Texas Instruments

*Application Report*

*SPRA674A - August 2001*

# *TMS320C6000 Expansion Bus: Multiple DSPs to i80960Kx/Jx Microprocessor Interface*

*Zoran Nikolic* *DSP Applications*

## ABSTRACT

This application note describes interface between the TMS320C6000™ DSPs and the i80960Jx/Kx microprocessor via the TMS320C6000 DSP expansion bus. This application report discusses the bus arbitration, the host to DSP and peer-to-peer DSP communication using the expansion bus synchronous mode.

The document cointains:

- A block diagram of the interface, and the glue logic VHDL code
- Information required for configuring the DSPs and the i80960Jx/Kx
- Timing diagrams illustrating the interface functionality

The interface can be easily extended to more than two DSPs.

**NOTE:** The information presented here has been verified using VHDL simulation.

## Contents

## List of Figures

## List of Tables

# 1 i80960Jx/Kx Interface

Interfacing the i80960Jx/Kx series of microprocessors to one TMS320C6000 DSP expansion bus is straightforward since the i80960Jx/Kx microprocessors have a multiplexed data/address bus similar to the expansion bus. Interfacing the i80960Jx/Kx to more than one TMS320C6000 DSP requires additional logic, especially if peer-to-peer DSP communication is required.

In a system with multiple DSPs and a host interfacing to the expansion bus in synchronous mode, several issues must be considered:

- The Expansion Bus Internal Slave Address (XBISA) register has to be configured prior to a data transfer to the DSP. The Auxiliary DMA channel moves data to/from a location in the DSP memory specified by the XBISA register. In a system with more than two potential bus masters, one has to make sure that the device performing a data transfer was the last device that modified the XBISA register.
- Peer-to-peer DSP communication using the TMS320C6000 expansion bus requires additional logic. The DSP that initiated a transfer may not be able to provide/accept data in every cycle. In other words, the master DSP can experience breaks during a transfer. Whenever the initiator DSP is not able to transfer a word, it asserts the XWAIT signal. The target DSP does not have any way of interpreting the initiator’s XWAIT signal, and thus, does not know when the initiator is not ready.

To address the issues mentioned above in a system with the i80960Jx/Kx and two DSPs, external glue logic is required.

Figure 1 shows the interface between i80960Jx/Kx and the expansion bus. Note that the internal bus arbiter of the expansion bus is disabled. In this system, the i80960Jx/Kx host can talk to either of the DSPs, and the DSPs can communicate to each other (DSPs can not master the i80960).

The expansion bus control signals from both DSPs as well as the host control signals pass through the glue logic. Thus, the glue logic can pass/change the control signals from the initiator to the target. The data bus (XD[31:0]), byte enables (XBE[3:0]), and clock (XCLKIN) are the only common signals for the two DSPs and the host (these signals are also connected to the glue logic). The glue logic drives the data bus and the byte enables only when correction of the target's XBISA register is required after the initiator was backed off. This is discussed in section 2.

Data lines XD[31:30] are used by the glue logic to decode a target DSP (a combination XD[31:30] = '01' corresponds to DSP1, while combination XD[31:30] = '10' corresponds to DSP2). The data line XD[29] is used as the control signal (XCNTL) that selects between the XBISA and the XBD register.

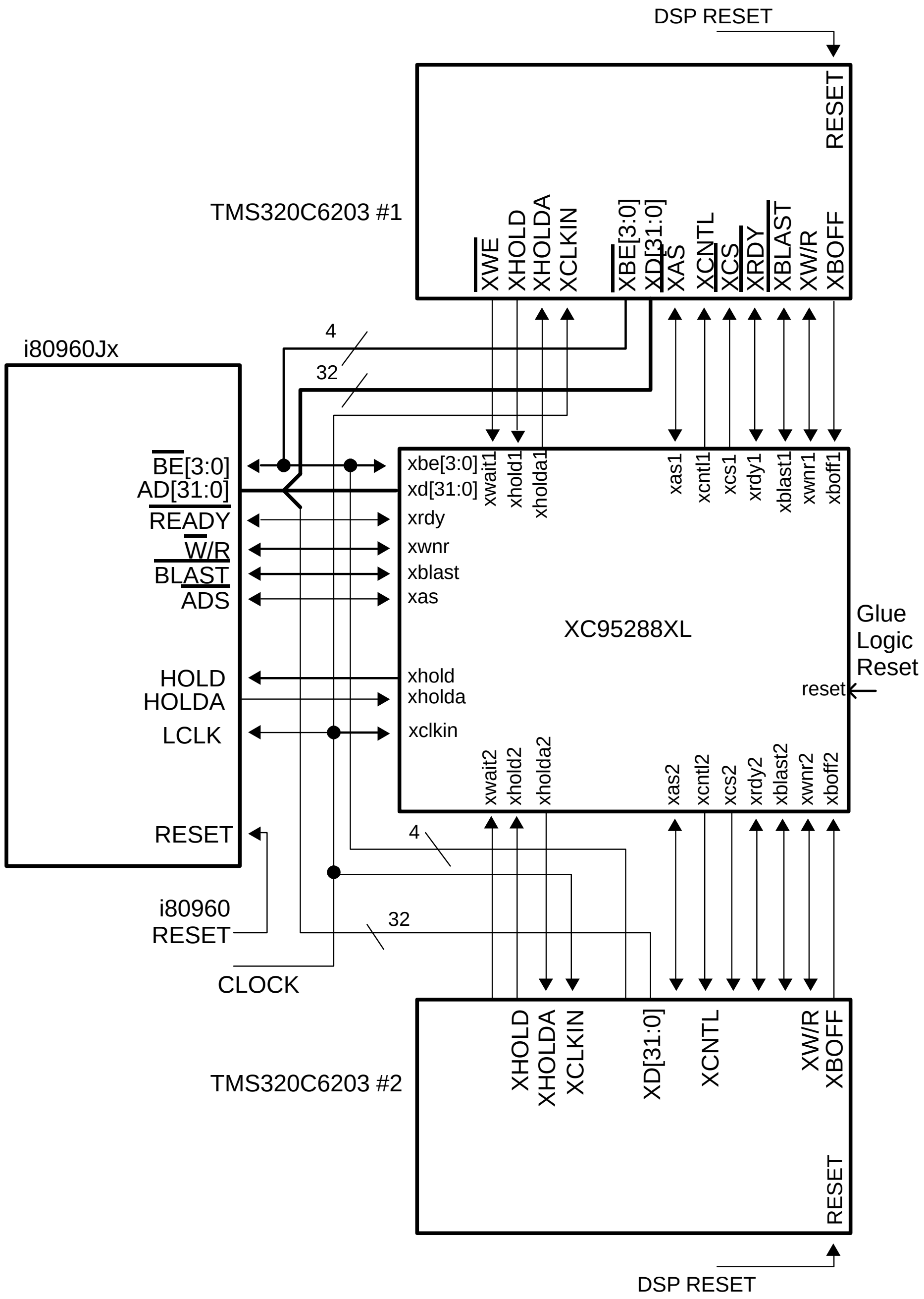

**Figure 1. Interface of the i80960Jx/Kx to Two DSPs Using the Expansion Bus**

When a 5 V host device is used (for example i80960Kx), the voltage translation interface is needed. The voltage translation logic is not shown in the block diagram.

Special care should be taken when bringing the system out of reset. When coming out of reset, the DSPs use the data pins on the expansion bus to latch a configuration word. The configuration word is set using pullup/down resistors. When in reset the Intel microprocessor drives the bus, which can prevent the DSPs from latching proper configuration values. In other words, when the DSP and the i80960Kx/Jx are in reset together, the DSP can not latch proper configuration values since the i80960Jx drives the bus. To prevent this situation, the following reset sequence is recommended:

1. All resets should be asserted together (i80960 RESET, DSP RESET and Glue Logic RESET). As long as they are in reset, the glue logic should drive the HOLD signal of the host high requesting the bus (this prevents the host from driving the bus and corrupting configuration of the DSPs).
2. The host (i80960) should be allowed to come out of reset first (while the DSPs and the glue logic are still in reset). The host will come out from reset, but since the HOLD signal is asserted by the glue logic, the host will not drive the bus.
3. The DSP RESET should be deasserted second. Since the host is not driving the data bus, the DSPs will latch the proper configuration word (the configuration word has to be stable five DSP clocks before, and five DSP clocks after, the rising edge of the DSP RESET).
4. Finally, the Glue Logic Reset should be deasserted. By deasserting the Glue Logic Reset, the bus ownership is returned to the host (after the Glue Logic Reset has been deasserted, the glue logic will stop driving the host’s HOLD signal high).

By following these four steps, the i80960Kx/Jx will not interfere with the DSP configuration procedure.

# 2 Configuration

The only programmable physical memory attribute for the i80960Jx microprocessor is the bus width, which can be 8-, 16- or 32-bits wide.

When assigning memory attributes, the physical address space is partitioned into eight fixed 512 MByte regions, determined by the upper three address bits. The physical memory attributes for each region is programmable through the PMCON registers. The PMCON registers are loaded from the Control Table. The i80960 Jx microprocessor provides one PMCON register for each region. The bus width for a region is controlled by the BW[1:0] bits in the PMCON register. The BW1 = 1 and BW0 = 0 for 32-bit expansion bus interface.

All eight PMCON registers are loaded automatically during system autoinitialization. Immediately after a hardware reset, the PMCON register contents are marked invalid in the Bus Control (BCON) register. The initial PMCON register values are stored in the Control Table in the Initialization Boot Record. After hardware reset, the processor first loads all PMCON registers from the Control Table. The processor then loads BCON from the Control Table. The BCON.ctv bit in BCON has to be set to use the programmed PMCON values for each memory region.

The Default Logical Memory Configuration register (DLMCON) provides default logical memory control for those accesses which do not fall within a region defined by the logical memory control register pairs. On the 80960Jx, the byte order programmed in the DLMCON register controls byte ordering for the entire 32-bit memory space. The DCEN bit field of the DLMCON register has to be set to zero to disable data caching. The BE bit field of the DLMCON register has to be set to zero to enable Little endian byte order for all accesses.

The interrupt controller register of the 80960Jx processors controls basic functionality such as interrupt mode, signal detection, global enable/disable, mask operation, interrupt vector caching, and sampling mode (for more detailed information on interrupt configuration please see the i80960 Jx User's Manual).

In the system, the two DSPs are configured using same pullup/down resistors on the data bus. When getting out of reset both DSPs sample the same configuration word set by pullup/down resistors on the data bus. The TMS320C6000 expansion bus boot configuration is presented in Table 1.

**Table 1. TMS320C6000 Expansion Bus Boot Configuration via Pullup/ Pulldown Resistors on XD[31:0]**

| Field | Description |
|---|---|
| BLPOL | Determines polarity of the $\overline{\text{XBLAST}}$ signal<br>BLPOL = 0, $\overline{\text{XBLAST}}$ is active low. |
| RWPOL | Determines polarity of expansion bus read/write signal<br>RWPOL = 0, XW/R_. |
| HMOD | Host mode (status in XB HPIC)<br>HMOD = 1, external host interface is in synchronous master/slave mode |
| XARB | Expansion bus arbiter (status in XBGC)<br>XARB = 0, internal expansion bus arbiter is disabled. |
| FMOD | Fifo mode (status in XBGC) |
| LEND | Little endian mode<br>LEND = 1, system operates in Little Endian Mode |
| BootMode[4:0] | Dictates the boot-mode of the device, including host port boot, ROM boot, memory map selection. For complete list of boot modes, refer to the *TMS320C6000 Peripherals Reference Guide* (SPRU190). |

# 3 Glue Logic Description

The glue logic takes care of the bus arbitration and makes the host-to-DSP and peer-to-peer DSP communication possible. A block diagram of the finite state machine implemented in the glue logic is presented in Figure 2. The machine has nine states.

