> 14D8<
INT_TO_664      20B1> 9D2<
IRQ1            20B8<
IRQ3            20B8<
IRQ4            20B8<
IRQ5            20B8<
IRQ6            20B8<
IRQ7            20B8<
IRQ8 *          20B8<
IRQ9            20B8<
```

```
IRQ10           20B8<
IRQ11           20B8<
IRQ12           20B8<
IRQ13           20B8<
IRQ14           20C8<
IRQ15           20C8<
ISA_0WS *       19C8<
ISA_AEN         19D2>
ISA_BALE        19D2>
ISA_CLK         19A2>
ISA_DACK0 *     19D2>
ISA_DACK1 *     19D2>
ISA_DACK2 *     19D2>
ISA_DACK3 *     19D2>
ISA_DACK5 *     19D2>
ISA_DACK6 *     19D2>
ISA_DACK7 *     19D2>
ISA_DRQ0        19C8<
ISA_DRQ1        19C8<
ISA_DRQ2        19C8<
ISA_DRQ3        19C8<
ISA_DRQ5        19C8<
ISA_DRQ6        19C8<
ISA_DRQ7        19C8<
ISA_IOCHCK *    19C8<
ISA_IOCHRDY     19D2<>
ISA_IOCS16 *    19C8<
ISA_IOR *       19D2<> 22A7<
ISA_IOW *       19D2<> 22A7<
ISA_LA <23..17> 19D2<> 23B7<
ISA_MASTER *    19C8<
ISA_MEMCS16 *   19C2<>
ISA_MEMR *      19C2<>
ISA_MEMW *      19C2<>
ISA_REFRESH *   19A2>
ISA_RESET       19C2>
ISA_RESET *     19C2>
ISA_SBHE *      19A2<>
ISA_SMEMR *     19A2>
ISA_SMEMW *     19A2>
ISA_TC          19D2>
L2_CLAIM *      7B7<
MA <11..0>      8D4> 5B7< 6B7<
MCE <7..0> *    8D4> 5D7< 6C7<
MCP_60X *       7D1> 14C1<
MD <63..0>      4D4<> 6A7<> 10D8<>
MDP <7..0>      4D4<> 6A7<> 10D8<>
MEM_BE <3..0>   8B6 8C6
MEM_DATA_OE *   8B6
MEM_RD_SMPL     8B6
MEM_WRL_OPEN    8B6
MPD <23..0>     5A2> 6D2>
MRE <3..0> *    8D4> 5C7< 5D7< 6D7<
MWE0 *          8D4> 5A7< 6C7<
MWE1 *          8D4> 5A7< 6D7<
MWS_P2MRXS      8B6
NMI_FROM_ISABRDG 20B1> 9D2<
PCI_AD <31..0>  9A8<> 18D2<> 23B7<> 18C7<
PCI_AD_OE *     8B6
PCI_C/BE <3..0> * 9B2<> 18B2<>
PCI_DEVSEL *    9C2<> 18B2<>
PCI_DIV0        15B7
PCI_DIV1        15C7
PCI_EXT_SEL     8B6
PCI_FRAME *     9C2<> 18B2<>
PCI_GNT0 *      18B2>
```

```
PCI_GNT1 *      18B2>
PCI_GNT2 *      18B2>
PCI_GNT3 *      18B2>
PCI_GNT4 *      21D1> 18A4<
PCI_IRDY *      9C2<> 18B2<> 8A5<
PCI_ISABRDG_CLK 15A1> 18C7<
PCI_LOCK *      9D2< 18C7<
PCI_OL_OPEN     8B6
PCI_OUT_SEL     8B6
PCI_PAR         9A8<> 18C2>
PCI_PERR *      9B2<> 21D1>
PCI_REQ0 *      18B7<
PCI_REQ1 *      18B7<
PCI_SERR *      9B2<> 18C7<
PCI_STOP *      9C2<> 18B2<>
PCI_TRDY *      9C2<> 18B2<> 8A5<
PCLK_60X        15C1> 14C8<
PLL_CFG <3..0>  15A8> 14B1<
POWER_GOOD/RESET * 7D7< 14D8< 18C7< 20C1<
RESET_INTERRUPT * 14D6
ROM_CE *        23A7<
ROM_LOAD        7C1> 8A5<
ROM_OE *        7C1> 23C7<
ROM_WE *        7C1> 23C7<
