APPLICATION NOTE

AN-96

Data Endianess and Data Swapping of Marvell®'s GT-64260A Device

Introduction

The purpose of this document is to explain the data Endianess between the device interfaces. It also helps the user of the GT-64260A (Marvell device) to gain a better understanding of how the Marvell device handles agents on the PCI bus with different Endianess.

Description of Endianess

CPU Endianess is the term used to describe how the CPU sets the byte order in a single native word. A single native word is the CPU bus size, e.g. for a 32-bit CPU a word is four bytes. Although the choice of byte ordering is arbitrary, there are only two orderings that are practical: Big Endian and Little Endian.

Big Endian

Big-Endian byte ordering is described as follows: the most-significant byte (MSB) of data is stored at the lowest address while the least-significant byte (LSB) of data is stored at the highest address.

Little Endian

Little-Endian byte ordering is described as follows: the least-significant byte (LSB) of data is stored at the lowest address while the most-significant byte (MSB) of data is stored at the highest address.

Regardless of the data Endianess you are dealing with, the bit order inside a byte or multi-byte structure is always in Little-Endian ordering. That is, the least significant bit (LSb) is always the rightmost bit. The most significant bit (MSb) is always the leftmost bit.

Tabulated Bit and Byte Orientation

Table 1 and Table 2 show the data bit and data byte orientation in Big and Little Endian format.

Table 1: Big Endian Bit and Byte Orientation

Data Bit	07...00	15...08	23...16	31...24	39...32	47...40	55...48	63...56
Address	Highest							Lowest
Data Bytes	0	1	2	3	4	5	6	7
	LSB							MSB

Table 2: Little Endian Bit and Byte Orientation

Data Bit	**63...56**	**55...48**	**47...40**	**39...32**	**31...24**	**23...16**	**15...08**	**07...00**
Address	Highest							Lowest
Data Bytes	7	6	5	4	3	2	1	0
	MSB							LSB

Table 3 and Table 4 show the data bit and byte orientation in the Motorola PowerPC CPU and PCI data buses. Although the bit numbering convention is different, the significance of each bit is the same for both the CPU and the PCI buses.

Table 3: CPU Data Bus Bit Orientation

DH[0:31]	DL[0:31]
MSb	LSb
0 1 2 3 . . . . 27 28 29 30 31	0 1 2 3 . . . . 27 28 29 30 31

Table 4: PCI Data Bus Bit Orientation

AD[63:32]	AD[31:0]
MSb	LSb
63 62 61 60 . . . . 35 34 33 32	31 30 29 28 . . . . 3 2 1 0

GT-64260A Endianess

The Marvell device only supports a Big-Endian CPU data bus. (The Endianess bit (bit 12) in the CPU Configuration Register must be set to '0'.) The Marvell device provides the capability to swap the byte order of data that enables Endianess conversion between the CPU and other interfaces.

Memory and Device Interface

The Endianess convention of the local memory attached to the Marvell device, such as the SDRAM and device bus, is assumed to be the same as the CPU. This means data transferred to/from the local memory or device data bus is never swapped.

Figure 5, Figure 6, and Figure 7 show what happens when the CPU writes a double word to 8- 16- and 32 bit devices. The blue shows what was written and the red shows how it appears on the device bus.

Table 5: CPU Writes a Double Word (64-bit) to 8-bit Device

PPC CPU Address	PPC CPU Data Bus		Device Bus	
A[0:31]	DH[0:31]	DL[0:31]	BAdr[2:0]	AD[31:0]
0x00	0x01020304	0x05060708	0x00	0x01010101
			0x01	0x02020202
			0x02	0x03030303
			0x03	0x04040404
			0x04	0x05050505
			0x05	0x06060606
			0x06	0x07070707
			0x07	0x08080808

Table 6: CPU Writes a Double Word (64-bit) to 16-bit Device

PPC CPU Address	PPC CPU Data Bus		Device Bus	
A[0:31]	DH[0:31]	DL[0:31]	BAdr[2:0]	AD[31:0]
0x00	0x01020304	0x05060708	0x00	0x01010102
			0x01	0x02020304
			0x02	0x03030506
			0x03	0x04040708

Table 7: CPU Writes a Double Word (64-bit) to 32-bit Device

PPC CPU Address	PPC CPU Data Bus		Device Bus	
A[0:31]	DH[0:31]	DL[0:31]	BAdr[2:0]	AD[31:0]
0x00	0x01020304	0x05060708	0x00	0x01020304
			0x01	0x05060708

Internal Register

The internal registers of the Marvell device are always programmed in Little-Endian ordering. On a CPU access to the internal registers, data is swapped.

PCI Interface

Data swapping on a CPU access to an agent on the PCI bus is controlled via the PCI Swap bits of each PCI Low Address register. This configurable setting allows a CPU access to PCI agents using a different Endianess convention. If the PCI Command register's MSwapEn bit (bit 21) is set to '1', the Marvell device PCI master performs data swapping according to the PCI Swap bits setting. If set to '0' (default), it works according to the MByteSwap (bit 0) and MWordSwap (bit 10) settings in the PCI Command register.

Although the PCI specification defines the PCI data bus only as a Little-Endian bus, the Marvell device also includes support for interfacing with Big-Endian PCI devices. Endianess conversion is supported in both directions: access to the PCI agent via the PCI master interface and access from the PCI agent via the PCI slave interface. Both the PCI master and slave support byte and word swapping. The swapping is referred to as a 64-bit word (as this is the Marvell device's internal data path width). Table 8 shows an example of the data 0x0011223344556677 being swapped using the combination of the word and byte swap control bits.

Table 8: Data Swap Control

Swap Control [Word, Byte]	Swapping Granularity	Swapped Data							
00	Byte	77	66	55	44	33	22	11	00
01	No Swapping	00	11	22	33	44	55	66	77
10	Byte and Word	33	22	11	00	77	66	55	44
11	Word	44	55	66	77	00	11	22	33

The correct swap setting depends on the PCI agent data bus width (32/64) and its Endianess (Big/Little), as well as the CPU data bus' Endianess. Since the Marvell device only supports a Big Endian CPU data bus, a Little Endian CPU data bus is not listed in the tables below. Table 9 and Table 10 show the correct swap settings for the different PCI agent data bus widths and Endianess orientations.

Table 9: 32-bit PCI Byte and Word Swap Control Bit Settings

	Little-Endian PCI Agent	Big-Endian PCI Agent
Big-Endian CPU data bus	Byte swapping	Word swapping

Table 10: 64-bit PCI Byte and Word Swap Control Bit Settings

	Little-Endian PCI Agent	Big-Endian PCI Agent
Big-Endian CPU data bus	Byte swapping	No swapping

PCI Slave Data Swapping

For maximum Endianess flexibility, you can configure each of the eight address ranges defined by the PCI Access Control registers for different data swapping. This feature enables different PCI masters with different Endianess conventions to interface with the Marvell device. If the PCI Command register’s SwapEn bit is cleared (default), the PCI slave handles data according to the settings of the PCI Command register’s SByteSwap [16] and SWordSwap bit [11]. The Marvell device’s internal registers always maintain Little-Endian data. By default, it is assumed that data driven on the PCI bus is in Little-Endian mode, and there is no data swapping on PCI access to the internal registers. However, the Marvell device also includes support for data swapping on the PCI access to internal registers via the PCI Command register’s SIntSwap bits [26:24].

PCI Master Data Swapping

Very similar to the slave data swapping mechanism, the PCI master also supports data swapping on any access to the PCI bus. It also supports flexible swapping control, determined by the initiator, on an address window basis. This feature enables the CPU, IDMAs, and Communication units to interface different PCI targets with different Endianess conventions. If the PCI Command register’s SwapEn bit is cleared (default), the PCI master handles data according to the setting of the PCI Command register’s MByteSwap bit [0] and MWordSwap bit [10].

Tabulated Data on the CPU and PCI Bus

The tables in this section show what the PCI bus looks like after the CPU writes data to a PCI agent, depending on the Endianess conversion, bus width, and swap settings used. Table 11lists the tables in this section and the settings that are illustrated in them.

Table 11: PCI Bus Tables

CPU Write	PCI Agent	Swap Type	Table
1 byte	32-bit	None	Table 12 on page 6
1 byte	32-bit	Byte	Table 13 on page 6
1 byte	32-bit	Word	Table 14 on page 7
1 byte	32-bit	Byte and Word	Table 15 on page 7
4 bytes	32-bit	None	Table 16 on page 8
4 bytes	32-bit	Byte	Table 17 on page 8
4 bytes	32-bit	Word	Table 18 on page 8
4 bytes	32-bit	Byte and Word	Table 19 on page 8
4 bytes	64-bit	None	Table 20 on page 8
4 bytes	64-bit	Byte	Table 21 on page 8
4 bytes	64-bit	Word	Table 22 on page 9
4 bytes	64-bit	Byte and Word	Table 23 on page 9

Table 11: PCI Bus Tables (Continued)

CPU Write	PCI Agent	Swap Type	Table
8 bytes	32-bit	None	Table 24 on page 9
8 bytes	32-bit	Byte	Table 25 on page 9
8 bytes	32-bit	Word	Table 26 on page 9
8 bytes	32-bit	Byte and Word	Table 27 on page 9
8 bytes	64-bit	None	Table 28 on page 10
8 bytes	64-bit	Byte	Table 29 on page 10
8 bytes	64-bit	Word	Table 30 on page 10
8 bytes	64-bit	Byte and Word	Table 31 on page 10

Table 12: 1-Byte CPU Write to a 32-bit PCI Agent with No Swapping

	PPC CPU Bus		PCI Bus	
Address Offset	DH[0:31]	DL[0:31]	AD[63:32]	AD[31:0]
0x00	0xAAXXXXXX	0xXXXXXXXX	N/A	0xAAXXXXXX
0x01	0xXXAAXXXX	0xXXXXXXXX	N/A	0xXXAAXXXX
0x02	0xXXXXAAXX	0xXXXXXXXX	N/A	0xXXXXAAXX
0x03	0xXXXXXXAA	0XXXXXXXXX	N/A	0xXXXXXXAA
0x04	0XXXXXXXXX	0XAAXXXXXX	N/A	0xAAXXXXXX
0x05	0xXXXXXXXX	0xXXAAXXXX	N/A	0xXXAAXXXX
0x06	0xXXXXXXXX	0xXXXXAAXX	N/A	0xXXXXAAXX
0x07	0xXXXXXXXX	0xXXXXXXAA	N/A	0xXXXXXXAA