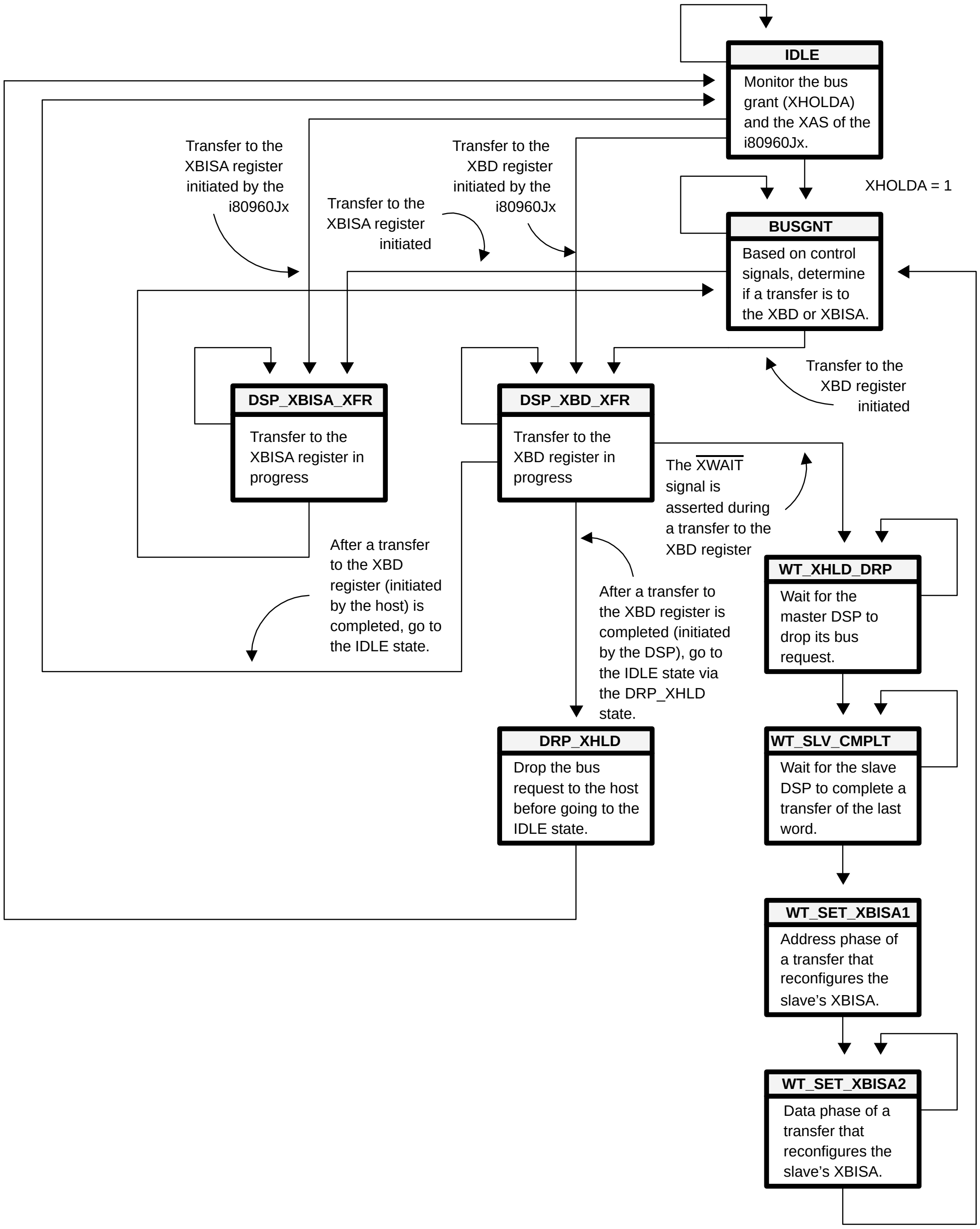
IDLE
Monitor the bus grant (XHOLDA) and the XAS of the i80960Jx.
XHOLDA = 1
BUSGNT
Based on control signals, determine if a transfer is to the XBD or XBISA.
Transfer to the XBISA register initiated by the i80960Jx
Transfer to the XBISA register initiated
Transfer to the XBD register initiated by the i80960Jx
Transfer to the XBD register initiated
DSP_XBISA_XFR
Transfer to the XBISA register in progress
DSP_XBD_XFR
Transfer to the XBD register in progress
The $\overline{\text{XWAIT}}$ signal is asserted during a transfer to the XBD register
After a transfer to the XBD register (initiated by the host) is completed, go to the IDLE state.
After a transfer to the XBD register is completed (initiated by the DSP), go to the IDLE state via the DRP_XHLD state.
WT_XHLD_DRP
Wait for the master DSP to drop its bus request.
DRP_XHLD
Drop the bus request to the host before going to the IDLE state.
WT_SLV_CMPLT
Wait for the slave DSP to complete a transfer of the last word.
WT_SET_XBISA1
Address phase of a transfer that reconfigures the slave's XBISA.
WT_SET_XBISA2
Data phase of a transfer that reconfigures the slave's XBISA.

**Figure 2. Glue Logic State Machine Diagram**

By default, the bus is owned by the i80960 microprocessor, and the DSPs have their internal bus arbiters disabled. To request the bus, the DSP asserts the bus request (the DSP1 asserts the XHOLD1; the DSP2 asserts the XHOLD2) to the glue logic. The bus requests from the DSPs (XHOLD1 and XHOLD2) are inputs to the glue logic (see Figure 1). The glue logic requests the bus from the host by asserting the XHOLD signal. The host grants the bus to the glue logic by asserting the XHOLDA signal. After receiving the bus grant from the host, the glue logic grants the bus to a DSP (by asserting either the XHOLDA1 or XHOLDA2). To make sure that the device performing a data transfer was the last device to modify the XBISA register, the glue logic allows the bus to change ownership only after a transfer to the XBD register is completed. If an initiator performed a transfer to the XBISA register, the glue logic will keep the expansion bus granted to the initiator until the initiator completes a transfer to the XBD register.

The bus ownership can be changed only when in the **IDLE** state. If a transfer is initiated by the host, depending on the XD[29] (XD[29] is used as the XCNTL signal), the state machine will jump from the **IDLE** state to the **DSP_XBD_XFR** or the **DSP_XBISA_XFR** state (see Figure 2). Note that the state machine returns to the **IDLE** state only via **DSP_XBD_XFR** state.

If the i80960Jx/Kx (host) is the bus master, and if the XHOLD1 (XHOLD2) gets asserted after/during a transfer to the XBISA register, the glue logic will keep the XHOLD signal low until the host completes a transfer to the XBD register.

If the bus is requested by the DSP, the state machine will jump from the **IDLE** state to the **BUSGNT** state only after the bus grant (XHOLDA) has been asserted by the host. In the **BUSGNT** state, the glue logic grants the bus to the DSP. Two flip-flops dedicated to hold the bus master identification number are also set (the DSP1 identification number is ‘01’, the DSP2 identification number is ‘10’, and the host identification number is ‘00’). When the DSP initiates a transfer, depending on the XD[29] (XD[29] is used as the XCNTL signal), the state machine jumps from the **BUSGNT** state to either the **DSP_XBD_XFR** or the **DSP_XBISA_XFR** state (see Figure 2). Note that the state machine returns to the **IDLE** state only from the **DSP_XBD_XFR** (via **DRP_XHLD**) state. When in the **DRP_XHLD** state, the glue logic drops the bus request to the host (XHOLD), and resets the flip-flops that hold the current bus master identification number before returning to the **IDLE** state.

If the DSP is the bus master, to make sure that the DSP was the last to modify the other DSP’s XBISA register, the glue logic will drop the bus request to the host only after the DSP completes a transfer to the XBD register. The glue logic will keep the bus request to the host (XHOLD) asserted as long as a transfer to the XBD is not completed.

Whenever the initiator DSP is not capable of sending/receiving a new word of data during the transfer, the initiator’s $\overline{\text{XWAIT}}$ signal gets asserted. As mentioned earlier, the problem in peer-to-peer DSP communication is that the target DSP does not have a way of interpreting the initiator’s $\overline{\text{XWAIT}}$ signal. The target DSP assumes that the initiator is capable of providing/accepting a new word of data every clock cycle. To prevent data corruption every time the $\overline{\text{XWAIT}}$ gets asserted, the glue logic backs off the initiator DSP and ends a transfer on the target DSP side (by asserting the target’s blast signal).

If the $\overline{\text{XWAIT}}$ gets asserted during a transfer to the XBD register, the glue logic asserts the $\overline{\text{XBLAST}}$ signal to the target DSP. At the same time, the glue logic backs off the initiator DSP by asserting the initiator’s $\overline{\text{XBOFF}}$ and deasserts the initiator’s $\overline{\text{XRDY}}$ signal. The state machine then waits for the target DSP to complete the last transfer, and for the initiator to drop its bus request (states **WT_XHLD_DRP** and **WT_SLV_CMPLT**).

At this point, the target's XBISA register is corrupted since one word of data was transferred while initiator was not ready (the XWAIT asserted). Before granting the bus back to the initiator DSP and allowing the initiator to continue the glue logic has to adjust the target's XBISA register. This is done in the states **WT_SET_XBISA1** and **WT_SET_XBISA2**.

Every time the bus master performs an access to the XBISA register, the XBISA value is latched in the internal glue logic register. During data transfers (transfers to the XBD register), the glue logic keeps track of the target's XBISA content by incrementing the internal XBISA register (by four) on every data transfer, if the XWAIT is not asserted. This way the glue logic can correct the target's XBISA register after the XWAIT is asserted.

The XWAIT signal is not monitored during a transfer to the XBISA register (see Figure 2). Transfers to the XBISA register are expected to be bursts of one word (the XWAIT does not get asserted during one word burst transfers).

A brief state machine description is given in Table 2. States are encoded using one-hot encoding (one flip-flop is used for each of the nine states).

**Table 2. Glue Logic State Machine Description**

| State Name | State Code | Brief Description |
|---|---|---|
| IDLE | 0x001 | In this state, the glue logic monitors the host's bus grant (XHOLDA) and the address strobe (XAS). This is the only state allowing change of the bus ownership.<br>In this state, the glue logic can assert the XHOLD. The XHOLD will be asserted if the XHOLD1 or the XHOLD2 are asserted.<br>If the host grants the bus (XHOLDA asserted), the flip-flops that hold the bus master identification number are set correspondingly (the DSP1 has priority over the DSP2). |
| BUSGNT | 0x002 | Based on the XD[31:29] and XAS signals, the glue logic determines if a transfer to the XBD or to the XBISA register is taking place.<br>If the XHOLDA is asserted, the bus is granted to the DSP identified by the flip-flops that hold the bus master identification number.<br>The XHOLD signal cannot change in this state. |
| DSP_XBD_XFR | 0x004 | A transfer to the XBD register is taking place.<br>On every transfer to the XBD register the glue logic internal XBISA register is incremented by four (unless the XWAIT is asserted).<br>After completing a transfer to the XBD register, the state machine will either jump to the IDLE state (if the transfer was initiated by the host), or to the DRP_XHLD state state (if the transfer was initiated by the DSP).<br>If the XWAIT is asserted in this state, the glue logic will assert the XBOFF and de-assert the XRDY signal to the initiator DSP, and assert the XBLAST to the target DSP.<br>The XHOLD signal can not change in this state. |

**Table 2. Glue Logic State Machine Description (Continued)**

| State Name | State Code | Brief Description |
|---|---|---|
| DSP_XBISA_XFR | 0x008 | A transfer to the XBISA register is taking place. If the initiator is the DSP, the XBISA value is stored in the internal glue logic register.<br>The $\overline{\text{XWAIT}}$ signal is ignored during transfers to the XBISA since they are one word long bursts.<br>The XHOLD signal cannot change in this state. |
| DRP_XHLD | 0x010 | Drop the bus request to the host (XHOLD), and return the bus to the host (reset flip-flops that hold the bus master identification number) before going to the IDLE state. |
| WT_XHLD_DRP | 0x020 | Wait for the initiator DSP to drop its bus request to the glue logic. Bus request to the host (XHOLD) is kept asserted by the glue logic.<br>The XHOLD signal can not change in this state. |
| WT_SLV_CMPLT | 0x100 | Wait for the target DSP to complete a transfer of the last word. The transfer corrupts the XBISA register of the target DSP (data is being transferred while the initiator DSP is not ready). The bus request to the host (XHOLD) is kept asserted by the glue logic.<br>The XHOLD signal cannot change in this state. |
| WT_SET_XBISA1 | 0x040 | Address phase of a trasfer that reconfigures the target's XBISA register.<br>The XHOLD signal cannot change in this state. |
| WT_SET_XBISA2 | 0x080 | Data phase of a transfer that reconfigures the target's XBISA register. The content of the internal gue logic XBISA register is presented on the data bus.<br>The XHOLD signal cannot change in this state. |

# 4 Functional Verification

The interface was functionally verified using VHDL simulation (Synopsys SmartModel of the i80960Jf, and the VHDL model of two TMS320C6203 DSPs were instantiated in the test bench). Diagrams presented in Figure 3 through Figure 8 are outputs from the simulation. To maximally exercise the glue logic and make the $\overline{\text{XWAIT}}$ signal as frequent as possible, the clock ratio between the operating frequency of the TMS320C620–3 and the XCLKin frequency was set to 4.

The glue logic VHDL code was regressed in the test bench using 180 test cases. The regression tests used different source/destination memory types (internal data memory, internal program memory, EMIF SBSRAM and EMIF SDRAM), and tested the interface over a range of different burst lengths and different bus speeds.

The following figures illustrate functionality of the glue logic. The control signals with index number equal to one belong to DSP1 (XWAIT1, XHOLD1, XHOLDA1, XAS1, XCNTL1, XCS1, XRDY1, XBLAST1, XWNR1 and XBOFF1). The control signals with the index number equal to two belong to DSP2 (XWAIT2, XHOLD2, XHOLDA2, XAS2, XCNTL2, XCS2, XRDY2, XBLAST2, XWNR2 and XBOFF2). The control signals without indexes belong to i80960Jf. The bottom trace (arbiter_state) represents the glue logic state machine.

Figure 3 presents a real scenario where burst transfers of 24 words between DSP1 and DSP2 are interleaved with transfers between the host and DSP2.