SA <19..0>      19A2<> 22A7< 23A7<
SD <15..0>      19A2<> 22C7<>
SHD *           7C1> 12D3<>
SMI *           14C1<
SPKR            20B1>
SRAM_ADS/ADDR0  7B1> 17B8< 17C8<
SRAM_ALE        7B1> 16D8<
SRAM_BCLK0      15C1> 17D8<
SRAM_BCLK1      15C1> 17D8<
SRAM_CNT_EN/ADDR1 7B1> 17B8<
SRAM_OE *       7B1> 17C8<
SRAM_PD <2..0>  17B8>
SRAM_WE *       7B1> 17C8<
SRESET          20B3
SRESET *        14D5
SRESET_60X *    20B1> 14D8<
TA *            7C1<> 13D1<
TAG_BCLK        15D1> 16D8<
TAG_CLEAR *     7B1> 16B8<
TAG_MATCH       16A1> 7B7<
TAG_VALID       7B1> 16C8<
TAG_WE *        7B1> 16A8<
TBST *          7C7<> 12A3>
TCK             14B7
TDI             14B7
TDO             14B7
TEA *           7D1> 13D8<
TEST_DIS *      14A7
TMS             14B7
TRST *          14C5
TS *            7C7<> 12C3<>
TSIZ <2..0>     7C7<> 12A3>
TT <4..0>       7C7<> 12B3<>
UNUSED_2XPCLK   15D2>
UNUSED_BCLK0    15B2>
UNUSED_PCICLK2  15B2>
UNUSED_PCICLK3  15B2>
UNUSED_PCICLK4  15B2>
UNUSED_PCICLK5  15B2>
UNUSED_PCICLK6  15B2>
XA <9..0>       22A2>
XATS *          12C3<> 7C7<
```

```
XD <7..0>       22D7<> 23A7<>
XDEN *          20B1> 22D7<
XDIR            20B1> 22D7<
XIOR *          22A2>
XIOW *          22A2>
XTAL1           15D6
XTAL2           15C6
```

IBM			
	SIZE B	REVISION	DRWNG PART NUMBER
			SHEET 24 OF

PowerPC™

IBM27-82660 Revision 2.1 Product Update

MPR660ESU-04

This document describes improvements to the IBM27-82660 Bridge from revision 1.0 to revision 2.0, and corrects an error in the revision 2.0 Product Update.

October 30, 1995

IBM®

Document Revision History

Errata Summary release 1.0 (April 19, 1995) [MPR660ESU-00]
Errata for the 1.0 revision of the chipset.

Errata Summary release 1.2 (April 19, 1995) [MPR660ESU-03]
Errata for IBM27–82664 revision 1.1 and IBM27–82663 revisions 1.0 and 2.0.

IBM27-82660 Revision 2.0 Product Update (June 5, 1995) [MPR660ESU-01]
Information on improvements to the chipset design from revision 1.0 to revision 2.0.

IBM27-82660 Revision 2.1 Product Update (October 30, 1995) [MPR660ESU-04]
Corrected information on improvements to the chipset design from revision 1.0 to revision 2.0.

Part Identification, IBM27-82660 Chipset

IBM27-82663 (663 Buffer)
Chipset Rev. 1.0 contains IBM27-82663 Rev. 1.0. The package is marked 94G0235.

Chipset Rev. 2.0 contains IBM27-82663 Rev. 2.0. The package is marked 94G0178.

IBM27-82664 (664 Controller)
Chipset Rev. 1.0 contains IBM27-82664 Rev. 1.0. The package is marked 94G0232.