Table 13: 1-Byte CPU Write to a 32-bit PCI Agent with Byte Swapping Only

	PPC CPU Bus		PCI Bus	
Address Offset	DH[0:31]	DL[0:31]	AD[63:32]	AD[31:0]
0x00	0xAAXXXXXX	0xXXXXXXXX	N/A	0xXXXXXXAA
0x01	0xXXAAXXXX	0xXXXXXXXX	N/A	0xXXXXAAXX
0x02	0xXXXXAAXX	0xXXXXXXXX	N/A	0xXXAAXXXX
0x03	0xXXXXXXAA	0XXXXXXXXX	N/A	0xAAXXXXXX
0x04	0XXXXXXXXX	0XAAXXXXXX	N/A	0xXXXXXXAA
0x05	0xXXXXXXXX	0xXXAAXXXX	N/A	0xXXXXAAXX

Table 13: 1-Byte CPU Write to a 32-bit PCI Agent with Byte Swapping Only

Address Offset	PPC CPU Bus		PCI Bus	
	DH[0:31]	DL[0:31]	AD[63:32]	AD[31:0]
0x06	0xXXXXXXXX	0xXXXXAAXX	N/A	0xXXAAXXXX
0x07	0xXXXXXXXX	0xXXXXXXAA	N/A	0xAAXXXXXX

Table 14: 1-Byte CPU Write to a 32-bit PCI Agent with Word Swapping Only

Address Offset	PPC CPU Bus		PCI Bus	
	DH[0:31]	DL[0:31]	AD[63:32]	AD[31:0]
0x00	0xAAXXXXXX	0xXXXXXXXX	N/A	0xAAXXXXXX
0x01	0xXXAAXXXX	0xXXXXXXXX	N/A	0xXXAAXXXX
0x02	0xXXXXAAXX	0xXXXXXXXX	N/A	0xXXXXAAXX
0x03	0xXXXXXXAA	0XXXXXXXXX	N/A	0xXXXXXXAA
0x04	0XXXXXXXXX	0XAAXXXXXX	N/A	0xAAXXXXXX
0x05	0xXXXXXXXX	0xXXAAXXXX	N/A	0xXXAAXXXX
0x06	0xXXXXXXXX	0xXXXXAAXX	N/A	0xXXXXAAXX
0x07	0xXXXXXXXX	0xXXXXXXAA	N/A	0xXXXXXXAA

Table 15: 1-Byte CPU Write to a 32-bit PCI Agent with Both Byte and Word Swapping

Address Offset	PPC CPU Bus		PCI Bus	
	DH[0:31]	DL[0:31]	AD[63:32]	AD[31:0]
0x00	0xAAXXXXXX	0xXXXXXXXX	N/A	0xXXXXXXAA
0x01	0xXXAAXXXX	0xXXXXXXXX	N/A	0xXXXXAAXX
0x02	0xXXXXAAXX	0xXXXXXXXX	N/A	0xXXAAXXXX
0x03	0xXXXXXXAA	0XXXXXXXXX	N/A	0xAAXXXXXX
0x04	0XXXXXXXXX	0XAAXXXXXX	N/A	0xXXXXXXAA
0x05	0xXXXXXXXX	0xXXAAXXXX	N/A	0xXXXXAAXX
0x06	0xXXXXXXXX	0xXXXXAAXX	N/A	0xXXAAXXXX
0x07	0xXXXXXXXX	0xXXXXXXAA	N/A	0xAAXXXXXX

Table 16: 4-Byte CPU Write to a 32-bit PCI Agent with No Swapping

	PPC CPU Bus		PCI Bus	
Address Offset	DH[0:31]	DL[0:31]	AD[63:32]	AD[31:0]
0x00	0x12345678	0xXXXXXXXX	N/A	0x12345678

Table 17: 4-Byte CPU Write to a 32-bit PCI Agent with Byte Swapping Only

	PPC CPU Bus		PCI Bus	
Address Offset	DH[0:31]	DL[0:31]	AD[63:32]	AD[31:0]
0x00	0x12345678	0xXXXXXXXX	N/A	0x78564312

Table 18: 4-Byte CPU Write to a 32-bit PCI Agent with Word Swapping Only

	PPC CPU Bus		PCI Bus	
Address Offset	DH[0:31]	DL[0:31]	AD[63:32]	AD[31:0]
0x00	0x12345678	0xXXXXXXXX	N/A	0x12345678

Table 19: 4-Byte CPU Write to a 32-bit PCI Agent with Both Byte and Word Swapping

	PPC CPU Bus		PCI Bus	
Address Offset	DH[0:31]	DL[0:31]	AD[63:32]	AD[31:0]
0x00	0x12345678	0xXXXXXXXX	N/A	0x78564312

Table 20: 4-Byte CPU Write to a 64-bit PCI Agent with No Swapping

	PPC CPU Bus		PCI Bus	
Address Offset	DH[0:31]	DL[0:31]	AD[63:32]	AD[31:0]
0x00	0x12345678	0xXXXXXXXX	0x12345678	0xXXXXXXXX

Table 21: 4-Byte CPU Write to a 64-bit PCI Agent with Byte Swapping Only

	PPC CPU Bus		PCI Bus	
Address Offset	DH[0:31]	DL[0:31]	AD[63:32]	AD[31:0]
0x00	0x12345678	0xXXXXXXXX	0xXXXXXXXX	0x78564312

Table 22: 4-Byte CPU Write to a 64-bit PCI Agent with Word Swapping Only

Address Offset	PPC CPU Bus		PCI Bus	
	DH[0:31]	DL[0:31]	AD[63:32]	AD[31:0]
0x00	0x12345678	0xXXXXXXXX	0xXXXXXXXX	0x12345678

Table 23: 4-Byte CPU Write to a 64-bit PCI Agent with Both Byte and Word Swapping

Address Offset	PPC CPU Bus		PCI Bus	
	DH[0:31]	DL[0:31]	AD[63:32]	AD[31:0]
0x00	0x12345678	0xXXXXXXXX	0x78564312	0xXXXXXXXX

Table 24: 8-Byte CPU Write to a 32-bit PCI Agent with No Swapping

Address Offset	PPC CPU Bus		PCI Bus	
	DH[0:31]	DL[0:31]	AD[31:0] 0x4	AD[31:0] 0x0
0x00	0x41B12233	0x44000000	0x44000000	0x12345678

Table 25: 8-Byte CPU Write to a 32-bit PCI Agent with Byte Swapping Only

Address Offset	PPC CPU Bus		PCI Bus	
	DH[0:31]	DL[0:31]	AD[31:0] 0x4	AD[31:0] 0x0
0x00	0x41B12233	0x44000000	0x00000044	0x3322B141

Table 26: 8-Byte CPU Write to a 32-bit PCI Agent with Word Swapping Only

Address Offset	PPC CPU Bus		PCI Bus	
	DH[0:31]	DL[0:31]	AD[31:0] 0x4	AD[31:0] 0x0
0x00	0x41B12233	0x44000000	0x44000000	0x41B12233

Table 27: 8-Byte CPU Write to a 32-bit PCI Agent with Byte and Word Swapping

Address Offset	PPC CPU Bus		PCI Bus	
	DH[0:31]	DL[0:31]	AD[31:0] 0x4	AD[31:0] 0x0
0x00	0x41B12233	0x44000000	0x3322B141	0x00000044

Table 28: 8-Byte CPU Write to a 64-bit PCI Agent with No Swapping

Address Offset	PPC CPU Bus		PCI Bus	
	DH[0:31]	DL[0:31]	AD[63:32]	AD[31:0]
0x00	0x41B12233	0x44000000	0x41B12233	0x44000000

Table 29: 8-Byte CPU Write to a 64-bit PCI Agent with Byte Swapping Only

Address Offset	PPC CPU Bus		PCI Bus	
	DH[0:31]	DL[0:31]	AD[63:32]	AD[31:0]
0x00	0x41B12233	0x44000000	0x00000044	0x3322B141

Table 30: 8-Byte CPU Write to a 64-bit PCI Agent with Word Swapping Only

Address Offset	PPC CPU Bus		PCI Bus	
	DH[0:31]	DL[0:31]	AD[63:32]	AD[31:0]
0x00	0x41B12233	0x44000000	0x44000000	0x41B12233

Table 31: 8-Byte CPU Write to a 64-bit PCI Agent with Both Byte and Word Swapping

Address Offset	PPC CPU Bus		PCI Bus	
	DH[0:31]	DL[0:31]	AD[63:32]	AD[31:0]
0x00	0x41B12233	0x44000000	0x3322B141	0x00000044

Endianess from the Software Point of View

Big Endian

Since the Marvell device's CPU interface uses Big Endian and the Marvell device's internal always uses Little Endian, the data must be manually byte swapped in the program.

For example: If the data sheet lists the register content as in Table 32, the data must be byte swapped as in Table 33.

Table 32: Bit and Byte Orientation in the Internal Register

Data bit	31 . . . 28	27 . . . 24	23 . . . 20	19 . . . 16	15 . . . 12	11 . . . 08	07 . . . 04	03 . . . 00
Data byte	7	6	5	4	3	2	1	0
	MSB							LSB

Table 33: Bit and Byte Orientation in the Internal Register

Data bit	07 . . . 04	03 . . . 00	15 . . . 12	11 . . . 08	23 . . . 20	19 . . . 16	31 . . . 28	27 . . . 24
Data byte	1	0	3	2	5	4	7	6

Appendix A: Swapping Transaction Waveforms

This appendix contains CPU and PCI bus waveforms that illustrate the effects of the various swapping options.

There are two kinds of waveforms:

- CPU bus waveforms showing the originating transaction.
- PCI bus waveforms showing the resulting transaction based on the swap setting used.

The waveforms were captured on the CPU and PCI buses. Figure 1 shows a waveform of the CPU writing 1 byte to a PCI agent at address 0x120C0000. The data is 0xAA000000. Figure 2, Figure 3, Figure 4, and Figure 5 show the appearance of the data on the PCI bus for the different swap bit settings, for a 32-bit PCI agent. Figure 6, Figure 7, Figure 8, and Figure 9 show the data for a 64-bit PCI agent.