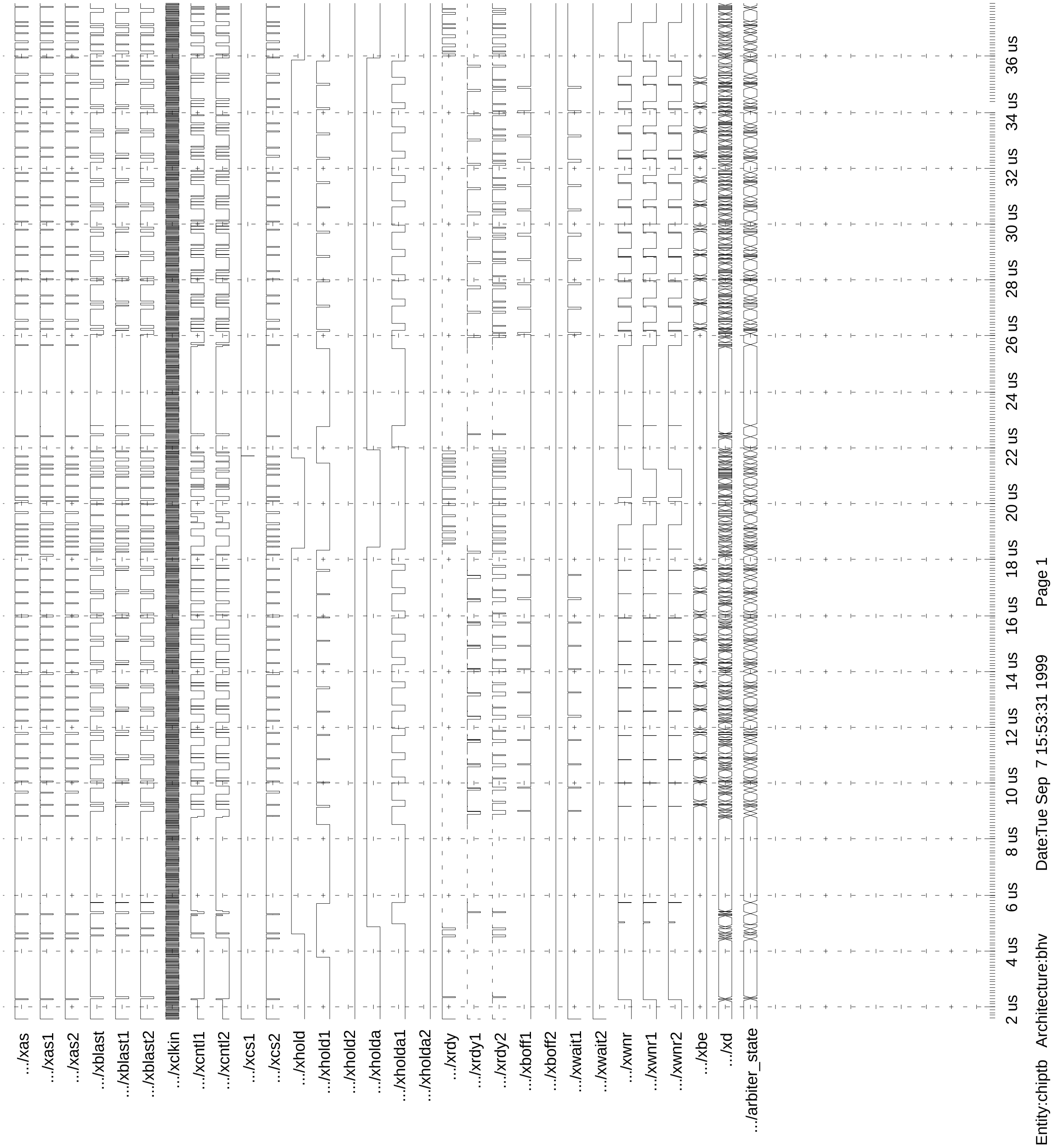

**Figure 3. Peer-to-Peer DSP Burst Transfers of 24 Words Interleaved With Host-to-DSP Transfers**

The host performs a transfer to the XBISA of DSP2 (the transfer takes place around 2∞s time stamp). DSP1 asserts its XHOLD1 signal around 4∞s time stamp, but the glue asserts the XHOLD to the host after the host completes the XBD transfer to DSP2 (around 5 µs time stamp). DSP1 gets the bus (XHOLDA asserted by the host) and performs a write burst to DSP2.

Between 18∞s and 22∞s the host is transferring data to DSP2. DSP1 requests the bus again around the 22∞s time stamp. After getting the bus, DSP1 sets the XBISA and reads a burst of data from DSP2.

Figure 4 illustrates a situation where DSP1 requests the bus after the host performed a transfer to the XBISA register of DSP2. The bus is granted to DSP1 after the host completes a transfer to the XBD register.

The host performs a write to the XBISA register at 2250 ns time stamp. Note that the state machine stays in **BUSGNT** (0x002) state after the XBISA transfer. The bus request of the DSP1 to the glue logic (XHOLD1) gets asserted at 3750 ns time stamp. The bus request to the host (XHOLD) is not changing at that time, since the glue logic expects the host to perform a transfer to the XBD register. Thus, DSP1 cannot start a transfer and corrupt the XBISA register previously set by the host. The host performs a one-word transfer to the XBD register at 4500 ns time stamp. Immediately after that the bus request to the host (XHOLD) gets asserted. The host performs one more transfer to the XBD register before granting the bus to the glue logic. The host grants the bus to the glue logic at 4875 ns time stamp (XHOLDA asserted). The glue logic grants the bus to DSP1 at 5 µs time stamp (XHOLDA1 asserted). After that DSP1 performs a transfer to the XBISA register of DSP2, and drops the bus request (XHOLD1) to the glue logic at 5750 ns time stamp. Note that the state machine stays in **BUSGNT** (0x002) state after the XBISA transfer. The bus is held by the glue logic (the XHOLD is kept high) since the glue logic expects DSP1 to perform a transfer to the XBD register (this way the host can not start a transfer and corrupt the XBISA register previously set by DSP1). DSP1 requests the bus from the glue logic at 8500 ns time stamp (XHOLD1 asserted). The glue logic immediately grants the bus to DSP1 (by asserting the XHOLDA1). DSP1 starts a transfer to the XBD.

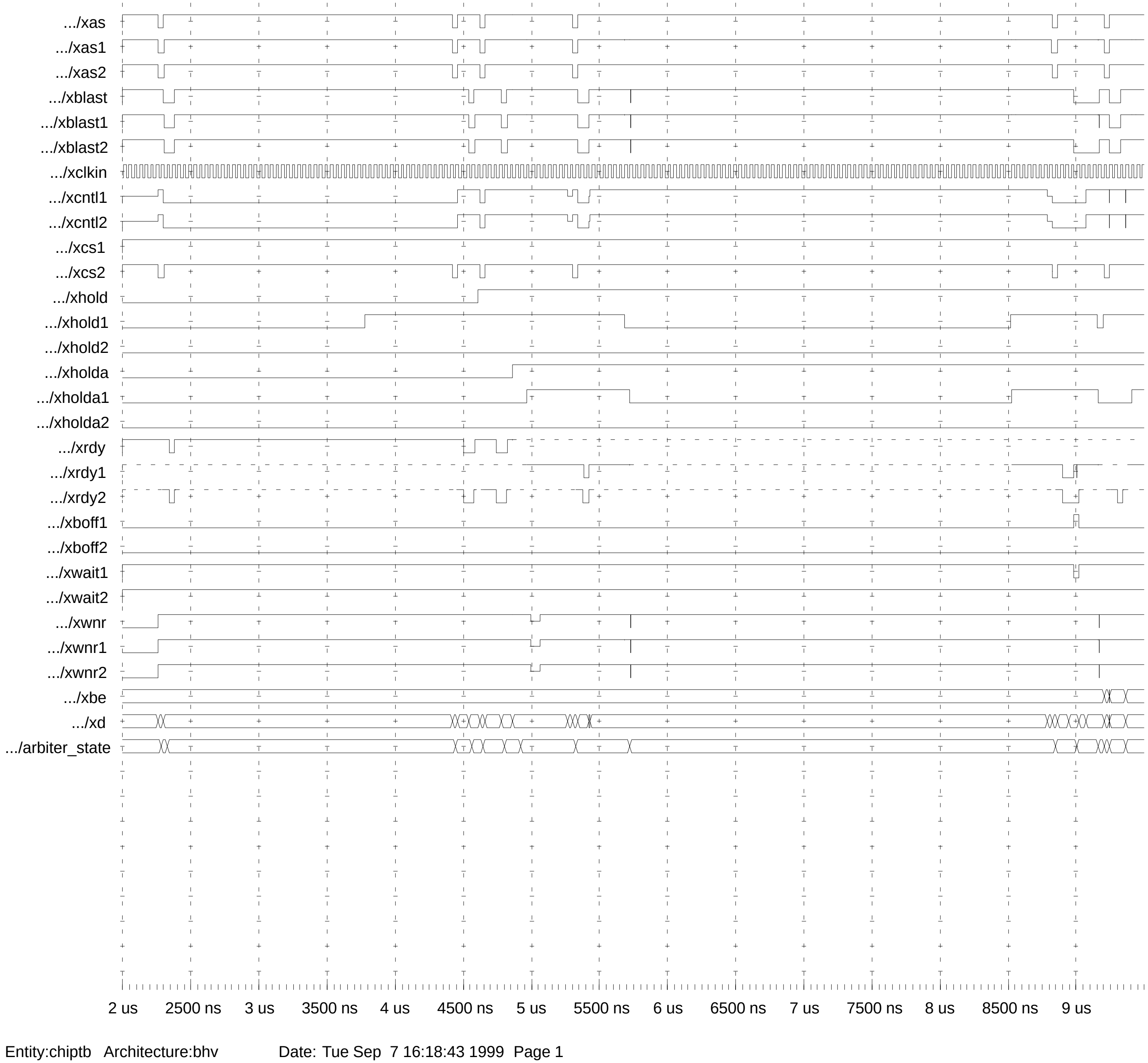

**Figure 4. DSP1 Requests the Bus After the Host Sets the XBISA Register of DSP2**

Figure 5 zooms into the region in which DSP1 requests the bus after the host performed a transfer to the XBISA register of DSP2. The finite state machine (the bottom trace marked as arbiter_state) is initially in the **IDLE** (0x001) state. The state machine (arbiter_state) will jump to the **DSP_XBISA_XFR** (0x008) when the host performs a transfer to the XBISA register of DSP2. After the transfer is completed, the state machine is in the **BUSGNT** (0x002) state. The state machine (arbiter_state) will jump to the **DSP_XBD_XFR** (0x004) when the host performs a transfer to the XBD register of DSP2. After the transfer is completed, the state machine goes back to the **IDLE** (0x001) state. The host performs an additional transfer to the XBD register of DSP2 before granting the bus to the glue logic (the state machine goes from the **IDLE** state to the **DSP_XBD_XFR**, and then back to the **IDLE**). After the host grants the bust to the glue logic (XHOLDA asserted), the state machine goes to the **BUSGNT** state. DSP1 first sets the XBISA register of DSP2 (during the transfer, the state machine is in the **DSP_XBISA_XFR**). During that transfer, a value of the XBISA register presented by DSP1 is latched in the internal glue logic register. After the transfer is completed, the state machine goes to the **BUSGNT** state.

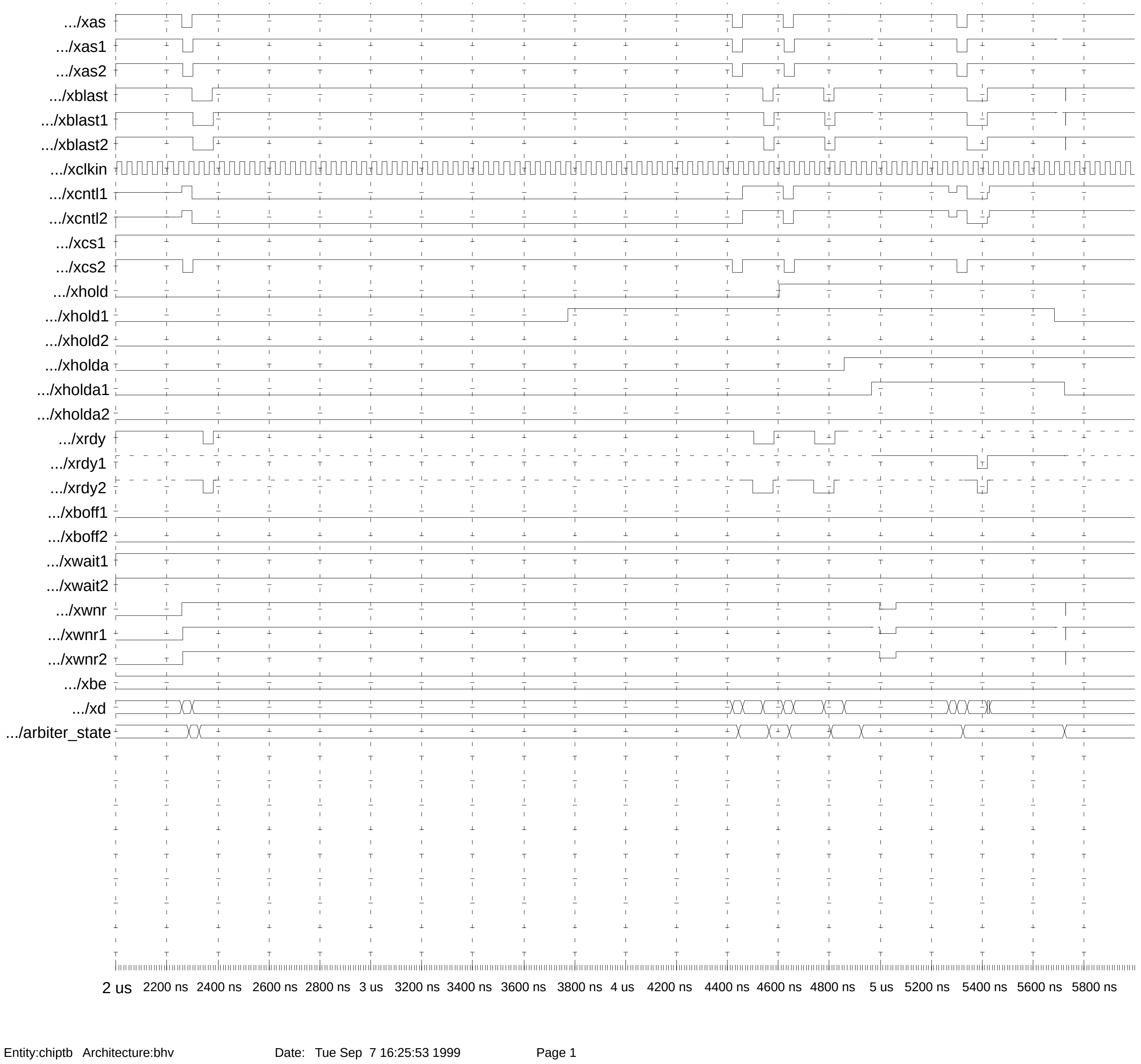

**Figure 5. DSP1 Requests the Bus After the Host Sets the XBISA Register of DSP2 (zoom in)**

Whenever the initiator DSP is not capable of sending/receiving a new word of data during the transfer, the initiator's $\overline{\text{XWAIT}}$ signal gets asserted. The target DSP assumes that the initiator is capable of providing/accepting a new word of data every clock cycle. In order to prevent data corruption every time the $\overline{\text{XWAIT}}$ gets asserted, the glue logic ends a transfer on the target's side (by asserting the target DSP blast signal).