Chipset Rev. 2.0 contains IBM27-82664 Rev. 1.1. The package is marked 94G0176.

IBM27-82660 Revision 2.1 Product Update

October 30, 1995

This document describes improvements to the IBM27-82660 Bridge from revision 1.0 to revision 2.0, and corrects an error in the revision 2.0 Product Update.

Revision 1.0 of the 660 Bridge is described by a combination of three documents:

- *The IBM 27-82660 PowerPC to PCI Bridge User's Manual*, MPR660UMU-01 (beta version)
- *A System Implementation Example for the IBM 82660 Chipset (Preliminary)* schematic set, attached to the User's Manual
- The *IBM27-82660 Revision 1.0 Errata Summary*, MPR660ESU-00, or the most current 1.x revision.

Between revisions 1.0 and 2.0 of the 660 Bridge, several changes were implemented:

1. All known revision 1.0 errata were designed out. All external logic workarounds that were required to fix revision 1.0 errata were eliminated.
2. Two signal connections were added to the chipset. DBG# was added to the 664 Controller. PCI_TRDY# was added to the 663 Buffer.
3. The entire pinout of the 663 Buffer was rearranged.
4. The example implementation of the 663 Buffer was changed to enhance signal to noise ratio.

Revision 2.0 of the 660 Bridge is also described by a combination of four documents:

1. *The IBM 27-82660 PowerPC to PCI Bridge User's Manual*, MPR660UMU-01 (beta version)
2. The *IBM27-82660 Example Implementation For MPR660UMU–02* schematic set, attached to the User's Manual
3. This document, the *IBM27–82660 Revision 2.1 Product Update*
4. The most current version of the revision 2.x Errata Summary.

1 All Known Revision 1.0 Errata Designed Out

All external logic that was required as a workaround for the revision 1.0 chipset has been eliminated. Except as noted herein, all of the workarounds should be removed from the design, and the 660 Bridge incorporated in the design as shown in the *IBM27-82660 Example Implementation For MPR660UMU–02* schematic set.

2 Signals Added to the Chipset

DBG# was added to the 664 Controller to support a 604 in fast L2 mode. This change is documented in the MPR660UMU-01 release of the 660 Bridge User's Manual, and is noted here for completeness. See the 660 User's Manual for more information.

PCI_TRDY# was connected to the 663 Buffer to improve performance.

3 663 Buffer Pinout Rearranged

The pinout of the 663 Buffer was rearranged to reduce ground bounce effects by distributing simultaneously switched signals more evenly around the perimeter of the device. Previously, signal groups that were logically grouped were also grouped together physically. Redistributing these I/O more evenly has the effect of more evenly distributing the instantaneous current drawn from the power pins. This more even distribution of the current draw also evens out the instantaneous voltage drop experienced by the device during simultaneous switching events.

Note that the previous version of this document (revision 2.0) incorrectly showed the MEM_CHECK[0:7] pins in reversed order.