Figure 1: 1-Byte CPU Write to a PCI Agent at Address 0x120C0000

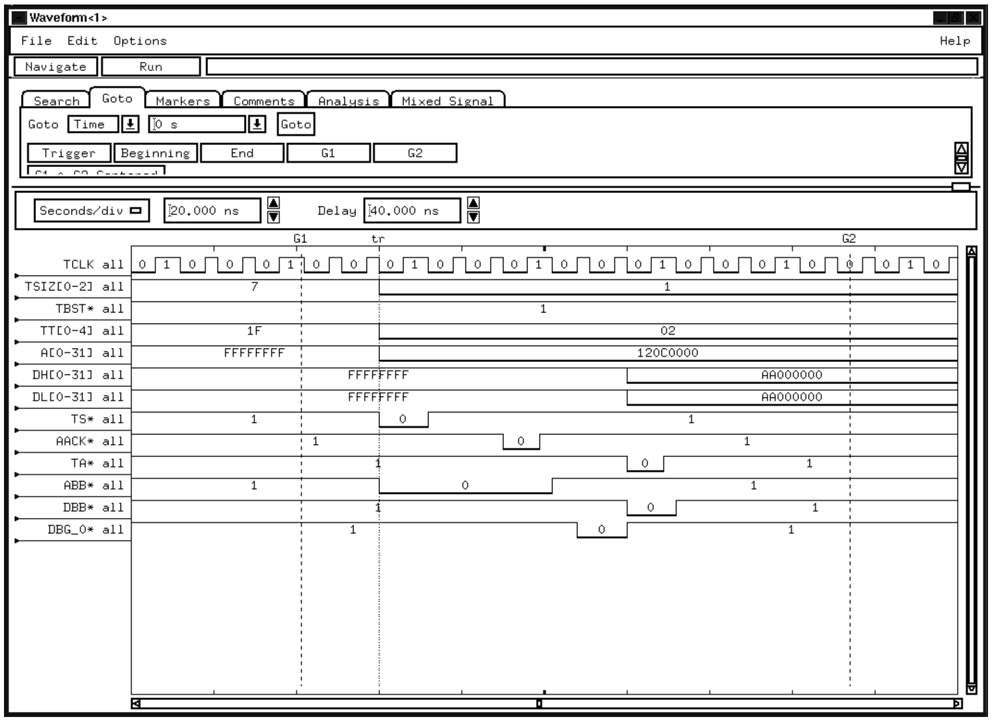


Figure 2: 1-Byte CPU Write to a 32-bit PCI Agent with No Swapping

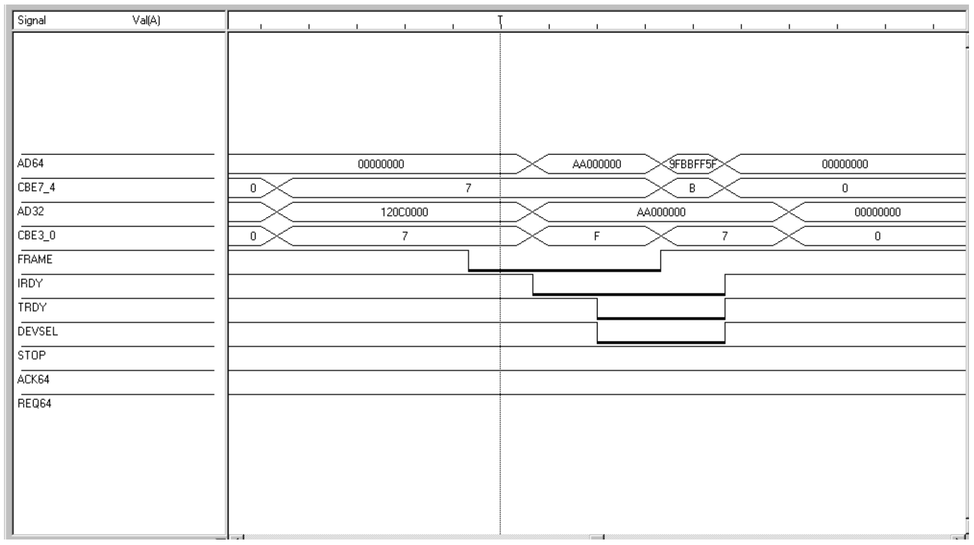


Figure 3: 1-Byte CPU Write to a 32-bit PCI Agent with Byte Swapping

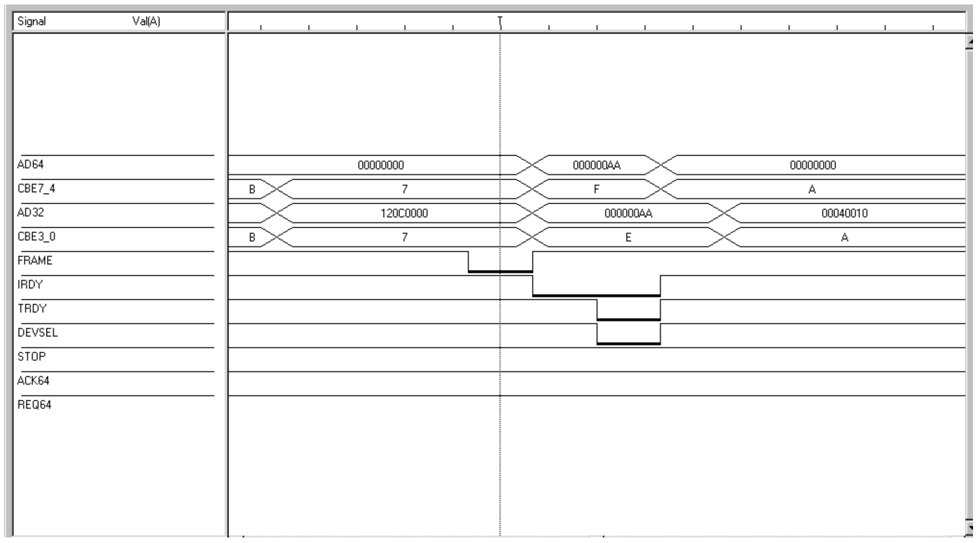


Figure 4: 1-Byte CPU Write to a 32-bit PCI Agent with Word Swapping

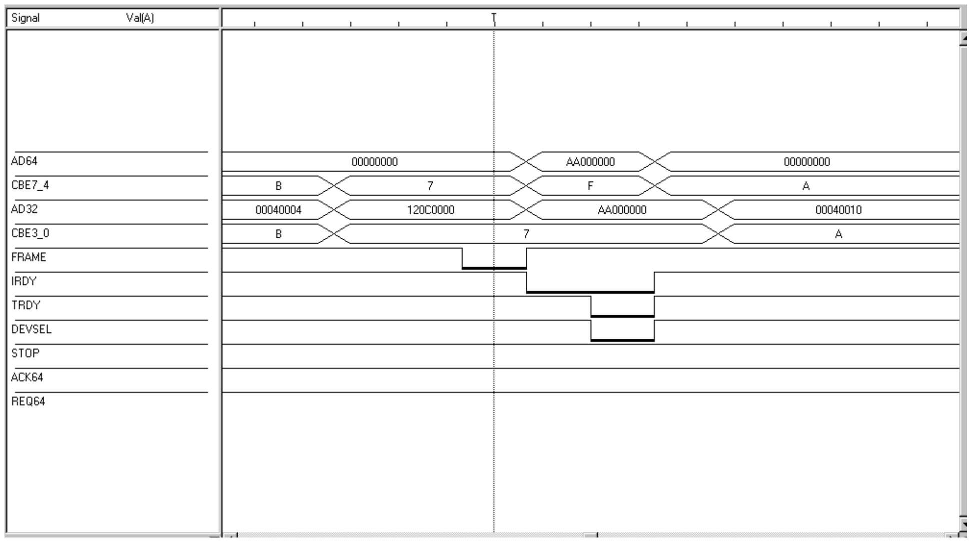


Figure 5: 1-Byte CPU Write to a 32-bit PCI Agent with Byte and Word Swapping

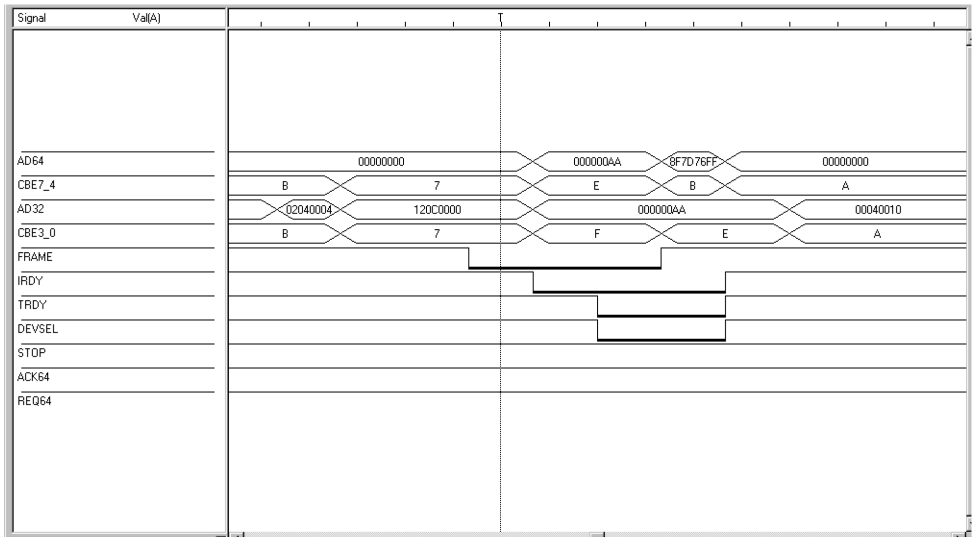


Figure 6: 1-Byte CPU Write to a 64-bit PCI Agent with No Swapping

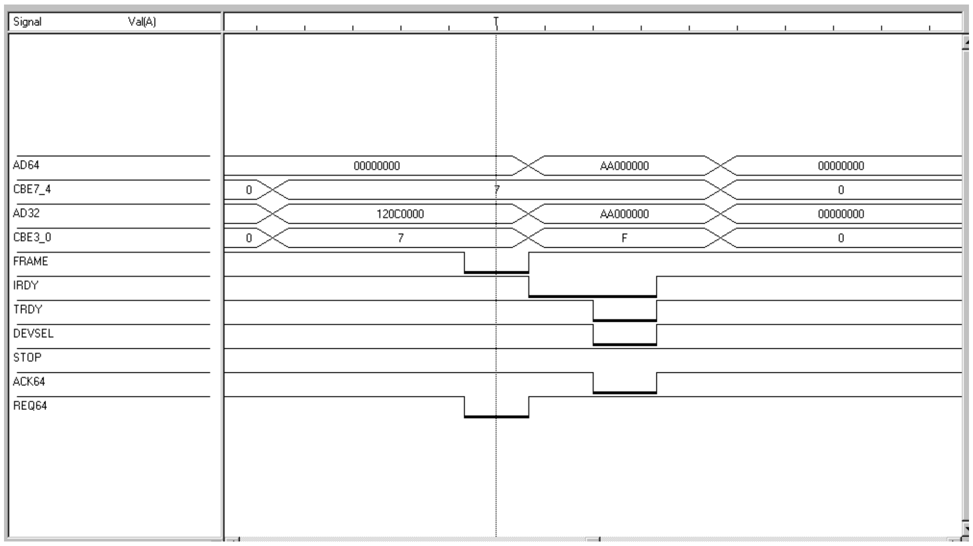


Figure 7: 1-Byte CPU Write to a 64-bit PCI Agent with Byte Swapping

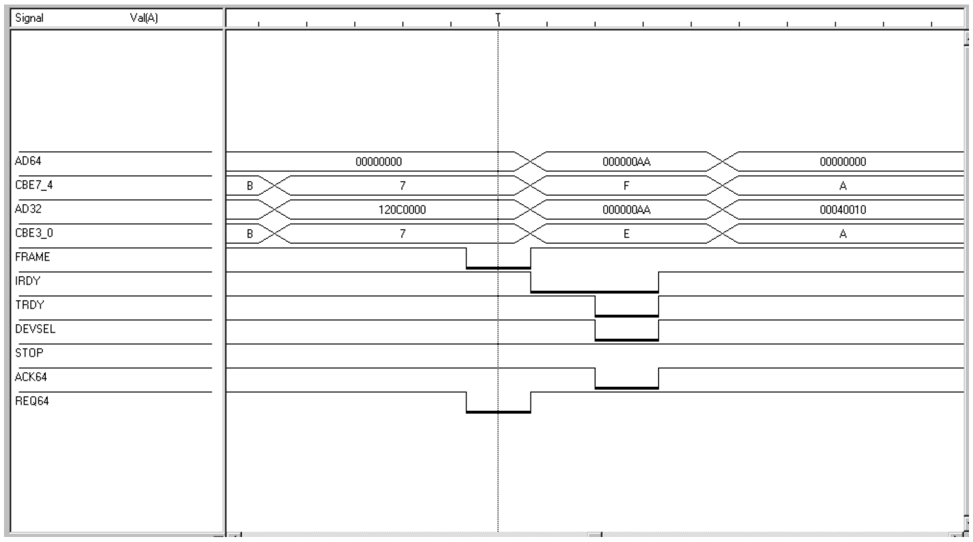


Figure 8: 1-Byte CPU Write to a 64-bit PCI Agent with Word Swapping

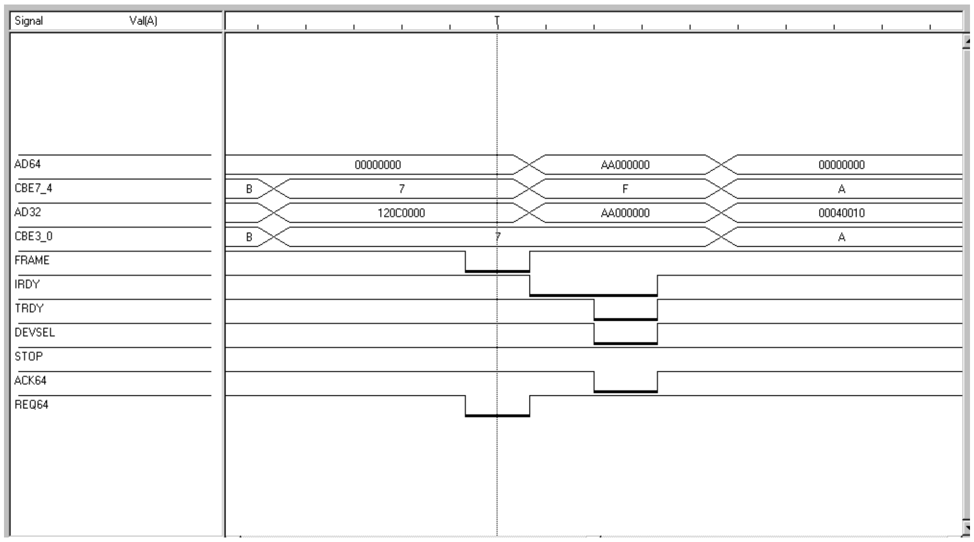


Figure 9: 1-Byte CPU Write to a 64-bit PCI Agent with Byte and Word Swapping