Figure 6 illustrates a situation where the $\overline{\text{XWAIT1}}$ signal gets asserted during a write transfer to DSP2 initiated by DSP1.

The state machine is in the **DSP_XBD_XFR** state (0x004) when the $\overline{\text{XWAIT1}}$ signal gets asserted (at 8975 ns time stamp). The glue logic asserts the $\overline{\text{XBLAST2}}$, XBOFF1 and deasserts the $\overline{\text{XRDY1}}$. During this period, a word presented on the data bus by DSP1 is latched by DSP2. This transfer corrupts the XBISA register of DSP2 (the XBISA register of DSP2 is incremented but invalid data has been transferred – the $\overline{\text{XWAIT1}}$ is asserted). The glue logic internal XBISA register keeps track of the correct XBISA value (used to correct the DSP2 XBISA register). On the following rising edge of the XCLKIN clock, the state machine goes to the **WT_XHLD_DRP** state (0x020) and stays in that state until DSP1 responds to the back off. DSP1 backs off at 9150 ns time stamp (drops the XHOLD1). Sometime after that ,the glue logic deasserts the XHOLDA1. At this point, the glue logic owes the bus (the XHOLD is kept asserted). After DSP1 releases the bus, the glue logic goes to the **WT_SLV_CMPLT** (0x100) state. In this state glue logic checks if DSP2 completed the last transfer. To restart, interrupted transfer DSP1 asserts its bus request (XHOLD1) at 9200 ns time stamp. Before granting the bus back to DSP1 and allowing DSP1 to continue, the glue logic needs to adjust the target's XBISA register. To set the XBISA register of DSP2, the glue logic goes to the **WT_SET_XBISA1** (0x040) state, and **WT_SET_XBISA2** (0x080) state. The **WT_SET_XBISA1** (0x040) state performs an address phase of the XBISA transfer, and **WT_SET_XBISA2** (0x080) state performs an actual transfer from the internal glue logic XBISA register to the XBISA register of DSP2. After the DSP2 XBISA register is updated, the glue logic goes to the **BUSGNT** (0x002) state. The transfer interrupted by the back-off can be restarted.

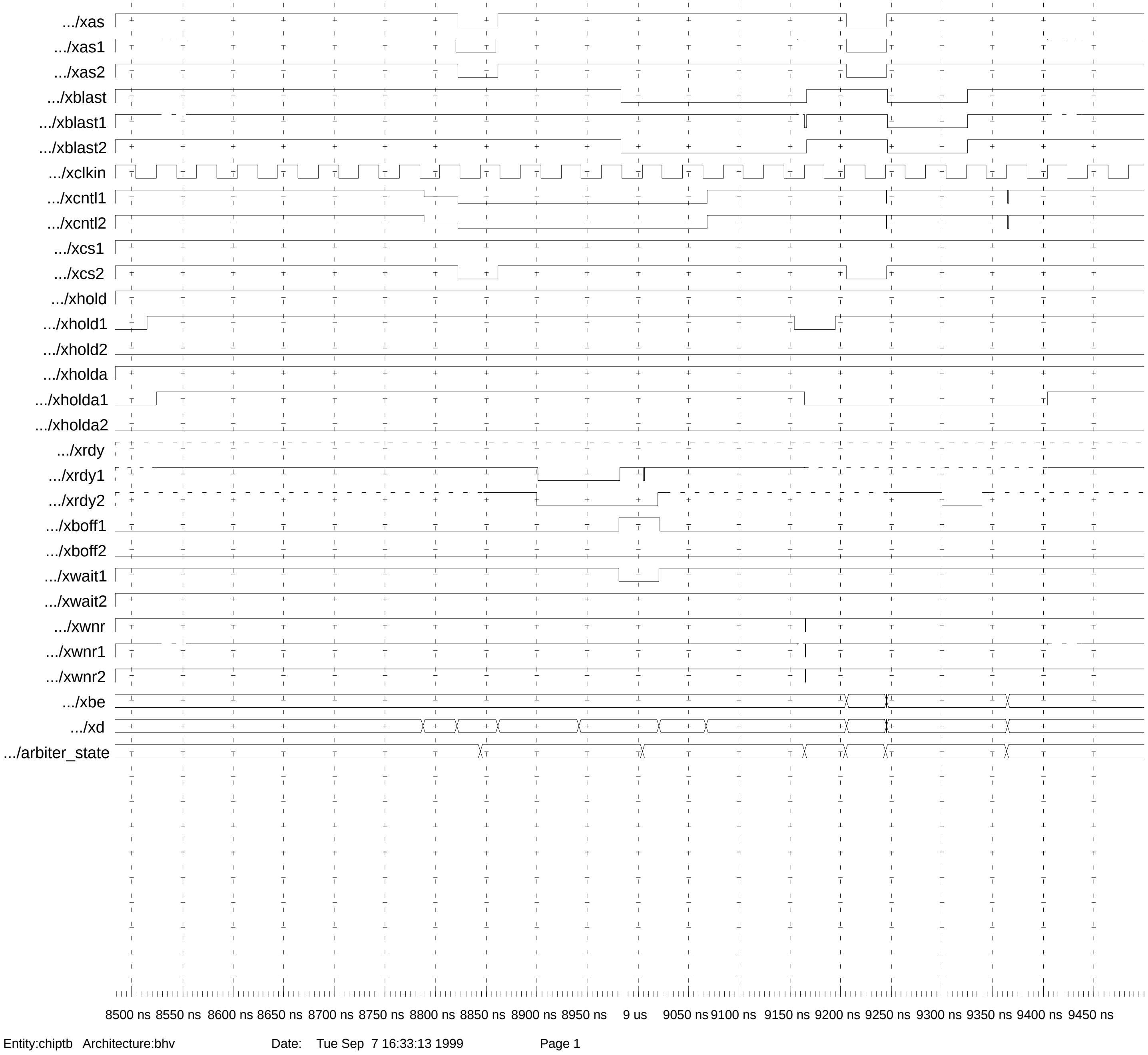

**Figure 6. Peer-to-Peer DSP Communication – $\overline{\text{XWAIT1}}$ Gets Asserted During a Write Transfer to DSP2, Initiated by DSP1**

Figure 7 illustrates a situation where DSP1 requests the bus to perform a read from DSP2. After the bus is granted, DSP1 first sets the XBISA register of DSP2, and then reads a burst of data from the XBD register.

The bus request of DSP1 to the glue logic (XHOLD1) gets asserted at 21500 ns time stamp. The bus request to the host (XHOLD) is not changing at that time since the glue logic waits the host to complete a transfer in progress to the XBD register. The bus request to the host (XHOLD) is asserted by the glue logic at 21600 ns time stamp. The host performs one more transfer to the XBD register before granting the bus to the glue logic. The host grants the bus to the glue logic at 21950 ns time stamp (XHOLDA asserted). The glue logic grants the bus to DSP1 at 22050 ns time stamp (XHOLDA1 asserted). DSP1 performs a write transfer to the XBISA register of DSP2, and drops the bus request (XHOLD1) to the glue logic at 22750 ns time stamp. Note that the state machine stays in **BUSGNT** (0x002) state after the XBISA transfer. The bus is held by the glue logic (XHOLD is kept high) since the glue logic expects DSP1 to perform a transfer to the XBD register (this way the host can not start a transfer and corrupt the XBISA register previously set by DSP1). DSP1 requests the bus from the glue logic at 25550 ns time stamp (XHOLD1 asserted). The glue logic immediately grants the bus to DSP1 (by asserting the XHOLDA1). DSP1 starts a transfer to the XBD.

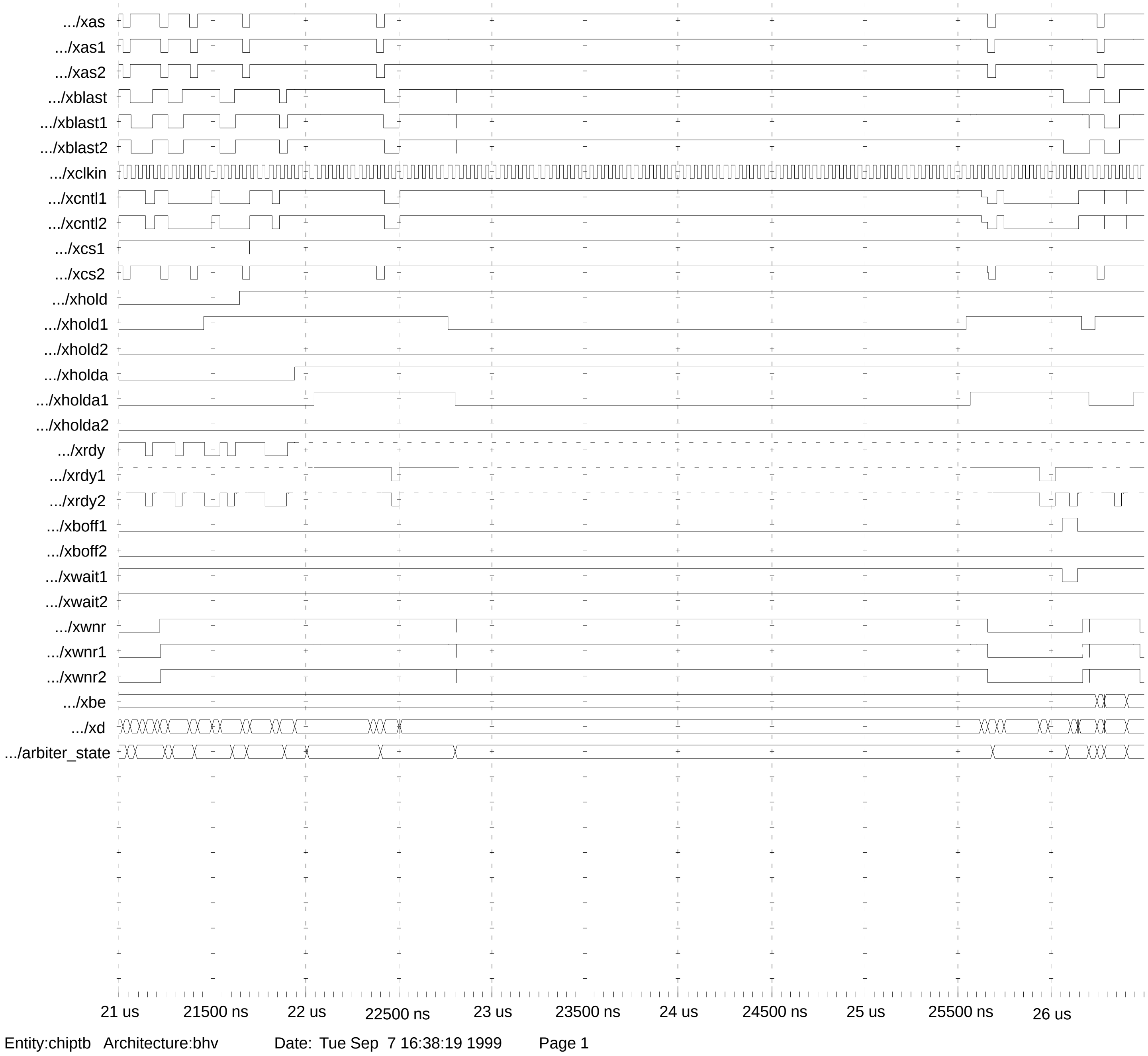

Figure 7. A Burst Read from DSP2 Initiated by DSP1 (the XBISA Register Write Followed by the XBD Register Read)

Figure 8 illustrates a situation where the $\overline{\text{XWAIT1}}$ signal gets asserted during a read from DSP2 by DSP1.

The state machine is in the **DSP_XBD_XFR** state (0x004) when the $\overline{\text{XWAIT1}}$ signal gets asserted (at 30460 ns time stamp). The glue logic asserts the $\overline{\text{XBLAST2}}$, XBOFF1 and deasserts the $\overline{\text{XRDY1}}$. DSP2 increments its XBISA register, assuming that the data has been accepted by DSP1 (DSP2 does not see that DSP1 is not ready to accept the data). This transfer corrupts the XBISA register of DSP2. The glue logic internal XBISA register keeps track of the correct XBISA value (used to correct the DSP2 XBISA register). On the following rising edge of the XCLKIN clock, the state machine goes to the **WT_XHLD_DRP** state (0x020), and stays in that state until DSP1 responds to the back-off. DSP1 backs off at 30560 ns time stamp (drops the XHOLD1). Sometime after that, the glue logic deasserts the XHOLDA1. At this point, the glue logic owes the bus (the XHOLD is kept asserted). After DSP1 releases the bus, the glue logic goes to the **WT_SLV_CMPLT** (0x100) state (in this state glue logic checks if DSP2 completed the last transfer). In order to restart interrupted transfer, DSP1 asserts its bus request (XHOLD1) at 30640 ns time stamp. Before granting the bus back to DSP1 and allowing DSP1 to continue, the glue logic needs to adjust the target's XBISA register. To set the XBISA register of DSP2 the glue logic goes to the **WT_SET_XBISA1** (0x040) state and **WT_SET_XBISA2** (0x080) state. The **WT_SET_XBISA1** (0x040) state performs an address phase of the XBISA transfer, and **WT_SET_XBISA2** (0x080) state performs an actual transfer from the internal glue logic XBISA register to the XBISA register of DSP2. After DSP2 XBISA register is updated, the glue logic goes to the **BUSGNT** (0x002) state. The transfer interrupted by the backoff can be restarted.