3.1 New 663 Buffer Numeric Pin List

Pin	663 Signal Name	Pin	663 Signal Name	Pin	663 Signal Name
1	CPU_DATA[22]	50	PCI_AD[26]	99	CPU_DATA[49]
2	CPU_DATA[23]	51	PCI_AD[27]	100	MEM_DATA[45]
3	CPU_DPAR[2]	52	V_{DD}	101	MEM_DATA[46]
4	CPU_DATA[24]	53	GND	102	MEM_DATA[47]
5	MEM_DATA[24]	54	CPU_DATA[32]	103	MEM_CHECK[2]
6	MEM_DATA[25]	55	CPU_DATA[33]	104	CPU_DATA[50]
7	MEM_DATA[26]	56	CPU_DATA[34]	105	CPU_DATA[51]
8	V_{DD}	57	CPU_DATA[35]	106	CPU_DATA[52]
9	GND	58	MEM_DATA[32]	107	CPU_DATA[53]
10	PCI_AD[04]	59	MEM_DATA[33]	108	MEM_DATA[48]
11	PCI_AD[05]	60	MEM_DATA[34]	109	MEM_DATA[49]
12	PCI_AD[06]	61	V_{DD}	110	V_{DD}
13	PCI_AD[07]	62	PCI_AD[28]	111	MEM_DATA[50]
14	PCI_AD[08]	63	PCI_AD[29]	112	MEM_DATA[51]
15	PCI_AD[09]	64	PCI_AD[30]	113	MEM_DATA[52]
16	PCI_AD[10]	65	PCI_AD[31]	114	CPU_DATA[54]
17	PCI_AD[11]	66	GND	115	CPU_DATA[55]
18	PCI_AD[12]	67	CPU_DATA[36]	116	CPU_DPAR[6]
19	PCI_AD[13]	68	CPU_DATA[37]	117	CPU_DATA[56]
20	PCI_AD[14]	69	CPU_DATA[38]	118	MEM_DATA[53]
21	PCI_AD[15]	70	CPU_DATA[39]	119	MEM_DATA[54]
22	V_{DD}	71	GND	120	GND
23	GND	72	V_{DD}	121	MEM_DATA[55]
24	CPU_DATA[25]	73	MEM_DATA[35]	122	MEM_CHECK[1]
25	CPU_DATA[26]	74	MEM_DATA[36]	123	MEM_DATA[56]
26	CPU_DATA[27]	75	MEM_DATA[37]	124	CPU_DATA[57]
27	CPU_DATA[28]	76	MEM_DATA[38]	125	CPU_DATA[58]
28	MEM_DATA[27]	77	CPU_DPAR[4]	126	CPU_DATA[59]
29	MEM_DATA[28]	78	CPU_DATA[40]	127	CPU_DATA[60]
30	MEM_DATA[29]	79	CPU_DATA[41]	128	V_{DD}
31	CPU_DATA[29]	80	CPU_DATA[42]	129	GND
32	CPU_DATA[30]	81	MEM_DATA[39]	130	MEM_DATA[57]
33	CPU_DATA[31]	82	MEM_CHECK[3]	131	MEM_DATA[58]
34	CPU_DPAR[3]	83	MEM_DATA[40]	132	MEM_DATA[59]
35	MEM_DATA[30]	84	V_{DD}	133	MEM_DATA[60]
36	MEM_DATA[31]	85	GND	134	CPU_DATA[61]
37	MEM_CHECK[4]	86	CPU_DATA[43]	135	CPU_DATA[62]
38	V_{DD}	87	CPU_DATA[44]	136	CPU_DATA[63]
39	GND	88	CPU_DATA[45]	137	CPU_DPAR[7]
40	PCI_AD[16]	89	CPU_DATA[46]	138	MEM_DATA[61]
41	PCI_AD[17]	90	MEM_DATA[41]	139	MEM_DATA[62]
42	PCI_AD[18]	91	MEM_DATA[42]	140	MEM_DATA[63]
43	PCI_AD[19]	92	MEM_DATA[43]	141	MEM_CHECK[0]
44	PCI_AD[20]	93	MEM_DATA[44]	142	V_{DD}
45	PCI_AD[21]	94	CPU_DATA[47]	143	GND
46	PCI_AD[22]	95	CPU_DPAR[5]	144	PCI_AD_OE#
47	PCI_AD[23]	96	V_{DD}	145	MEM_DATA_OE#
48	PCI_AD[24]	97	GND	146	CPU_DATA_OE#
49	PCI_AD[25]	98	CPU_DATA[48]	147	MEM_RD_SMPL

New 663 Buffer Numeric Pin List (Continued)