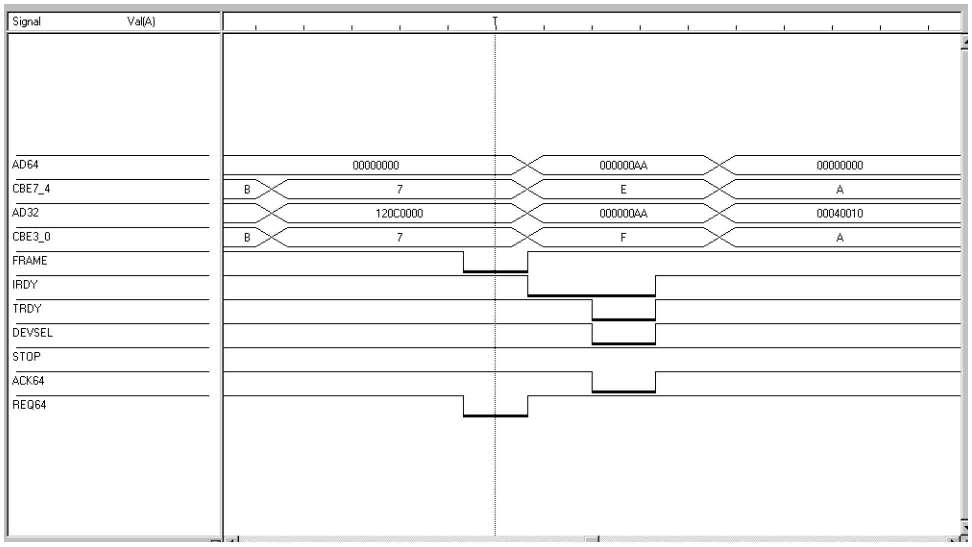


Figure 10 shows a waveform of the CPU writing 4 bytes to a PCI agent at address 0x120C0000. The data is 0x11223344. Figure 11, Figure 12, Figure 13, and Figure 14 show the appearance of the data on the PCI bus for the different swap bit settings, for a 32-bit PCI agent. Figure 15, Figure 16, Figure 17, and Figure 18 show the data for a 64-bit PCI agent.

Figure 10: Four-Byte CPU Write to a PCI Agent at Address 0x120C0000

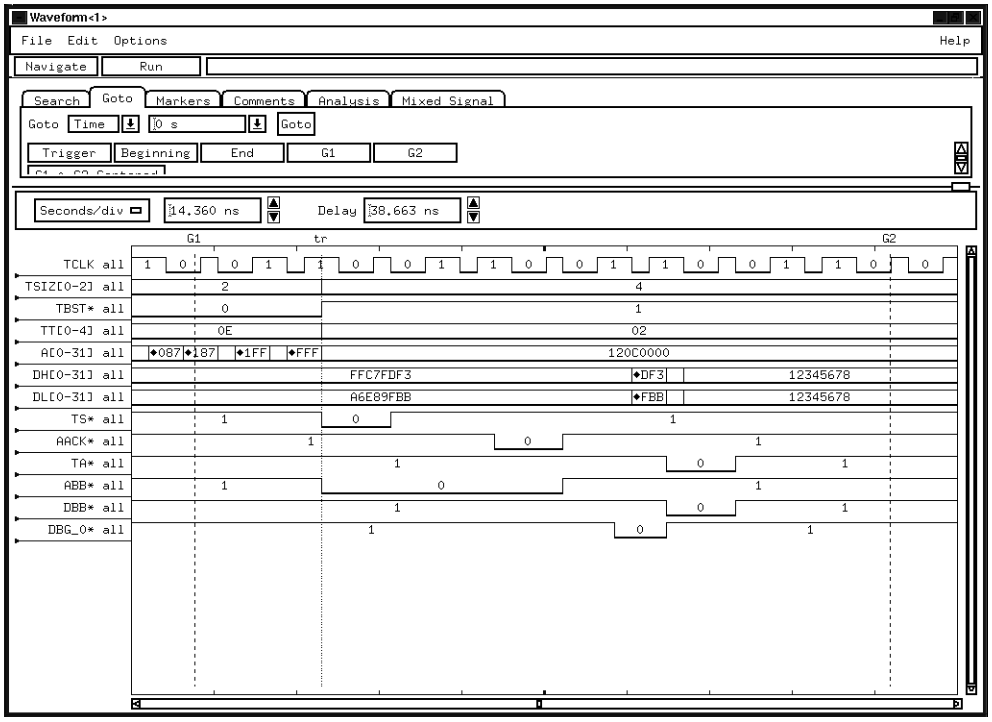

Waveform<1>
File Edit Options Help
Navigate Run
Search Goto Markers Comments Analysis Mixed Signal
Goto Time 0 s Goto
Trigger Beginning End G1 G2
Seconds/div 14.360 ns Delay 38.663 ns
G1 tr G2
TCLK all
TSIZ[0-2] all 2 4
TBST* all 0 1
TT[0-4] all 0E 02
A[0-31] all 087 187 1FF FFF 120C0000
DH[0-31] all FFC7FDF3 DF3 12345678
DL[0-31] all A6E89FBB FBB 12345678
TS* all
AACK* all
TA* all
ABB* all
DBB* all
DBG_0* all