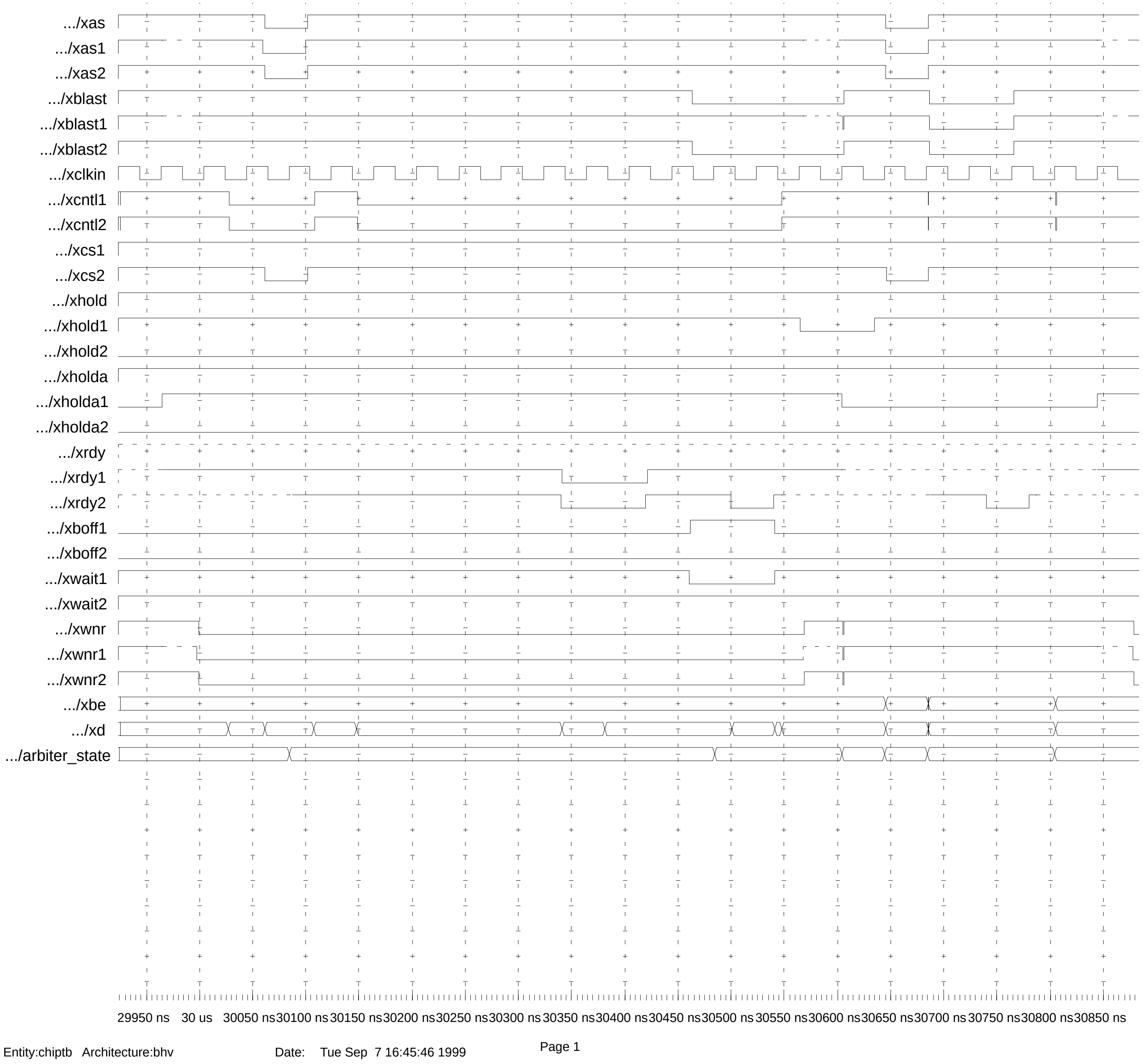

**Figure 8. Peer-to-Peer DSP Communication (the $\overline{\text{XWAIT1}}$ Gets Asserted During a Read From DSP2 Initiated by DSP1)**

## 5 Implementation Issues

The glue logic requires 63 pins and 49 flip-flops.

The glue logic VHDL was synthesized for the Xilinx CPLD XC9500XL-95288XL-TQ144-6. The timing numbers given in Appendix D are for the CPLD implementation.

## 6 Timing Requirements

The timing analysis presented is for a case when the glue logic VHDL is synthesized for the Xilinx CPLD XC9500XL-95288XL-TQ144-6. The list of glue logic timing parameters is given in Appendix D. The interface is operating at 30MHz (one XCLKIN period is 33 ns). Table 3 and Table 4 present timing requirements for host-to-DSP communication, while Table 5 and Table 6 present timing requirements for peer-to-peer DSP communication.

Table 3 presents timing requirements of the TMS320C6000 when the i80960JD is mastering the bus.

**Table 3. Timing Requirements for TMS320C6000 (i80960JD Expansion Bus Master)**

| I80960JD Symbol | C6000 Symbol | Parameter | I80960JD Min (ns) | C6000[†] Min (ns) |
|---|---|---|---|---|
| $T_{cyc} – t_{OV1} – t_{PD(XD\text{-}XCS)}$ | $t_{su(XCSV\text{-}XCKIH)}$ | Chip Select (XCS) valid before XCLKIN high | 12.7 | 3.5 |
| $t_{OV1} + t_{PD(XD\text{-}XCS)}$ | $t_{h(XCKIH\text{-}XCSV)}$ | Chip Select (XCS) valid after XCLKIN high | 9.3 | 2.8 |
| $T_{cyc} – t_{OV1} – t_{PD(XAS\text{-}XAS1)}$ | $t_{su(XASV\text{-}XCKIH)}$ | Address strobe (XAS) valid before XCLKIN high | 7.7 | 3.5 |
| $t_{OV1} + t_{PD(XAS\text{-}XAS1)}$ | $t_{h(XCKIH\text{-}XASV)}$ | Address strobe (XAS) valid after XCLKIN high | 14.3 | 2.8 |
| $T_{cyc} – t_{OV1}$ – <br> $– t_{PD(XBLST\text{-}XBLST1)}$ | $t_{su(XBLTV\text{-}XCKIH)}$ | Burst Last (XBLAST) valid before XCLKIN high | 13.5 | 3.5 |
| $t_{OV1} + t_{PD(XBLAST\text{-}XBLAST1)}$ | $t_{h(XCKIH\text{-}XBLTV)}$ | Burst Last (XBLAST) valid after XCLKIN high | 8.5 | 2.8 |
| $T_{cyc} – t_{OV1}$ | $t_{su(XD\text{-}XCKIH)}$ | Data (XD) valid before XCLKIN high (WRITE) | 19.5 | 3.5 |
| $t_{OV1}$ | $t_{h(XCKIH\text{-}XD)}$ | Data (XD) valid after XCLKIN high (WRITE) | 13.5 | 2.8 |
| $T_{cyc} – t_{OV1}$ | $t_{su(XBEV\text{-}XCKIH)}$ | Byte Enable (XBE[3:0]) valid before XCLKIN high | 19.5 | 3.5 |
| $t_{OV1}$ | $t_{h(XCKIH\text{-}XBEV)}$ | Byte Enable (XBE[3:0]) valid after XCLKIN high | 13.5 | 2.8 |
| $T_{cyc} – t_{OV1} – t_{PD(XWnR\text{-}XWnR1)}$ | $t_{su(XWR\text{-}XCKIH)}$ | Read/Write (XR/W) valid before XCLKIN high | 13.5 | 3.5 |
| $t_{OV1} + t_{PD(XWnR\text{-}XWnR1)}$ | $t_{h(XCKIH\text{-}XWR)}$ | Read/Write (XR/W) valid after XCLKIN high | 8.5 | 2.8 |

[†] C6000, in this case, refers to C6202, C6202B, C6203, C6204.

Table 4 presents timing requirements of the i80960JD.

### Table 4. i80960JD Timing Requirements (i80960JD Expansion Bus Master)

| I80960JD Symbol | C6000 Symbol | Parameter | I8096JD Min (ns) | C6000[†] Min (ns) |
|---|---|---|---|---|
| TIS2 | $T_{cyc} - t_{d(XCKIH-XRY)} -$<br>$- t_{PD(XRDY1-XRDY)}$ | Ready signal (RDYRCV) valid before XCLKIN high | 6.5 | 10.5 |
| TIH2 | $t_{d(XCKIH-XRY)} +$<br>$+ t_{PD(XRDY1-XRDY)}$ | Ready signal (RDYRCV) valid after XCLKIN high | 1 | 11 |
| TIS1 | $T_{cyc} - t_{d(XCKIH-XDV)}$ | Data (XD) valid before XCLKN high (READ) | 6 | 16.5 |
| TIH1 | $t_{d(XCKIH-XDIV)}$ | Data (XD) invalid after XCLKIN high (READ) | 1.5 | 5 |

[†] C6000, in this case, refers to C6202, C6202B, C6203, C6204.
[‡] P = 1/CPU clock frequency in ns. For example, when running parts at 250 MHz, use P = 4 ns.

Timing of the peer-to-peer DSP interface is described in Table 5 and Table 6. Table 5 refers to the target device timing requirements (C6000™), while Table 6 refers to the initiator device timing requirements (C6000). Peer-to-peer DSP interface is presented for the case of DSPs with similar timing parameters.

### Table 5. Target TMS320C6000[†] Timing Requirements

| Initiator DSP Symbol | Target DSP Symbol | Parameter | Initiator DSP Min (ns) | Target DSP Min (ns) |
|---|---|---|---|---|
| $T_{cyc} - t_{d(XCKIH-XDV)} -$<br>$- t_{PD(XD-XCS1)}$ | $t_{su(XCSV-XCKIH)}$ | Chip Select (XCS) valid before XCLKIN high | 9.7 | 3.5 |
| $t_{d(XCKIH-XDIV)} + t_{PD(XD-XCS1)}$ | $t_{h(XCKIH–XCSV)}$ | Chip Select (XCS) valid after XCLKIN high | 11.8 | 2.8 |
| $T_{cyc} - t_{d(XCKIH-XASV)} -$<br>$- t_{PD(XAS1-XAS2)}$ | $t_{su(XASV-XCKIH)}$ | Address strobe (XAS) valid before XCLKIN high | 5.5 | 3.5 |
| $t_{d(XCKIH-XASV)}+t_{PD(XAS1-XAS2)}$ | $t_{h(XCKIH-XASV)}$ | Address strobe (XAS) valid after XCLKIN high | 16 | 2.8 |
| $T_{cyc} - t_{d(XCKIH-XBLTV)} -$<br>$- t_{PD(XWAIT1-XBLST2)}$ | $t_{su(XBLTV-XCKIH)}$ | Burst Last (XBLAST) valid before XCLKIN high | 4.7 | 3.5 |
| $t_{d(XCKIH-XBLTV)} +$<br>$+ t_{PD(XBLST1-XBLST2)}$ | $t_{h(XCKIH-XBLTV)}$ | Burst Last (XBLAST) valid after XCLKIN high | 11 | 2.8 |
| $T_{cyc} - t_{d(XCKIH-XDV)}$ | $t_{su(XD-XCKIH)}$ | Data (XD) valid before XCLKIN high (WRITE) | 16.5 | 3.5 |
| $t_{d(XCKIH-XDIV)}$ | $t_{h(XCKIH-XD)}$ | Data (XD) valid after XCLKIN high (WRITE) | 5 | 2.8 |
| $T_{cyc} - t_{d(XCKIH-XBEV)}$ | $t_{su(XBEV-XCKIH)}$ | Byte Enable (XBE[3:0]) valid before XCLKIN high | 16.5 | 3.5 |
| $t_{d(XCKIH-XBEV)}$ | $t_{h(XCKIH-XBEV)}$ | Byte Enable (XBE[3:0]) valid after XCLKIN high | 5 | 2.8 |

**Table 5. Target TMS320C6000[†] Timing Requirements (Continued)**

| Initiator DSP Symbol | Target DSP Symbol | Parameter | Initiator DSP Min (ns) | Target DSP Min (ns) |
|---|---|---|---|---|
| $T_{cyc} - t_{d(XCKIH\text{-}XWRV)} -$<br>$- t_{PD(XWnR1\text{-}XWnR2)}$ | $t_{su(XWR\text{-}XCKIH)}$ | Read/Write (XR/W) valid before XCLKIN high | 10.5 | 3.5 |
| $t_{d(XCKIH\text{-}XWRV)} +$<br>$+ t_{PD(XWnR1\text{-}XWnR2)}$ | $t_{h(XCKIH\text{-}XWR)}$ | Read/Write (XR/W) valid after XCLKIN high | 11 | 2.8 |

[†] TMS320C6000, in this case refers to C6202, C6202B, C6203, C6204.

**Table 6. Initiator TMS320C6000[†] Timing Requirements**

| Initiator DSP Symbol | Target DSP Symbol | Parameter | Initiator DSP Min (ns) | Target DSP Min (ns) |
|---|---|---|---|---|
| $t_{su(XRY\text{-}XCKIH)}$ | $T_{cyc} - t_{d(XCKIH\text{-}XRY)} -$<br>$- t_{PD(XRDY1\text{-}XRDY2)}$ | Ready signal (XRDY) valid before XCLKIN high | 3.5 | 10.5 |
| $t_{h(XCKIH\text{-}XRY)}$ | $t_{d(XCKIH\text{-}XRY)} +$<br>$+ t_{PD(XRDY1\text{-}XRDY2)}$ | Ready signal (XRDY) valid after XCLKIN high | 2.8 | 11 |
| $t_{su(XDV\text{-}XCKIH)}$ | $T_{cyc} - t_{d(XCKIH\text{-}XDV)}$ | Data (XD) valid before XCLKN high (READ) | 3.5 | 16.5 |
| $t_{h(XCKIH\text{-}XDV)}$ | $t_{d(XCKIH\text{-}XDIV)}$ | Data (XD) invalid after XCLKIN high (READ) | 2.8 | 5 |
| $t_{su(XBFF\text{-}XCKIH)}$ | $T_{cyc} - t_{d(XCKIH\text{-}XWE)} -$<br>$- t_{PD(XWAIT1\text{-}XBOFF1)}$ | Backoff signal valid before XCLKIN high | 3.5 | 10.5 |
| $t_{h(XCKIH\ \text{-}XBFF)}$ | $t_{d(XCKIH\text{-}XWE)} +$<br>$+ t_{PD(XWAIT1\text{-}XBOFF1)}$ | Backoff signal invalid after XCLKIN high | 2.8 | 11 |

[†] TMS320C6000, in this case, refers to C6202, C6202B, C6203, C6204.