Pin	663 Signal Name
148	CPU_RDL_OPEN
149	ECC_LE_SEL
150	MEM_WRL_OPEN
151	CRS_C2PWXS
152	MWS_P2MRXS
153	PCI_EXT_SEL
154	C2P_WRL_OPEN
155	TEST#
156	MIO_TEST
157	CPU_CLK
158	V_{DD}
159	GND
160	ROM_LOAD
161	MEM_BE[0]
162	MEM_BE[1]
163	MEM_BE[2]
164	MEM_BE[3]
165	PCI_OL_OPEN
166	AOS_RR_MMRS
167	PCI_IRDY#
168	PCI_TRDY#
169	PCI_OUT_SEL
170	DUAL_CTRL_REF
171	MEM_ERR#
172	V_{DD}
173	GND
174	CPU_PAR_ERR#
175	SBE#
176	CPU_DATA[00]
177	CPU_DATA[01]
178	CPU_DATA[02]

Pin	663 Signal Name
179	CPU_DATA[03]
180	MEM_DATA[00]
181	V_{DD}
182	MEM_DATA[01]
183	MEM_DATA[02]
184	MEM_DATA[03]
185	CPU_DATA[04]
186	CPU_DATA[05]
187	CPU_DATA[06]
188	CPU_DATA[07]
189	MEM_DATA[04]
190	MEM_DATA[05]
191	GND
192	V_{DD}
193	MEM_DATA[06]
194	MEM_DATA[07]
195	MEM_CHECK[7]
196	CPU_DPAR[0]
197	CPU_DATA[08]
198	CPU_DATA[09]
199	CPU_DATA[10]
200	MEM_DATA[08]
201	MEM_DATA[09]
202	MEM_DATA[10]
203	MEM_DATA[11]
204	V_{DD}
205	GND
206	MEM_DATA[12]
207	CPU_DATA[11]
208	CPU_DATA[12]
209	CPU_DATA[13]

Pin	663 Signal Name
210	CPU_DATA[14]
211	MEM_DATA[13]
212	MEM_DATA[14]
213	MEM_DATA[15]
214	MEM_CHECK[6]
215	MEM_DATA[16]
216	V_{DD}
217	GND
218	CPU_DATA[15]
219	CPU_DPAR[1]
220	CPU_DATA[16]
221	CPU_DATA[17]
222	MEM_DATA[17]
223	MEM_DATA[18]
224	MEM_DATA[19]
225	MEM_DATA[20]
226	CPU_DATA[18]
227	CPU_DATA[19]
228	CPU_DATA[20]
229	CPU_DATA[21]
230	V_{DD}
231	MEM_DATA[21]
232	MEM_DATA[22]
233	MEM_DATA[23]
234	MEM_CHECK[5]
235	GND
236	PCI_AD[00]
237	PCI_AD[01]
238	PCI_AD[02]
239	PCI_AD[03]
240	GND

3.2 New 663 Buffer Alphabetic Pin List

Pin	663 Signal Name
166	AOS_RR_MMRS
154	C2P_WRL_OPEN
157	CPU_CLK
176	CPU_DATA[00]
177	CPU_DATA[01]
178	CPU_DATA[02]
179	CPU_DATA[03]
185	CPU_DATA[04]
186	CPU_DATA[05]
187	CPU_DATA[06]
188	CPU_DATA[07]
197	CPU_DATA[08]
198	CPU_DATA[09]
199	CPU_DATA[10]
207	CPU_DATA[11]
208	CPU_DATA[12]
209	CPU_DATA[13]
210	CPU_DATA[14]
218	CPU_DATA[15]
220	CPU_DATA[16]
221	CPU_DATA[17]
226	CPU_DATA[18]
227	CPU_DATA[19]
228	CPU_DATA[20]
229	CPU_DATA[21]
1	CPU_DATA[22]
2	CPU_DATA[23]
4	CPU_DATA[24]
24	CPU_DATA[25]
25	CPU_DATA[26]
26	CPU_DATA[27]
27	CPU_DATA[28]
31	CPU_DATA[29]
32	CPU_DATA[30]
33	CPU_DATA[31]
54	CPU_DATA[32]
55	CPU_DATA[33]
56	CPU_DATA[34]
57	CPU_DATA[35]
67	CPU_DATA[36]
68	CPU_DATA[37]
69	CPU_DATA[38]
70	CPU_DATA[39]
78	CPU_DATA[40]
79	CPU_DATA[41]
80	CPU_DATA[42]
86	CPU_DATA[43]