Figure 11: Four-Byte CPU Write to a 32-bit PCI Agent with No Swapping

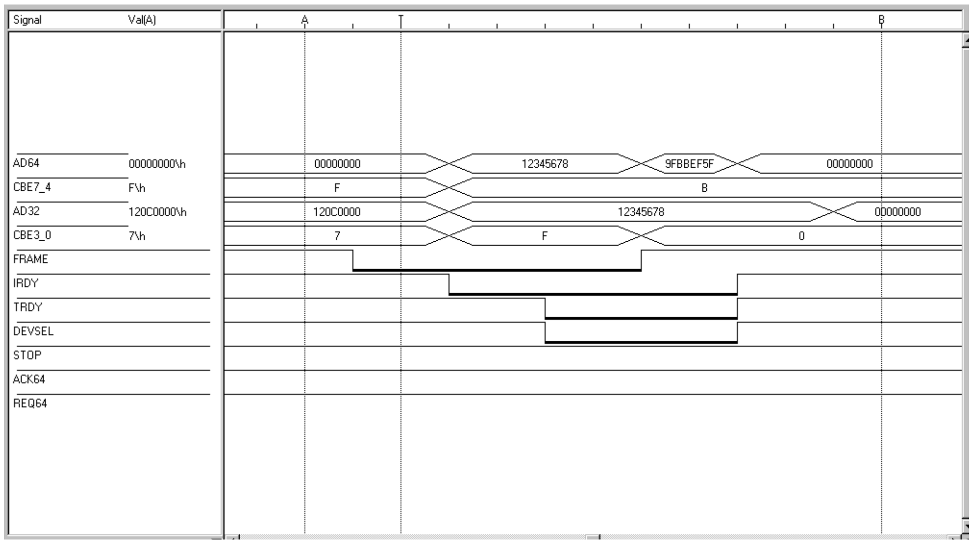


Figure 12: Four-Byte CPU Write to a 32-bit PCI Agent with Byte Swapping

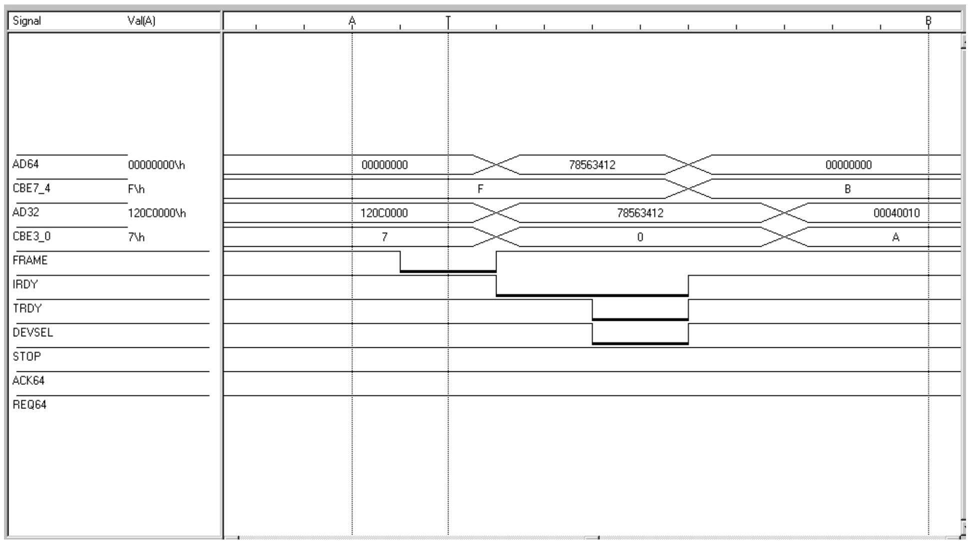


Figure 13: Four-Byte CPU Write to a 32-bit PCI Agent with Word Swapping

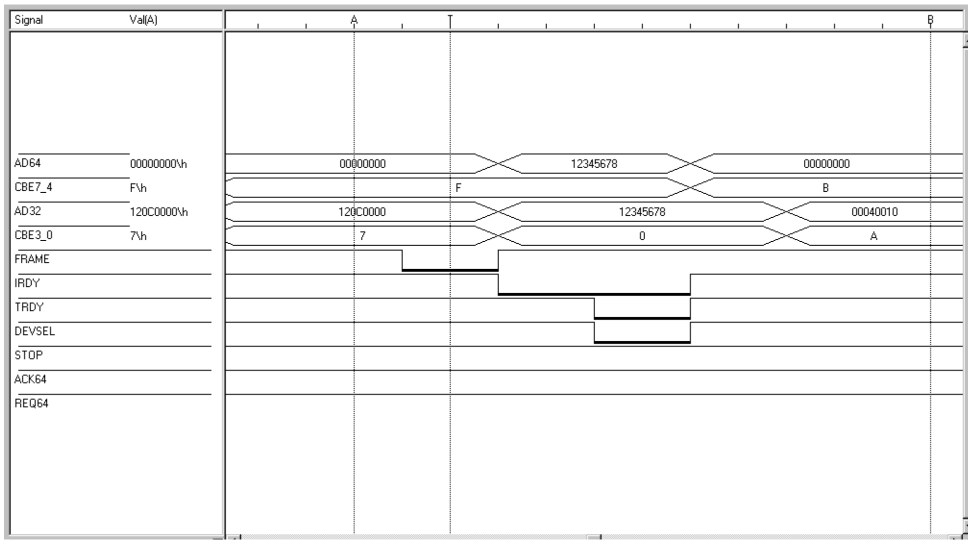


Figure 14: Four-Byte CPU Write to a 32-bit PCI Agent with Byte and Word Swapping

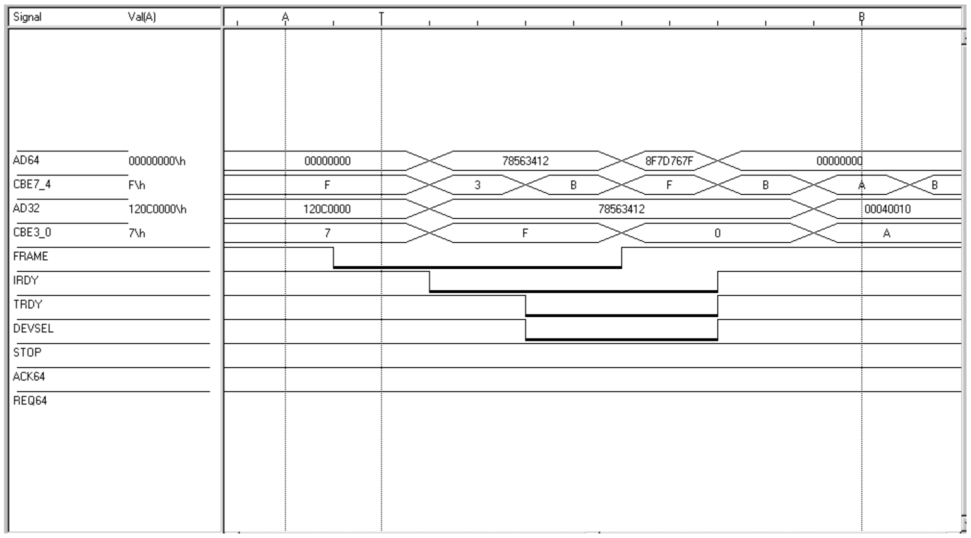


Figure 15: Four-Byte CPU Write to a 64-bit PCI Agent with No Swapping

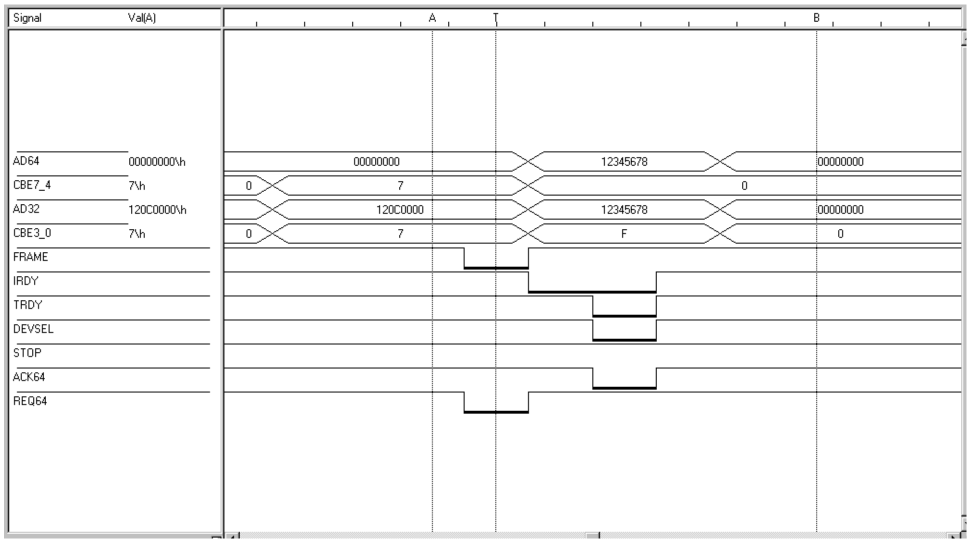


Figure 16: Four-Byte CPU Write to a 64-bit PCI Agent with Byte Swapping

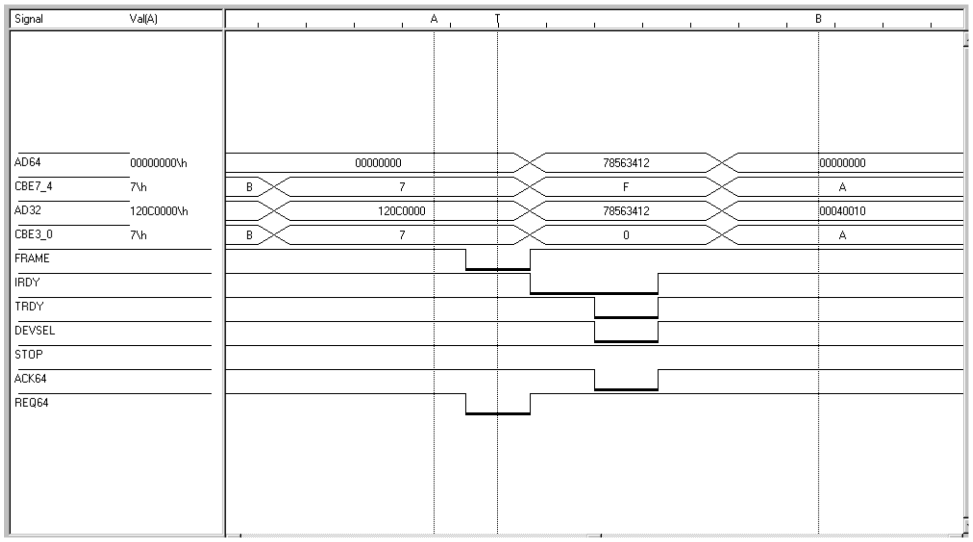


Figure 17: Four-Byte CPU Write to a 64-bit PCI Agent with Word Swapping

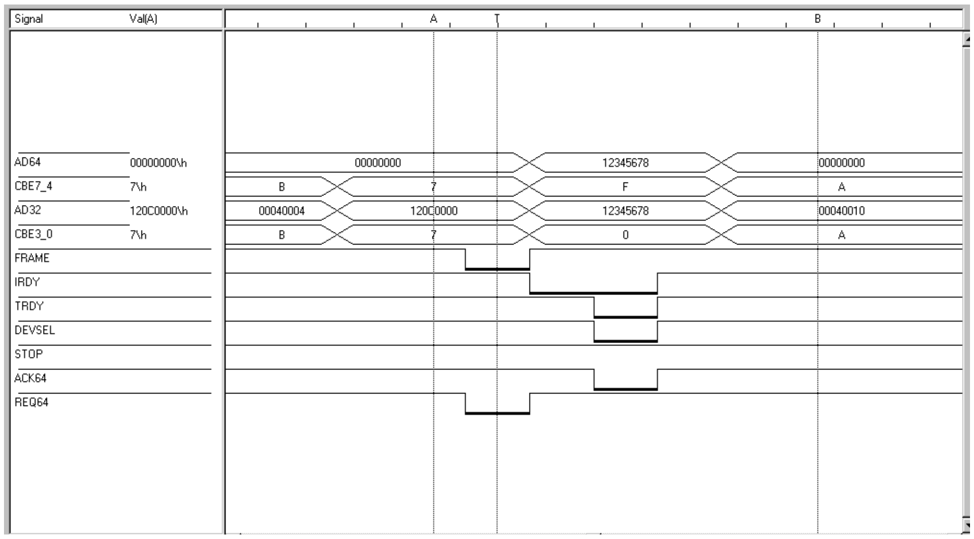


Figure 18: Four-Byte CPU Write to a 64-bit PCI Agent with Byte and Word Swapping

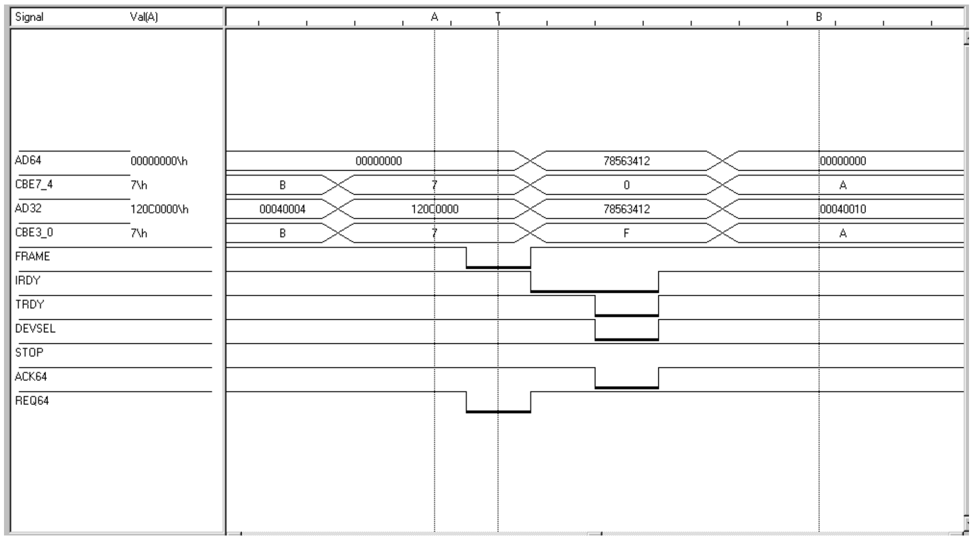