The timing tables above show that the timing parameters for all devices are met in the interface of i80960JD and the TMS320C6000. This interface is based on an i80960JD (local bus is running at 30 MHz) and a TMS320C6000 device at any frequency ranging from 100 MHz–300 MHz (TMS320C6203C frequency ranging from 250 MHz–400 MHz). The timing parameters for the glue logic are obtained by synthesizing the VHDL code for the Xilinx CPLD XC9500XL-95288XL-TQ144-6 (Appendix D).

## 7 Throughput Issues

Data throughput of the peer-to-peer DSP communication in this interface depends on several factors:

- Type of memory for source/destination of a transfer

  Higher throughput can be achieved if data is moved to/from the internal DSP memory. Slower rates are to be expected if data is moved to/from SBSRAM or SDRAM memory connected to the EMIF.

- Clock ratio between the DSP clock and the expansion bus clock

  For higher clock ratios (DSP CPU clock rate to XCLKIN clock), the $\overline{\text{XWAIT}}$ signal gets asserted less frequently.

- Frequency of the DSP clock

  Higher CPU frequency will result in higher throughput.
- Burst length

  Longer bursts result in higher throughput. This is especially important in peer-to-peer DSP communication. The i80960Jx/Kx burst length is limited to four words.

Every time the initiator DSP asserts the $\overline{\text{XWAIT}}$, additional cycles are added. These additional cycles are needed to update the XBISA register of the target (the XBISA register of the target DSP is corrupted every time the $\overline{\text{XWAIT}}$ is asserted). Therefore, to keep transfer as fast as possible condition that result in the $\overline{\text{XWAIT}}$ assertion should be avoided.

To achieve maximum performance one should:

- Use long bursts (over 16 words),
- Move data to/from the internal DSP data memory, and
- Keep the ratio between the DSP clock and the XCLKIN clock over or equal to eight.

Under these circumstances the $\overline{\text{XWAIT}}$ signal is extremely low-frequent, and the maximum data throughput that can be achieved using this interface is over 100Mbytes/s (assuming that the DSPs are running at 250 MHz).

For more detailed throughput considerations please refer to *TMS320C6000 Expansion Bus Performance* (SPRA643).

## 8 References

1. i960Jx Microprocessor User's Manual, Intel Corporation.
2. *TMS320C6000 Peripherals Reference Guide* (SPRU190).
3. *TMS320C6000 Expansion Bus Performance Host Interface Performance* (SPRA643).
4. 80960JA/JF/JD/JS/JC/JT 3.3 V Embedded 32-Bit Microprocessor, Intel Corporation.

# Appendix A Glue Logic VHDL Code

This section contains the VHDL code for the glue logic. The main part of the VHDL is the state machine. The machine has nine states, and the states are encoded using one hot encoding.

The VHDL code can be easily extended to support interface of more than two DSPs to the i80960 host.

```
--
--
--
--
--
-- TITLE   :  Glue Logic for the i80960Jf interface to two TMS320C6203 DSPs
-- COMPANY :  Texas Instruments Inc.
-- ENGR    :  Zoran Nikolic
-- DATE    :  09/03/99
--
--
--
--
-- DESCRIPTION:
--
--
-- The state machine takes care of the expansion bus arbitration and peer-to-peer
-- DSP communication.  The bus ownership can be changed only when in the IDLE
-- state.  If during a transfer to the XBD register, the XWAIT gets asserted following:
-- actions are taken: 1) the initiator DSP will is backed-off, 2) the XBLAST signal
-- of the target DSP is asserted 3) the target's XBISA register is corrected, and
-- 4) the bus is granted back to the initiator DSP to re-start interrupted transfer.
--
-- The XBISA value is stored and incremented by four during each transfer to the XBD
-- register(unless the XWAIT gets asserted).
--
-- Two mstrDSP flip-flops hold the identification number of current bus master (if
-- that is the host the mstrDSP is reset to zero).
--
--
--
--
--
--
--
--
--
--

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;
entity i960_glue is
    port (
        XCLKIN     : in STD_LOGIC;
        XAS        : inout STD_LOGIC;
        XAS1       : inout STD_LOGIC;
        XAS2       : inout STD_LOGIC;
        XD         : inout STD_LOGIC_VECTOR( 31 DOWNTO 0);
        XBE        : out STD_LOGIC_VECTOR( 3 DOWNTO 0);
        XCS1       : out STD_LOGIC;
        XCS2       : out STD_LOGIC;
        XWnR1      : inout STD_LOGIC;
        XWnR2      : inout STD_LOGIC;
        XWnR       : inout STD_LOGIC;
        XBOFF1     : out STD_LOGIC;
        XBOFF2     : out STD_LOGIC;
        XCNTL1     : out STD_LOGIC;
        XCNTL2     : out STD_LOGIC;
        XBLAST1    : inout STD_LOGIC;
        XBLAST2    : inout STD_LOGIC;
        XBLAST     : inout STD_LOGIC;
        XRDY1      : inout STD_LOGIC;
        XRDY2      : inout STD_LOGIC;
        XRDY       : inout STD_LOGIC;
        XHOLD      : out STD_LOGIC;
        XHOLD1     : in STD_LOGIC;
        XHOLD2     : in STD_LOGIC;
        XHOLDA     : in STD_LOGIC;
        XHOLDA1    : out STD_LOGIC;
        XHOLDA2    : out STD_LOGIC;
        XWAIT1     : in STD_LOGIC;
        XWAIT2     : in STD_LOGIC;
        RESET      : in STD_LOGIC
    );
end i960_glue;

architecture i960_glue_arch of i960_glue is

signal XCS1int, XCS2int : STD_LOGIC;
signal xrdy_out, xrdy_in, xrdy1_in, xrdy2_in :STD_LOGIC;
signal xblast_in, xblast_out, xblast_bckf, xblast_wait : STD_LOGIC;
signal XAS_in, XAS_out, xwnr_in, xwnr_out, XAS_xbisa : STD_LOGIC;
signal CS1int, CS2int : STD_LOGIC;
signal xholda1_int, xholda2_int, wait_ctrl, setXBISA: STD_LOGIC;

signal rBoff1, rBoff2 : STD_LOGIC;
signal xblast_w : STD_LOGIC;

signal XBISA: STD_LOGIC_VECTOR(31 DOWNTO 0);


signal wt_slave_rdy, xfr_cmplt: STD_LOGIC;

constant IDLE          : std_logic_vector(8 downto 0) :=”000000001”;
constant BUSGNT        : std_logic_vector(8 downto 0) :=”000000010”;
constant DSP_XBD_XFR   : std_logic_vector(8 downto 0) :=”000000100”;
constant DSP_XBISA_XFR : std_logic_vector(8 downto 0) :=”000001000”;
constant DRP_XHLD      : std_logic_vector(8 downto 0) :=”000010000”;
constant WT_XHLD_DRP   : std_logic_vector(8 downto 0) :=”000100000”;
constant WT_SET_XBISA1 : std_logic_vector(8 downto 0) :=”001000000”;
constant WT_SET_XBISA2 : std_logic_vector(8 downto 0) :=”010000000”;
constant WT_SLV_CMPLT  : std_logic_vector(8 downto 0) :=”100000000”;

signal Arbiter_State, Next_Arbiter_State: std_logic_vector(8 downto 0);
signal mstrDSP: integer range 0 to 3;
```

```
begin

   XHOLDA1 <= xholda1_int;
   XHOLDA2 <= xholda2_int;

   XCS1<=XCS1int;
   XCS2<=XCS2int;

   XBOFF1 <= (not XWAIT1);
   XBOFF2 <= (not XWAIT2);


   xblast_wait <= not((not XWAIT1 or not XWAIT2) or xblast_w);

   XBE <= (others => ’Z’) when setXBISA =’0’ else
                 ”0000”;

   XAS_out <=XAS_in when setXBISA = ’0’ else
             XAS_xbisa when setXBISA = ’1’ else
             ’Z’;
   xwnr_out <=xwnr_in when setXBISA = ’0’ else
             ’1’ when setXBISA = ’1’ else
             ’Z’;
-----------------------------------------------------------------

   XCNTL1 <= XD(29) when setXBISA = ’0’ else
             ’1’;

   XCNTL2 <= XD(29) when setXBISA = ’0’ else
             ’1’;

-----------------------------------------------------------------

   XCS1int <= ’0’     when XAS_in = ’0’ and XD(31)= ’0’ and XD(30)  =’1’ else
               CS1int when setXBISA =’1’ else
               ’1’;

   XCS2int <= ’0’     when XAS_in = ’0’ and XD(31)= ’1’ and XD(30)  =’0’ else
               CS2int when setXBISA =’1’ else
               ’1’;

-----------------------------------------------------------------

   XRDY1 <= xrdy_out when xholda1_int = ’1’ else
            ’Z’;

   xrdy1_in <= XRDY1 when xholda1_int = ’0’ else
               ’1’;
-----------------------------------------------------------------

   XRDY2 <= xrdy_out when xholda2_int = ’1’ else
            ’Z’;
   xrdy2_in <= XRDY2 when xholda2_int = ’0’ else
               ’1’;
-----------------------------------------------------------------

   XRDY <= xrdy_out when XHOLDA = ’0’ else
           ’Z’;

-----------------------------------------------------------------
```

```
-----------------------------------------------------------------
  xblast_in <= XBLAST1 when xholda1_int = ’1’ else
               XBLAST2 when xholda2_int = ’1’ else
               XBLAST when XHOLDA = ’0’ else
               ’1’;

   XBLAST1 <= xblast_out when xholda1_int = ’0’ else
              ’Z’;
-----------------------------------------------------------------

   XBLAST2 <= xblast_out when xholda2_int = ’0’ else
              ’Z’;
-----------------------------------------------------------------

   XBLAST <= xblast_out when XHOLDA = ’1’ else
             ’Z’;
-----------------------------------------------------------------
-----------------------------------------------------------------
   XAS_in <= XAS1 when xholda1_int = ’1’ else
             XAS2 when xholda2_int = ’1’ else
             XAS when XHOLDA = ’0’ else
             ’1’;

   XAS1 <= XAS_out when xholda1_int = ’0’ else
           ’Z’;

-----------------------------------------------------------------

   XAS2 <= XAS_out when xholda2_int = ’0’ else
           ’Z’;

-----------------------------------------------------------------

   XAS <= XAS_out when XHOLDA = ’1’ else
          ’Z’;
-----------------------------------------------------------------
-----------------------------------------------------------------
  xwnr_in <= XWnR1 when xholda1_int = ’1’ else
             XWnR2 when xholda2_int = ’1’ else
             XWnR when XHOLDA = ’0’ else
             ’1’;

   XWnR1 <= xwnr_out when xholda1_int = ’0’ else
            ’Z’;
-----------------------------------------------------------------

   XWnR2 <= xwnr_out when xholda2_int = ’0’ else
            ’Z’;

-----------------------------------------------------------------

   XWnR <= xwnr_out when XHOLDA = ’1’ else
           ’Z’;

-----------------------------------------------------------------
```

```
–––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

   xblast_out <= xblast_in when wait_ctrl = ’0’ and setXBISA =’0’ else
                 xblast_bckf when wait_ctrl = ’0’ and setXBISA =’1’ else
                 xblast_wait;

   xrdy_in <= xrdy1_in and xrdy2_in;

   xrdy_out <= xrdy_in when wait_ctrl = ’0’ else
               ’1’;

–––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

   XD <= XBISA when setXBISA = ’1’ else
         ”ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ”;
–––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

   state_comb: process (Arbiter_State, XWAIT1, XWAIT2, XBLAST, XRDY,
                        XHOLD1, XHOLD2, XHOLDA, XAS_in, XD,
                        xfr_cmplt, wt_slave_rdy, mstrDSP)
       begin
              wait_ctrl <= ’0’;
              setXBISA <= ’0’;
              xblast_bckf <= ’1’;
              CS1int <= ’1’;
              CS2int <= ’1’;
              XAS_xbisa <= ’1’;

              Next_Arbiter_State <= Arbiter_State;
              case Arbiter_State is

                          when IDLE =>

                               if XHOLDA =’1’ and (mstrDSP>0) then

                                      Next_Arbiter_State <= BUSGNT;
                               end if;

                               if XAS_in = ’0’ then
                                      case XD(31 downto 29) is
                                             when ”010” =>
                                                  Next_Arbiter_State <= DSP_XBD_XFR;
                                             when ”100” =>
                                                  Next_Arbiter_State <= DSP_XBD_XFR;
                                             when ”011” =>
                                                  Next_Arbiter_State <= DSP_XBISA_XFR;
                                                   when ”101” =>
                                                  Next_Arbiter_State <= DSP_XBISA_XFR;
                                             when others =>
                                                  Next_Arbiter_State <= BUSGNT;
                                       end case;

                                end if;
```

```
when BUSGNT =>


     if XAS_in = ’0’ then
            case XD(31 downto 29) is
                   when ”010” =>
                        Next_Arbiter_State <= DSP_XBD_XFR;
                   when ”100” =>
                        Next_Arbiter_State <= DSP_XBD_XFR;
                   when ”011” =>
                        Next_Arbiter_State <= DSP_XBISA_XFR;
                   when ”101” =>
                        Next_Arbiter_State <= DSP_XBISA_XFR;
                   when others =>
                        Next_Arbiter_State <= BUSGNT;
             end case;

     end if;

when DSP_XBD_XFR =>


     if (mstrDSP = 1 and XHOLD1 =’0’) or
        (mstrDSP = 2 and XHOLD2 =’0’) then

            Next_Arbiter_State <= DRP_XHLD;

     elsif mstrDSP = 0 then
            if XBLAST = ’0’ and XRDY = ’0’ then

                      Next_Arbiter_State <= IDLE;

            end if;
     end if;

     if XWAIT1 =’0’ or XWAIT2 =’0’ then

            Next_Arbiter_State <= WT_XHLD_DRP;
            wait_ctrl <= ’1’;

      end if;

when DSP_XBISA_XFR =>


     if (mstrDSP = 1 and XHOLD1 =’0’) or
        (mstrDSP = 2 and XHOLD2 =’0’) then

            Next_Arbiter_State <= BUSGNT;

     elsif mstrDSP = 0 then

            Next_Arbiter_State <= BUSGNT;

     end if;
```

```
                            when DRP_XHLD =>

                                Next_Arbiter_State <= IDLE;

                           when WT_XHLD_DRP =>

                                wait_ctrl <= ’1’;

                                case mstrDSP is
                                       when 1 => if XHOLD1 =’0’ then
                                                         Next_Arbiter_State <= WT_SLV_CMPLT;
                                                 end if;

                                      when 2 => if XHOLD2 =’0’ then
                                                         Next_Arbiter_State <= WT_SLV_CMPLT;
                                                 end if;
                                       when others =>
                                                 Next_Arbiter_State <= IDLE;
                                end case;

                           when WT_SLV_CMPLT =>

                                wait_ctrl <= ’1’;
                                if wt_slave_rdy = ’1’ then
                                       Next_Arbiter_State <= WT_SET_XBISA1;
                                       wait_ctrl <= ’0’;
                                end if;

                           when WT_SET_XBISA1 =>

                                setXBISA <= ’1’;

                                XAS_xbisa <= ’0’ ;

                                if mstrDSP = 1 then
                                        CS1int <= ’1’;
                                        CS2int <= ’0’;
                                elsif mstrDSP = 2 then
                                        CS1int <= ’0’;
                                        CS2int <= ’1’;
                                end if;

                                Next_Arbiter_State <= WT_SET_XBISA2;

                           when WT_SET_XBISA2 =>

                                setXBISA <= ’1’;

                                if xfr_cmplt = ’1’ then
                                        Next_Arbiter_State <= BUSGNT;
                                        xblast_bckf <= ’1’;
                                else
                                        xblast_bckf <= ’0’;
                                end if;

                           when others =>
                                 Next_Arbiter_State <= IDLE;
               end case;
        end process state_comb;
```

```
-----------------------------------------------------------------
   incrementXBISA: process(XCLKIN, xrdy_in, wait_ctrl, Arbiter_State, mstrDSP, RESET)
       begin

              if RESET = ’1’ then
                          XBISA <= (others => ’0’);

              elsif (XCLKIN’event and XCLKIN = ’1’) then
                    case Arbiter_State IS
                         when DSP_XBD_XFR => if xrdy_in =’0’ and wait_ctrl =’0’ and
                                                 (mstrDSP = 1 or  mstrDSP = 2 ) then
                                                         XBISA <= XBISA + 4;
                                             end if;

                         when DSP_XBISA_XFR => if xblast_in = ’0’ and xrdy_in =’0’ and
                                                 (mstrDSP = 1 or  mstrDSP = 2 ) then
                                                         XBISA <= XD;
                                               end if;
                         when others =>  XBISA <= XBISA;
                     end case;
              end if;

       end process incrementXBISA;
-----------------------------------------------------------------
   xbisaXferCmplt: process(XCLKIN, xrdy_in, Arbiter_State)
       begin

          if (XCLKIN’event and XCLKIN = ’1’) then
                case Arbiter_State is
                         when WT_SET_XBISA2 => if xrdy_in = ’0’ then
                                                     xfr_cmplt <= ’1’;
                                               end if;
                         when others =>  xfr_cmplt <= ’0’;
                end case;
          end if;

       end process xbisaXferCmplt;
-----------------------------------------------------------------
   XholdaSet: process(RESET, XCLKIN, XHOLD1, XHOLD2, XHOLDA, mstrDSP, Arbiter_State)
       begin
          if RESET = ’1’ then
                         xholda1_int <= ’0’;
                         xholda2_int <= ’0’;
          elsif XCLKIN’event and XCLKIN = ’1’ then

                case Arbiter_State is
                         when BUSGNT    =>  if mstrDSP = 1 and XHOLD1 = ’1’ and XHOLDA =’1’ then
                                                      xholda1_int <=’1’;
                                            elsif mstrDSP = 2 and XHOLD2 = ’1’ and XHOLDA =’1’ then
                                                      xholda2_int <=’1’;
                                            end if;

                         when others => if XHOLD1 =’0’ then
                                                      xholda1_int <=’0’;
                                        elsif XHOLD2 =’0’ then
                                                      xholda2_int <=’0’;
                                        end if;
                end case;
            end if;

       end process XholdaSet;
```

```
----------------------------------------------------------------

   BlstCntrlDrngWt : process (xrdy_in, XWAIT1, XWAIT2, XCLKIN, wait_ctrl, RESET)
       begin
          if RESET = ’1’ then
                xblast_w <= ’1’;
                wt_slave_rdy <= ’0’;
          elsif XCLKIN’event and XCLKIN =’1’  then
                xblast_w <= ((xblast_w and xrdy_in) or (not XWAIT1) or (not XWAIT2));
                wt_slave_rdy <= (wt_slave_rdy and wait_ctrl) or (not (xrdy_in or not wait_ctrl));

          end if;
   end process BlstCntrlDrngWt;
----------------------------------------------------------------

   BusArb : process (XHOLD1, XHOLD2, XCLKIN, Arbiter_State, RESET)
       begin
          if RESET = ’1’ then
               mstrDSP <= 0;
               XHOLD <=’1’; -- To request the bus from i960 when in reset.

          elsif XCLKIN’event and XCLKIN =’1’  then
               case Arbiter_State is
                     when IDLE =>
                                 if XHOLD1 = ’1’ or XHOLD2 = ’1’ then
                                       XHOLD <= ’1’;
                                 else
                                       XHOLD <= ’0’;
                                 end if;

                                 if XAS_in = ’1’ and (XHOLD1 = ’1’) and (XHOLDA = ’1’) then
                                       mstrDSP <= 1;
                                 elsif XAS_in = ’1’ and (XHOLD2 = ’1’) and (XHOLDA = ’1’) then
                                       mstrDSP <= 2;
                                 else
                                       mstrDSP <= 0;
                                 end if;

                     when DRP_XHLD =>
                                      mstrDSP <= 0;
                                      XHOLD <= ’0’;

                     when others =>
                                      mstrDSP <= mstrDSP;
              end case;

          end if;

    end process BusArb;
----------------------------------------------------------------

   state_clocked:process(XCLKIN, RESET)
       begin
             if RESET = ’1’ then
                   Arbiter_State <= IDLE;
             elsif XCLKIN’event and XCLKIN =’1’  then
                   Arbiter_State <= Next_Arbiter_State;
             end if;
       end process state_clocked;

end i960_glue_arch;
```