87	CPU_DATA[44]

Pin	663 Signal Name
88	CPU_DATA[45]
89	CPU_DATA[46]
94	CPU_DATA[47]
98	CPU_DATA[48]
99	CPU_DATA[49]
104	CPU_DATA[50]
105	CPU_DATA[51]
106	CPU_DATA[52]
107	CPU_DATA[53]
114	CPU_DATA[54]
115	CPU_DATA[55]
117	CPU_DATA[56]
124	CPU_DATA[57]
125	CPU_DATA[58]
126	CPU_DATA[59]
127	CPU_DATA[60]
134	CPU_DATA[61]
135	CPU_DATA[62]
136	CPU_DATA[63]
146	CPU_DATA_OE#
196	CPU_DPAR[0]
219	CPU_DPAR[1]
3	CPU_DPAR[2]
34	CPU_DPAR[3]
77	CPU_DPAR[4]
95	CPU_DPAR[5]
116	CPU_DPAR[6]
137	CPU_DPAR[7]
174	CPU_PAR_ERR#
148	CPU_RDL_OPEN
151	CRS_C2PWXS
170	DUAL_CTRL_REF
149	ECC_LE_SEL
9	GND
23	GND
39	GND
53	GND
66	GND
71	GND
85	GND
97	GND
120	GND
129	GND
143	GND
159	GND
173	GND
191	GND
205	GND

Pin	663 Signal Name
217	GND
235	GND
240	GND
161	MEM_BE[0]
162	MEM_BE[1]
163	MEM_BE[2]
164	MEM_BE[3]
141	MEM_CHECK[0]
122	MEM_CHECK[1]
103	MEM_CHECK[2]
82	MEM_CHECK[3]
37	MEM_CHECK[4]
234	MEM_CHECK[5]
214	MEM_CHECK[6]
195	MEM_CHECK[7]
180	MEM_DATA[00]
182	MEM_DATA[01]
183	MEM_DATA[02]
184	MEM_DATA[03]
189	MEM_DATA[04]
190	MEM_DATA[05]
193	MEM_DATA[06]
194	MEM_DATA[07]
200	MEM_DATA[08]
201	MEM_DATA[09]
202	MEM_DATA[10]
203	MEM_DATA[11]
206	MEM_DATA[12]
211	MEM_DATA[13]
212	MEM_DATA[14]
213	MEM_DATA[15]
215	MEM_DATA[16]
222	MEM_DATA[17]
223	MEM_DATA[18]
224	MEM_DATA[19]
225	MEM_DATA[20]
231	MEM_DATA[21]
232	MEM_DATA[22]
233	MEM_DATA[23]
5	MEM_DATA[24]
6	MEM_DATA[25]
7	MEM_DATA[26]
28	MEM_DATA[27]
29	MEM_DATA[28]
30	MEM_DATA[29]
35	MEM_DATA[30]
36	MEM_DATA[31]
58	MEM_DATA[32]

New 663 Buffer Alphabetic Pin List (Continued)

Pin	663 Signal Name
59	MEM_DATA[33]
60	MEM_DATA[34]
73	MEM_DATA[35]
74	MEM_DATA[36]
75	MEM_DATA[37]
76	MEM_DATA[38]
81	MEM_DATA[39]
83	MEM_DATA[40]
90	MEM_DATA[41]
91	MEM_DATA[42]
92	MEM_DATA[43]
93	MEM_DATA[44]
100	MEM_DATA[45]
101	MEM_DATA[46]
102	MEM_DATA[47]
108	MEM_DATA[48]
109	MEM_DATA[49]
111	MEM_DATA[50]
112	MEM_DATA[51]
113	MEM_DATA[52]
118	MEM_DATA[53]
119	MEM_DATA[54]
121	MEM_DATA[55]
123	MEM_DATA[56]
130	MEM_DATA[57]
131	MEM_DATA[58]
132	MEM_DATA[59]
133	MEM_DATA[60]
138	MEM_DATA[61]
139	MEM_DATA[62]
140	MEM_DATA[63]
145	MEM_DATA_OE#

Pin	663 Signal Name
171	MEM_ERR#
147	MEM_RD_SMPL
150	MEM_WRL_OPEN
156	MIO_TEST
152	MWS_P2MRXS
236	PCI_AD[00]
237	PCI_AD[01]
238	PCI_AD[02]
239	PCI_AD[03]
10	PCI_AD[04]
11	PCI_AD[05]
12	PCI_AD[06]
13	PCI_AD[07]
14	PCI_AD[08]
15	PCI_AD[09]