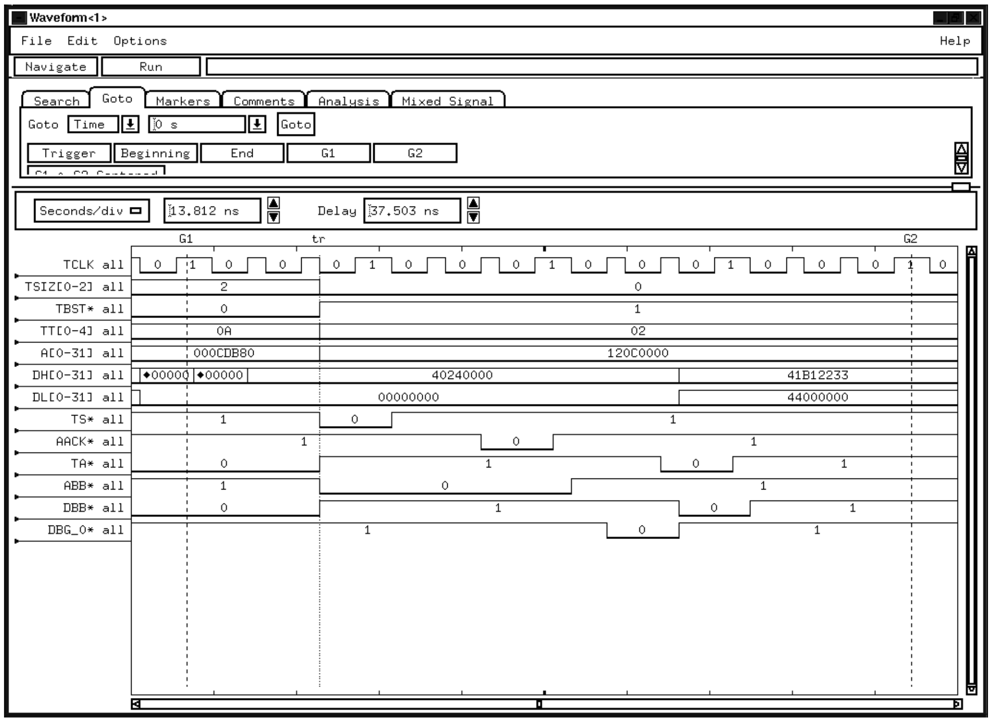


Figure 19 shows a waveform of the CPU writing 8 bytes to a PCI agent at address 0x120C0000. The data is 0x41B1223344000000. Figure 20, Figure 21, Figure 22, and Figure 23 show the appearance of the data on the PCI bus for the different swap bit settings, for a 32-bit PCI agent. Figure 24, Figure 25, Figure 26, and Figure 26 show the data for a 64-bit PCI agent.

Figure 19: Eight-Byte CPU Write to a PCI Agent at Address 0x120C0000



Goto Time 0 s

Seconds/div 13.812 ns Delay 37.503 ns

Figure 20: Eight-Byte CPU Write to a 32-bit PCI Agent with No Swapping

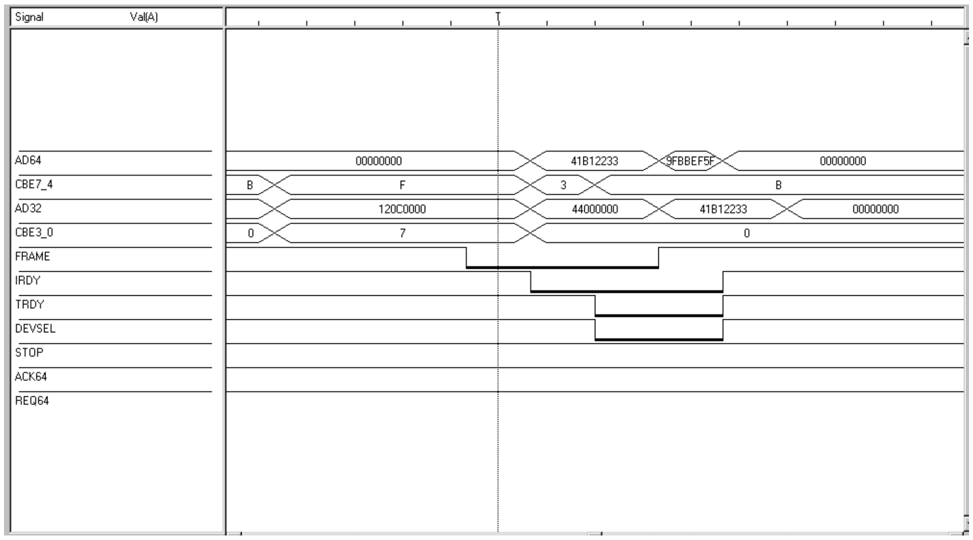


Figure 21: Eight-Byte CPU Write to a 32-bit PCI Agent with Byte Swapping

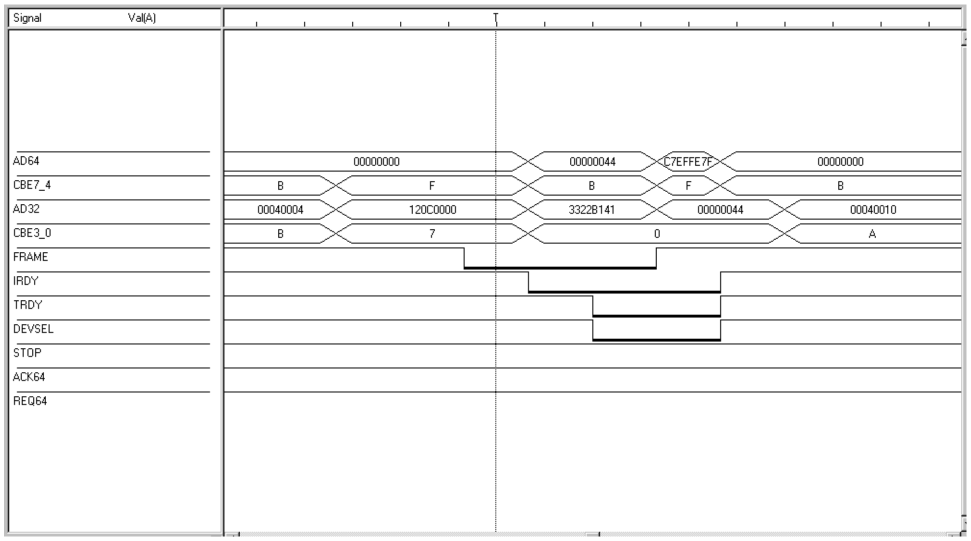


Figure 22: Eight-Byte CPU Write to a 32-bit PCI Agent with Word Swapping

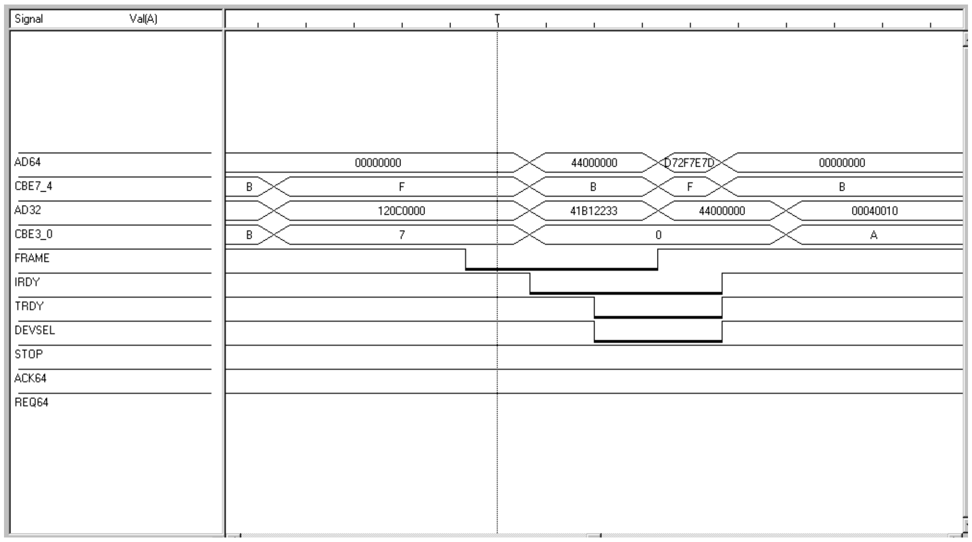


Figure 23: Eight-Byte CPU Write to a 32-bit PCI Agent with Byte and Word Swapping