# Appendix B i80960JD Timing Requirements

### Table B–1. Intel 80960JD Timing Parameters

| Symbol | Characteristic | Min (ns) | Max (ns) |
|---|---|---|---|
| $t_{OV1}$ | Output valid delay, except ALE inactive and DT/$\overline{\text{R}}$ for 3.3 V input signals | 2.5 | 13.5 |
| $T_{OF}$ | Output float delay | 2.5 | 13.5 |
| $t_{IS1}$ | input setup to $\overline{\text{CLKIN}}$ AD[31:0] | 6 | |
| $t_{IH1}$ | Input Hold from $\overline{\text{CLKIN}}$ AD[31:0] | 1.5 | |
| $t_{IS2}$ | Input setup to CLKIN $\overline{\text{RDYRCV}}$ | 6.5 | |
| $t_{IH2}$ | Input hold from CLKIN $\overline{\text{RDYRCV}}$ | 1 | |
| TLX | Address valid to ALE Inactive | 10 | |
| TLXL | ALE width | 8 | |
| TLXA | Address hold from ALE inactive | 8 | |
| TDXD | DT/$\overline{\text{R}}$ valid to $\overline{\text{DEN}}$ active | 8 | |

The timing requirements are given here only for a quick reference. For detailed description, notes, and restrictions, please refer to the 80960JA/JF/JD/JS/JC/JT 3.3 V Embedded 32-Bit Microprocessor data sheet.

# Appendix C TMS320C6203 Timing Parameters

### Table C–1. TMS320C6000[†] Timing Parameters (External Device is a Master)

| Symbol | Characteristic | Min (ns) | Max (ns) |
|---|---|---|---|
| $t_{su(XCSV\text{-}XCKIH)}$ | Setup time, XCS valid before XCLKIN high | 3.5 | |
| $t_{h(XCKIH\text{-}XCSV)}$ | Hold time, XCS valid after XCLKIN high | 2.8 | |
| $t_{su(XASV\text{-}XCKIH)}$ | Setup time, XAS valid before XCLKIN high | 3.5 | |
| $t_{h(XCKIH\text{-}XASV)}$ | Hold time, XAS valid after XCLKIN high | 2.8 | |
| $t_{su(XCTL\text{-}XCKIH)}$ | Setup time, XCNTL valid before XCLKIN high | 3.5 | |
| $t_{h(XCKIH\text{-}XCTL)}$ | Hold time, XCNTL valid after XCLKIN high | 2.8 | |
| $t_{su(XWR\text{-}XCKIH)}$ | Setup time, XWR valid before XCLKIN high | 3.5 | |
| $t_{h(XCKIH\text{-}XWR)}$ | Hold time, XWR valid after XCLKIN high | 2.8 | |
| $t_{su(XBLTV\text{-}XCKIH)}$ | Setup time, XBLAST valid before XCLKIN high | 3.5 | |
| $t_{h(XCKIH\text{-}XBLTV)}$ | Hold time, XBLAST valid after XCLKIN high | 2.8 | |
| $t_{su(XBEV\text{-}XCKIH)}$ | Setup time, XBE valid before XCLKIN high | 3.5 | |
| $t_{h(XCKIH\text{-}XBEV)}$ | Hold time, XBE valid after XCLKIN high | 2.8 | |
| $t_{su(XD\text{-}XCKIH)}$ | Setup time, XD valid before XCLKIN high | 3.5 | |
| $t_{h(XCKIH\text{-}XD)}$ | Hold time, XD valid after XCLKIN high | 2.8 | |
| $t_{d(XCKIH\text{-}XDLZ)}$ | Delay time, XCLKIN high to XD low impedance | 0 | |
| $t_{d(XCKIH\text{-}XDV)}$ | Delay time, XCLKIN high to XD valid | | 16.5 |
| $t_{d(XCKIH\text{-}XDIV)}$ | Delay time, XCLKIN high to XD invalid | 5 | |
| $t_{d(XCKIH\text{-}XDHZ)}$ | Delay time, XCLKIN high XD high impedance | | 4P[‡] |
| $t_{d(XCKIH\text{-}XRY)}$ | Delay time, XCLKIN high to XRDY valid | 5 | 16.5 |

[†] TMS320C6000, in this case, referes to C6202, C6202B, C6203, and C6204.
[‡] P = 1/CPU clock frequency in ns. For example, when running parts at 250 MHz, use P = 4 ns.

### Table C–2. TMS320C6000[†] Timing Parameters (TMS320C6000 is a Master)

| Symbol | Characteristic | Min [ns] | Max [ns] |
|---|---|---|---|
| $t_{su(XDV\text{-}XCKIH)}$ | Setup time, XD valid before XCLKIN high | 43.5 | |
| $t_{h(XCKIH\text{-}XDV)}$ | Hold time, XD valid after XCLKIN high | 2.8 | |
| $t_{su(XRY\text{-}XCKIH)}$ | Setup time, XRDY valid before XCLKIN high | 3.5 | |
| $t_{h(XCKIH\text{-}XRY)}$ | Hold time, XRDY valid after XCLKIN high | 2.8 | |
| $t_{su(XBFF\text{-}XCKIH)}$ | Setup time, XBOFF valid before XCLKIN high | 3.5 | |

[†] TMS320C6000, in this case, refers to C6202, C6202B, C6203, and C6204.
[‡] P = 1/CPU clock frequency in ns. For example, when running parts at 250 MHz, use P = 4 ns.