16	PCI_AD[10]
17	PCI_AD[11]
18	PCI_AD[12]
19	PCI_AD[13]
20	PCI_AD[14]
21	PCI_AD[15]
40	PCI_AD[16]
41	PCI_AD[17]
42	PCI_AD[18]
43	PCI_AD[19]
44	PCI_AD[20]
45	PCI_AD[21]
46	PCI_AD[22]
47	PCI_AD[23]
48	PCI_AD[24]
49	PCI_AD[25]
50	PCI_AD[26]

Pin	663 Signal Name
51	PCI_AD[27]
62	PCI_AD[28]
63	PCI_AD[29]
64	PCI_AD[30]
65	PCI_AD[31]
144	PCI_AD_OE#
153	PCI_EXT_SEL
167	PCI_IRDY#
165	PCI_OL_OPEN
169	PCI_OUT_SEL
168	PCI_TRDY#
160	ROM_LOAD
175	SBE#
155	TEST#
8	V_{DD}
22	V_{DD}
38	V_{DD}
52	V_{DD}
61	V_{DD}
72	V_{DD}
84	V_{DD}
96	V_{DD}
110	V_{DD}
128	V_{DD}
142	V_{DD}
158	V_{DD}
172	V_{DD}
181	V_{DD}
192	V_{DD}
204	V_{DD}
216	V_{DD}
230	V_{DD}

4 Example Implementation Change

To improve output signal quality, 33Ω series resistors have been added to the 663 buffer connections to the CPU data bus lines CPU_DATA[0:63] and CPU_DPAR[0:8]. Each resister is placed between the 663 Buffer CPU_DATA[n] pin and all other CPU bus agent(s) attached to the CPU_DATA[n] net. In the case of the CPU_DPAR[n] nets, the resistor is placed between the 663 Buffer CPU_DPAR[n] pin and all other CPU bus agent(s) attached to the CPU_DPAR[n] net. For maximum benefit, place the series resistors as close to the 663 Buffer package as is feasible. These resistors and the new pinout of the 663 Buffer are shown on the *IBM27-82660 Example Implementation For MPR660UMU–02* schematic set.

5 Additional Resources

The *IBM PowerPC 603/604 Reference Design* revision 2.1 and the *IBM PowerPC 604 SMP Reference Design* are now available to designers from your IBM technical representative. These designs feature the 660 Bridge, and include comprehensive and extensive documentation of system-related issues.

PowerPC

USA and Canada:
IBM Microelectronics Division
1580 Route 52, Bldg. 504
Hopewell Junction, NY 12533-6531
Tel: (800) PowerPC
Fax: (800) PowerFax

Europe:
IBM
La Pompignane BP 1021
34006 Montpellier, France
Tel: (33) 6713–5757 (Français)
(33) 6713–5756 (Italiano)
Fax: (33) 6713–5750
(from Paris add 16)

Europe:
IBM
Informations Systeme GmbH
Laatzener Str. 1
30539 Hannover, Germany
Tel: (49) 511–516–3444 (English)
(49) 511–516–3555 (Deutsche)
Fax: (49) 511–516–3888

Japan:
IBM
800, Ichimiyake
Yasu–cho, Yasu–gun
Shiga–ken, Japan 520–23
Tel: (81) 775–87–4745
Fax: (81) 775–87–4735

Document Number: MPR660ESU–04