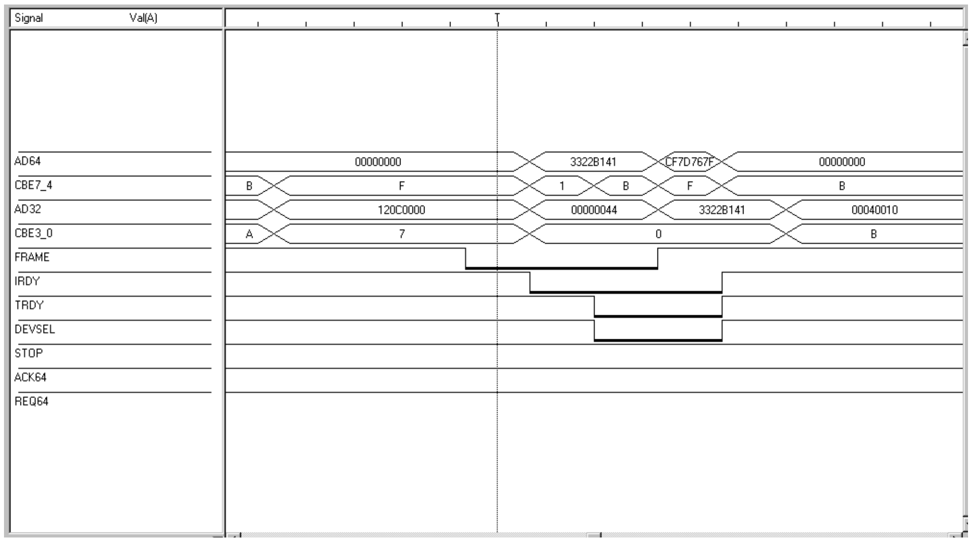


Figure 24: Eight-Byte CPU Write to a 64-bit PCI Agent with No Swapping

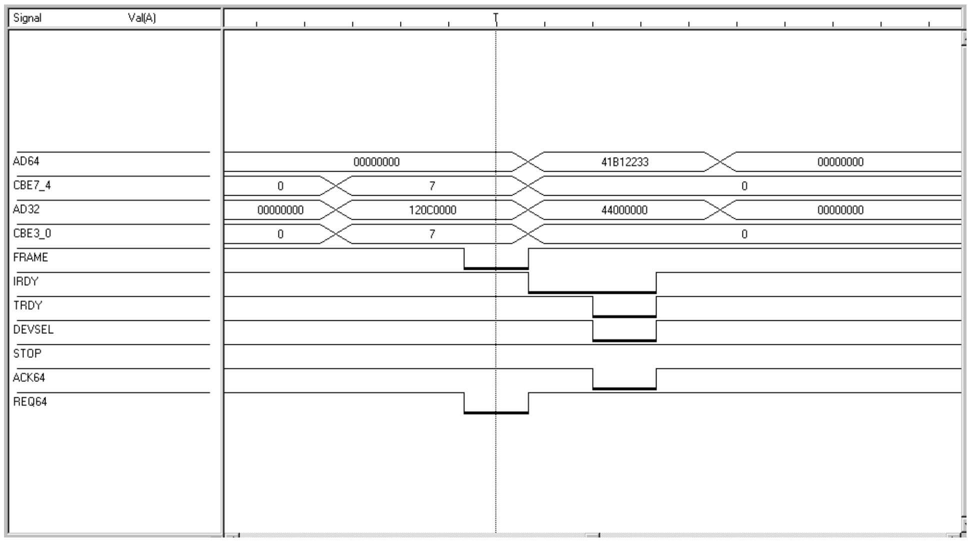


Figure 25: Eight-Byte CPU Write to a 64-bit PCI Agent with Byte Swapping

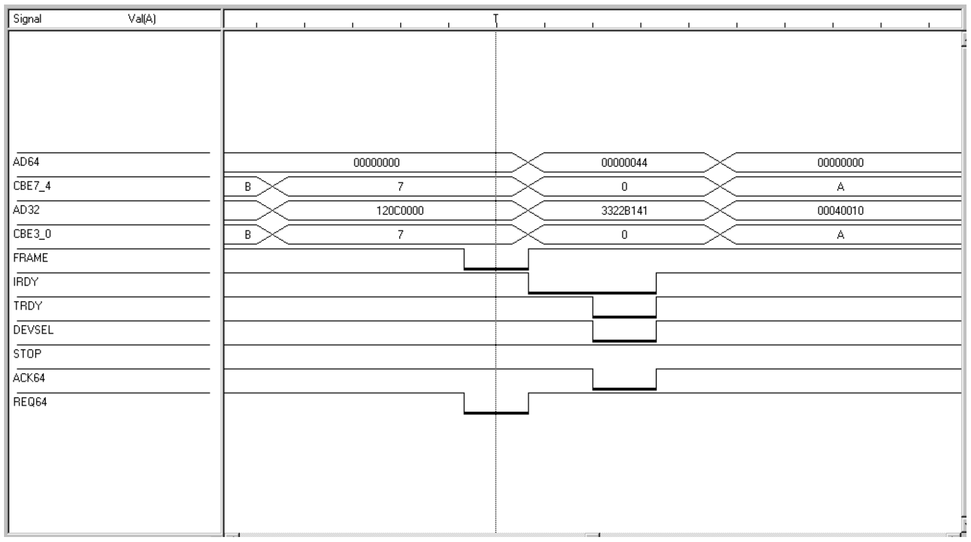


Figure 26: Eight-Byte CPU Write to a 64-bit PCI Agent with Word Swapping

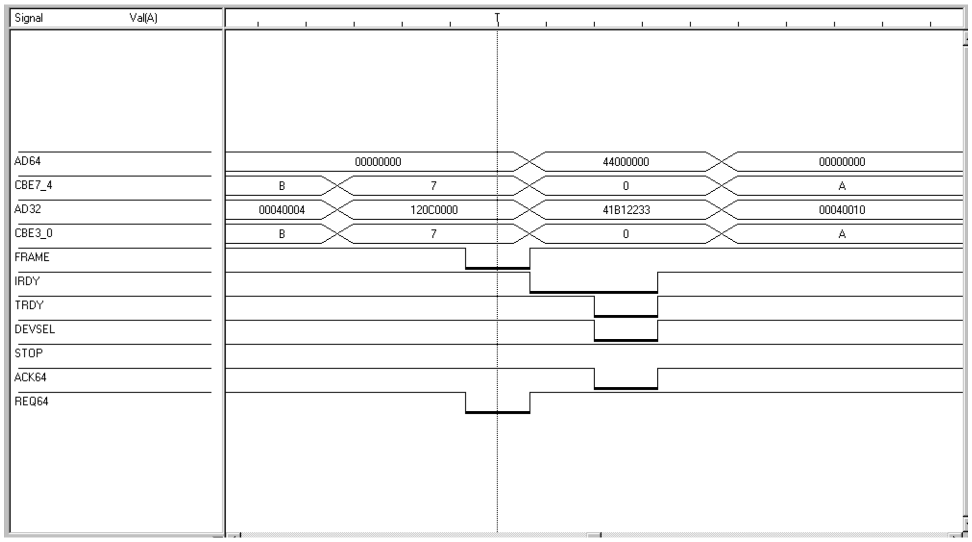
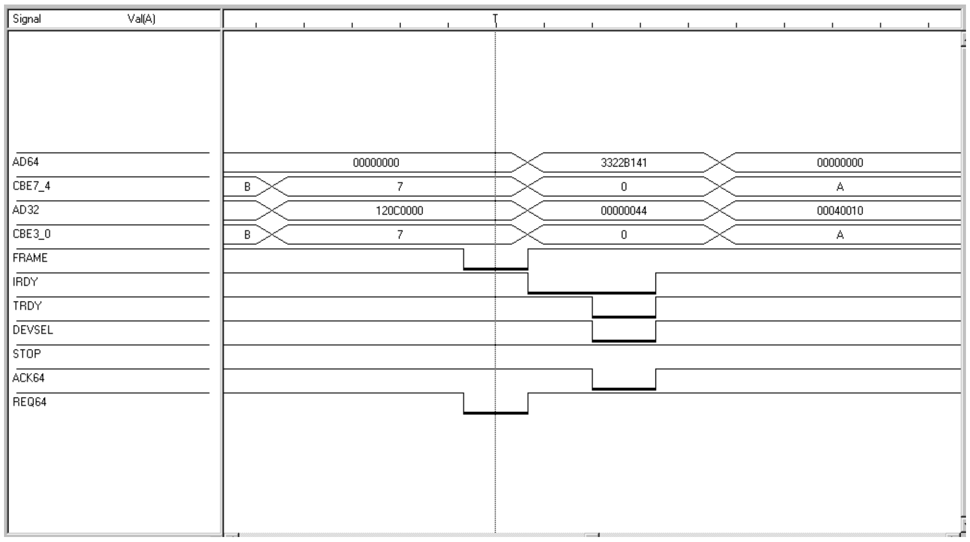


Figure 27: Eight-Byte CPU Write to a 64-bit PCI Agent with Byte and Word Swapping



Appendix B: Data Transfer Waveforms

This appendix contains CPU and device bus waveforms that show the data on the CPU bus and on the device bus.

The waveforms were captured on the CPU and device buses while the CPU wrote to various device data widths.

Figure 28: Double-Word (64-bit) CPU Write to 8-bit Device

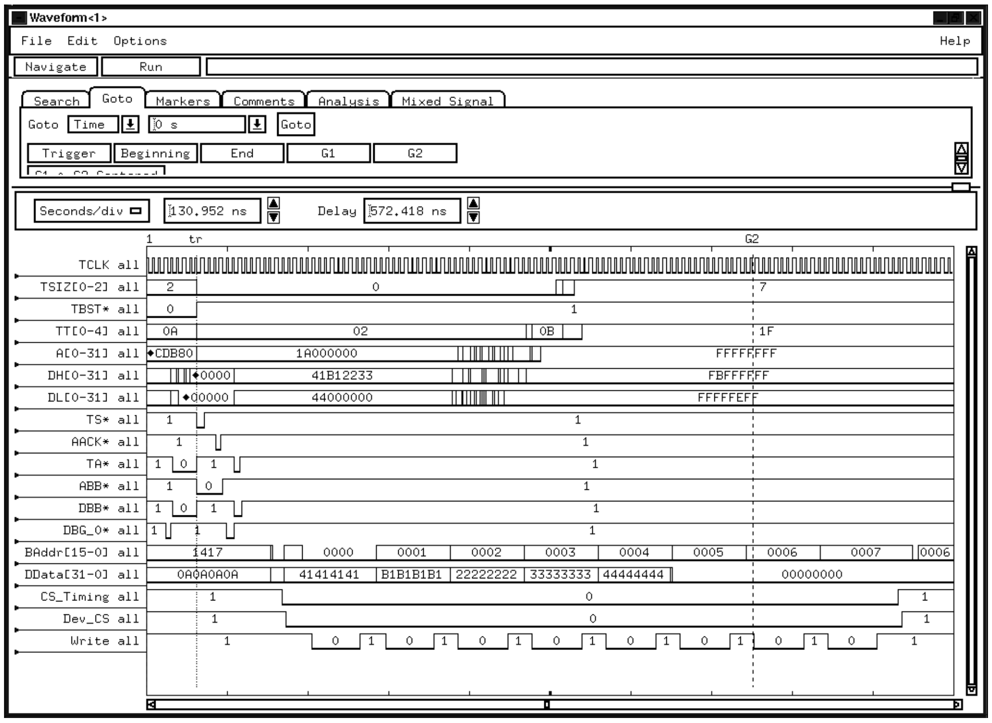
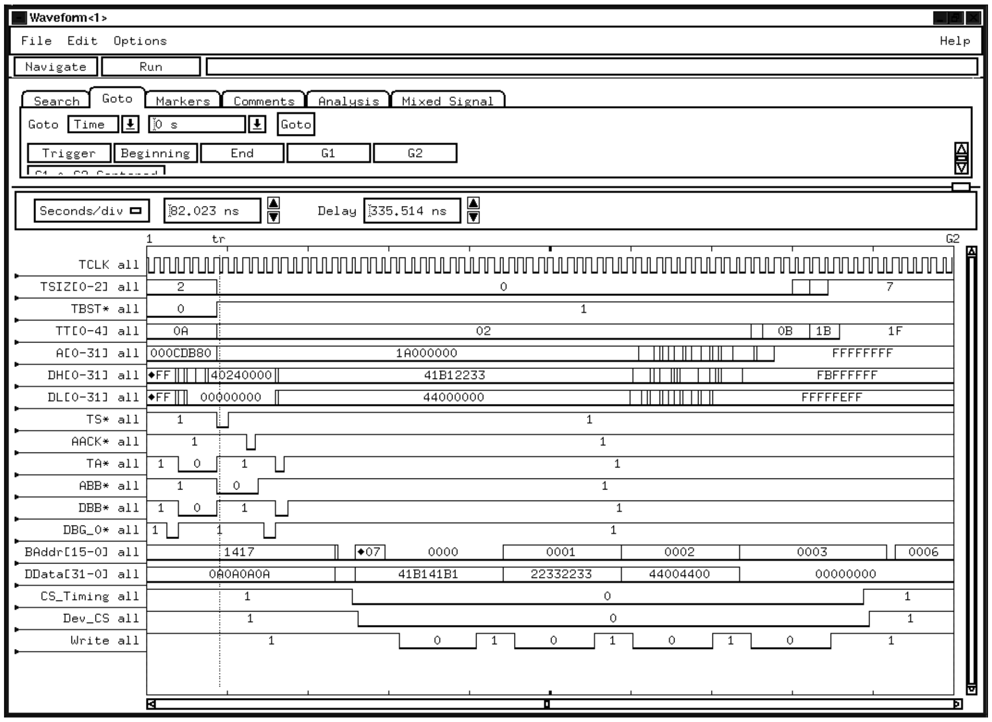

Waveform<1>
File Edit Options Help
Navigate Run
Search Goto Markers Comments Analysis Mixed Signal
Goto Time 0 s Goto
Trigger Beginning End G1 G2
Seconds/div 130.952 ns Delay 572.418 ns
TCLK all
TSIZ[0-2] all 2 0 7
TBST* all 0 1
TT[0-4] all 0A 02 0B 1F
A[0-31] all CDB80 1A000000 FFFFFFFF
DH[0-31] all 0000 41B12233 FBFFFFFF
DL[0-31] all 00000 44000000 FFFFFEFF
TS* all 1 1
AACK* all 1 1
TA* all 1 0 1 1
ABB* all 1 0 1
DBB* all 1 0 1 1
DBG_0* all 1 1 1
BAddr[15-0] all 1417 0000 0001 0002 0003 0004 0005 0006 0007 0006
DData[31-0] all 0A0A0A0A 41414141 B1B1B1B1 22222222 33333333 44444444 00000000
CS_Timing all 1 0 1
Dev_CS all 1 0 1
Write all 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

Figure 29: Double-Word (64-bit) CPU Write to 16-bit Device


Waveform<1>
File Edit Options Help
Navigate Run
Search Goto Markers Comments Analysis Mixed Signal
Goto Time 0 s Goto
Trigger Beginning End G1 G2
Seconds/div 82.023 ns Delay 335.514 ns
1 tr G2
TCLK all
TSIZ[0-2] all 2 0 7
TBST* all 0 1
TT[0-4] all 0A 02 0B 1B 1F
A[0-31] all 000CDB80 1A000000 FFFFFFFF
DH[0-31] all FF 40240000 41B12233 FBFFFFFF
DL[0-31] all FF 00000000 44000000 FFFFFEFF
TS* all 1 1
AACK* all 1 1
TA* all 1 0 1 1
ABB* all 1 0 1
DBB* all 1 0 1 1
DBG_0* all 1 1 1
BAddr[15-0] all 1417 07 0000 0001 0002 0003 0006
DData[31-0] all 0A0A0A0A 41B141B1 22332233 44004400 00000000
CS_Timing all 1 0 1
Dev_CS all 1 0 1
Write all 1 0 1 0 1 0 1 0 1

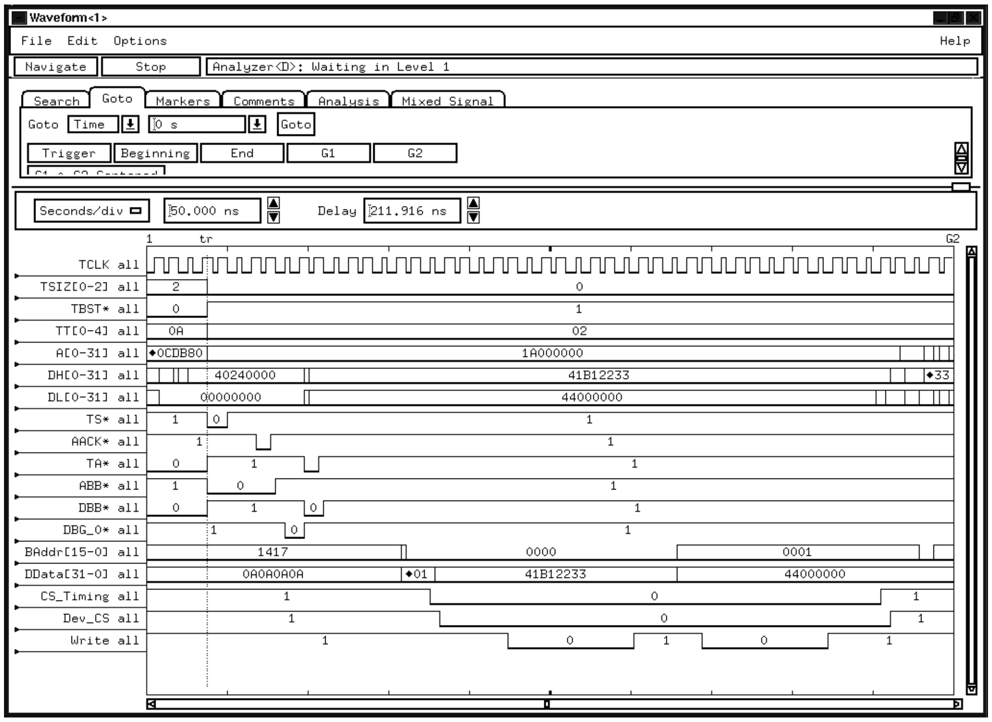
CONFIDENTIAL

Document Classification: Proprietary Information

Not approved by Document Control - For Review Only

June 3, 2002, Preliminary

Figure 30: Double-Word (64-bit) CPU Write to 32-bit Device


Waveform<1>
File Edit Options Help
Navigate Stop
Analyzer<D>: Waiting in Level 1
Search Goto Markers Comments Analysis Mixed Signal
Goto Time 0 s Goto
Trigger Beginning End G1 G2
Seconds/div 50.000 ns
Delay 211.916 ns
TCLK all
TSIZ[0-2] all 2 0
TBST* all 0 1
TT[0-4] all 0A 02
A[0-31] all 0CDB80 1A000000
DH[0-31] all 40240000 41B12233 33
DL[0-31] all 00000000 44000000
TS* all 1 0 1
AACK* all 1 1
TA* all 0 1 1
ABB* all 1 0 1
DBB* all 0 1 0 1
DBG_0* all 1 0 1
BAddr[15-0] all 1417 0000 0001
DData[31-0] all 0A0A0A0A 01 41B12233 44000000
CS_Timing all 1 0 1
Dev_CS all 1 0 1
Write all 1 0 1 0 1