### Table C–2. TMS320C6000[†] Timing Parameters (TMS320C6000 is a Master) (Continued)

| Symbol | Characteristic | Min [ns] | Max [ns] |
|---|---|---|---|
| $t_{h(XCKIH\text{-}XBFF)}$ | Hold time, XBOFF valid after XCLKIN high | 2.8 | |
| $t_{d(XCKIH\text{-}XAS)}$ | Delay time, XCLKIN high to XAS valid | 5 | 16.5 |
| $t_{d(XCKIH\text{-}XWR)}$ | Delay time, XCLKIN high to XWR valid | 5 | 16.5 |
| $t_{d(XCKIH\text{-}XBLTV)}$ | Delay time, XCLKIN high to XBLAST valid | 5 | 16.5 |
| $t_{d(XCKIH\text{-}XBEV)}$ | Delay time, XCLKIN high to XBE valid | 5 | 16.5 |
| $t_{d(XCKIH\text{-}XDLZ)}$ | Delay time, XCLKIN high to XD low impedance | 0 | |
| $t_{d(XCKIH\text{-}XDV)}$ | Delay time, XCLKIN high to XD valid | | 16.5 |
| $t_{d(XCKIH\text{-}XDIV)}$ | Delay time, XCLKIN high to XD invalid | 5 | |
| $t_{d(XCKIH\text{-}XDHZ)}$ | Delay time, XCLKIN high to XD high impedance | | 4P[‡] |
| $t_{d(XCKIH\text{-}XWTV)}$ | Delay time, XCLKIN high to XWE/XWAIT valid | 5 | 16.5 |

[†] TMS320C6000, in this case, refers to C6202, C6202B, C6203, and C6204.
[‡] P = 1/CPU clock frequency in ns. For example, when running parts at 250 MHz, use P = 4 ns.

The timing requirements are given here only for a quick reference. For detailed description notes and restrictions, please refer to the corresponding data sheet.

# Appendix D  Glue Logic Timing Requirements

```
Performance Summary Report
                           --------------------------

Design:     last
Device:     XC95288XL-6-TQ144
Program:    Timing Report Generator:  version M1.5.25
Date:       Wed Sep 08 19:21:17 1999

These results are based on advance timing information. Please contact your local Xilinx
Sales Representative for pricing and availability.
Performance Summary:

Pad to Pad (tPD)                          :         11.8ns (2 macrocell levels)
Pad ’XWAIT1’ to Pad ’XBLAST1’

Clock net ’XCLKIN’ path delays:

Clock Pad to Output Pad (tCO)             :         14.7ns (3 macrocell levels)
Clock Pad ’XCLKIN’ to Output Pad ’XD<2>’                                  (GCK)

Clock to Setup (tCYC)                     :         18.2ns (3 macrocell levels)
Clock to Q, net ’N1433.Q’ to DFF Setup(D) at ’Arbiter_State<2>.D’         (GCK)
Target FF drives output net ’Arbiter_State<2>’

Setup to Clock at the Pad (tSU)           :         15.7ns (2 macrocell levels)
Data signal ’XHOLDA’ to DFF D input Pin at ’Arbiter_State<2>.D’
Clock pad ’XCLKIN’                                                        (GCK)

                          Minimum Clock Period: 18.2ns
                     Maximum Internal Clock Speed: 54.9Mhz
                            (Limited by Cycle Time)

--------------------------------------------------------------------------------
                            Pad to Pad (tPD) (nsec)
\ From      X     X     X     X     X     X     X     X     X     X     X     X
 \          A     A     A     B     B     B     D     D     D     H     R     R
  \         S     S     S     L     L     L     <     <     <     O     D     D
   \              1     2     A     A     A     2     3     3     L     Y     Y
    \                         S     S     S     9     0     1     D     1     2
     \                        T     T     T     >     >     >     A
      \                             1     2
  To   \------------------------------------------------------------------------

XAS            11.0  11.0                                      11.8
XAS1     11.8        11.0                                      11.8
XAS2     11.8  11.0                                            11.8
XBLAST                            6.0   6.0                     6.0
XBLAST1                     6.0         6.0                     6.0
XBLAST2                     6.0   6.0                           6.0
XBOFF1
XBOFF2
XCNTL1                                        6.0
XCNTL2                                        6.0
XCS1      6.8   6.8   6.8                           6.8   6.8   6.8
XCS2      6.8   6.8   6.8                           6.8   6.8   6.8
XRDY                                                            4.5   6.0   6.0
XRDY1                                                                       6.0
```

```
XRDY2                                                                 6.0
XWnR                                                            6.0
XWnR1                                                           6.0
XWnR2                                                           6.0

--------------------------------------------------------------------------------
                            Pad to Pad (tPD) (nsec)
\ From      X     X     X     X     X
 \          W     W     W     W     W
  \         A     A     n     n     n
   \        I     I     R     R     R
    \       T     T           1     2
     \      1     2
      \
  To   \------------------------------

XAS
XAS1
XAS2
XBLAST   11.8  11.8
XBLAST1  11.8  11.8
XBLAST2  11.8  11.8
XBOFF1    6.0
XBOFF2          6.0
XCNTL1
XCNTL2
XCS1
XCS2
XRDY      6.8   6.8
XRDY1     6.8   6.8
XRDY2     6.8   6.8
XWnR                        6.0   6.0
XWnR1                 6.0         6.0
XWnR2                 6.0   6.0

--------------------------------------------------------------------------------
                      Clock Pad to Output Pad (tCO) (nsec)
\ From      X
 \          C
  \         L
   \        K
    \       I
     \      N
      \
  To   \------

XAS      14.3
XAS1     14.3
XAS2     14.3
XBE<0>   14.7
XBE<1>   14.7
XBE<2>   14.7
XBE<3>   14.7
XBLAST   14.3
XBLAST1  14.3
XBLAST2  14.3
XCNTL1    8.5
XCNTL2    8.5
XCS1      9.3
```

```
XCS2      9.3
XD<0>    14.7
XD<10>   14.7
XD<11>   14.7
XD<12>   14.7
XD<13>   14.7
XD<14>   14.7
XD<15>   14.7
XD<16>   14.7
XD<17>   14.7
XD<18>   14.7
XD<19>   14.7
XD<1>    14.7
XD<20>   14.7
XD<21>   14.7
XD<22>   14.7
XD<23>   14.7
XD<24>   14.7
XD<25>   14.7
XD<26>   14.7
XD<27>   14.7
XD<28>   14.7
XD<29>   14.7
XD<2>    14.7
XD<30>   14.7
XD<31>   14.7
XD<3>    14.7
XD<4>    14.7
XD<5>    14.7
XD<6>    14.7
XD<7>    14.7
XD<8>    14.7
XD<9>    14.7
XHOLD     4.3
XHOLDA1   4.3
XHOLDA2   4.3
XRDY      9.3
XRDY1    10.5
XRDY2    10.5
XWnR      9.3
XWnR1    10.5
XWnR2    10.5

--------------------------------------------------------------------------------
                       Setup to Clock at Pad (tSU) (nsec)

\ From      X
 \          C
  \         L
   \        K
    \       I
     \      N
      \
  To   \------

XAS      15.7
XAS1     14.9
XAS2     14.9
XBLAST   14.9
```

```
XBLAST1  13.3
XBLAST2  13.3
XD<0>     4.9
XD<10>    4.9
XD<11>    4.1
XD<12>    5.7
XD<13>    4.1
XD<14>    5.7
XD<15>    4.1
XD<16>    4.9
XD<17>    4.1
XD<18>    5.7
XD<19>    4.1
XD<1>     4.1
XD<20>    4.9
XD<21>    4.1
XD<22>    4.9
XD<23>    4.1
XD<24>    4.9
XD<25>    4.1
XD<26>    5.7
XD<27>    4.9
XD<28>    4.9
XD<29>    5.7
XD<2>     4.9
XD<30>    9.9
XD<31>    9.9
XD<3>     4.1
XD<4>     4.9
XD<5>     4.1
XD<6>     5.7
XD<7>     4.1
XD<8>     4.9
XD<9>     4.1
XHOLD1   14.9
XHOLD2   14.9
XHOLDA   15.7
XRDY     14.9
XRDY1    13.3
XRDY2     9.9
XWAIT1   14.9
XWAIT2   14.9

--------------------------------------------------------------------------------
                          Clock to Setup (tCYC) (nsec)
                                (Clock: XCLKIN)
\ From                 A     A     A     A     A     A     A     A     A     N
 \                     r     r     r     r     r     r     r     r     r     1
  \                    b     b     b     b     b     b     b     b     b     4
   \                   i     i     i     i     i     i     i     i     i     2
    \                  t     t     t     t     t     t     t     t     t     9
     \                 e     e     e     e     e     e     e     e     e     .
      \                r     r     r     r     r     r     r     r     r     Q
       \               _     _     _     _     _     _     _     _     _
        \              S     S     S     S     S     S     S     S     S
         \             t     t     t     t     t     t     t     t     t
          \            a     a     a     a     a     a     a     a     a
           \           t     t     t     t     t     t     t     t     t
            \          e     e     e     e     e     e     e     e     e
```

```
             \         <     <     <     <     <     <     <     <     <
              \        0     1     2     3     4     5     6     7     8
               \       >     >     >     >     >     >     >     >     >
                \      .     .     .     .     .     .     .     .     .
                 \     Q     Q     Q     Q     Q     Q     Q     Q     Q
  To              \------------------------------------------------------------

Arbiter_State<0>.D  12.4  12.4  16.6  16.6  16.6  16.6  16.6  16.6  16.6
Arbiter_State<1>.D   7.4   7.4   7.4   7.4   7.4   7.4   7.4   7.4   7.4
Arbiter_State<2>.D  13.2  13.2  17.4  17.4  17.4  17.4  17.4  17.4  17.4
Arbiter_State<3>.D   7.4   7.4   7.4   7.4   7.4   7.4   7.4   7.4   7.4
Arbiter_State<4>.D   6.6   6.6   6.6   6.6   6.6   6.6   6.6   6.6   6.6
Arbiter_State<5>.D   6.6   6.6   6.6   6.6   6.6   6.6   6.6   6.6   6.6
Arbiter_State<6>.D   7.4   7.4   7.4   7.4   7.4   7.4   7.4   7.4   7.4
Arbiter_State<7>.D   6.6   6.6   6.6   6.6   6.6   6.6   6.6   6.6   6.6
Arbiter_State<8>.D   6.6   6.6   6.6   6.6   6.6   6.6   6.6   6.6   6.6
N1429.D              7.4   7.4   7.4   7.4   7.4   7.4   7.4   7.4   7.4   7.4
N1433.D              6.6   6.6   6.6   6.6   6.6   6.6   6.6   6.6   6.6
N1434.D              7.4   7.4   7.4   7.4   7.4   7.4   7.4   7.4   7.4
XBISA<0>.D          13.2  13.2  13.2  13.2  13.2  13.2  13.2  13.2  13.2
XBISA<10>.D         12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4
XBISA<11>.D         12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4
XBISA<12>.D         12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4
XBISA<13>.D         12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4
XBISA<14>.D         12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4
XBISA<15>.D         12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4
XBISA<16>.D         12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4
XBISA<17>.D         12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4
XBISA<18>.D         12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4
XBISA<19>.D         12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4
XBISA<1>.D          12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4
XBISA<20>.D         12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4
XBISA<21>.D         12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4
XBISA<22>.D         12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4
XBISA<23>.D         12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4
XBISA<24>.D         12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4
XBISA<25>.D         12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4
XBISA<26>.D         12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4
XBISA<27>.D         15.0  15.0  15.0  15.0  15.0  15.0  15.0  15.0  15.0
XBISA<28>.D         15.8  15.8  15.8  15.8  15.8  15.8  15.8  15.8  15.8
XBISA<29>.D         15.8  15.8  15.8  15.8  15.8  15.8  15.8  15.8  15.8
XBISA<2>.D          12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4
XBISA<30>.D         15.8  15.8  15.8  15.8  15.8  15.8  15.8  15.8  15.8
XBISA<31>.D         15.8  15.8  15.8  15.8  15.8  15.8  15.8  15.8  15.8
XBISA<3>.D          12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4
XBISA<4>.D          12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4
XBISA<5>.D          12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4
XBISA<6>.D          12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4
XBISA<7>.D          12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4
XBISA<8>.D          12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4
XBISA<9>.D          12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4  12.4
mstrDSP<0>.D         7.4   7.4   7.4   7.4   7.4   7.4   7.4   7.4   7.4
mstrDSP<1>.D         7.4   7.4   7.4   7.4   7.4   7.4   7.4   7.4   7.4
wt_slave_rdy.D       7.4   7.4   7.4   7.4   7.4   7.4   7.4   7.4   7.4
xblast_w.D
xfr_cmplt.D          6.6   6.6   6.6   6.6   6.6   6.6   6.6   6.6   6.6
```

```
--------------------------------------------------------------------------------
                          Clock to Setup (tCYC) (nsec)
                                (Clock: XCLKIN)
\ From                 N     N     X     X     X     X     X     X     X     X
 \                     1     1     B     B     B     B     B     B     B     B
  \                    4     4     I     I     I     I     I     I     I     I
   \                   3     3     S     S     S     S     S     S     S     S
    \                  3     4     A     A     A     A     A     A     A     A
     \                 .     .     <     <     <     <     <     <     <     <
      \                Q     Q     0     1     1     1     1     1     1     1
       \                           >     0     1     2     3     4     5     6
        \                          .     >     >     >     >     >     >     >
         \                         Q     .     .     .     .     .     .     .
          \                              Q     Q     Q     Q     Q     Q     Q
           \
            \
             \
              \
               \
                \
                 \
  To              \------------------------------------------------------------

Arbiter_State<0>.D  17.4  17.4
Arbiter_State<1>.D  12.4  12.4
Arbiter_State<2>.D  18.2  18.2
Arbiter_State<3>.D   7.4   7.4
Arbiter_State<4>.D
Arbiter_State<5>.D
Arbiter_State<6>.D
Arbiter_State<7>.D
Arbiter_State<8>.D
N1429.D
N1433.D              6.6
N1434.D                    7.4
XBISA<0>.D          13.2  13.2   6.6
XBISA<10>.D         12.4  12.4         6.6
XBISA<11>.D         12.4  12.4         6.6   6.6
XBISA<12>.D         12.4  12.4        10.8  10.8   6.6
XBISA<13>.D         12.4  12.4         6.6   6.6   6.6   6.6
XBISA<14>.D         12.4  12.4        11.6   7.4   7.4   7.4   7.4
XBISA<15>.D         12.4  12.4        10.8   6.6   6.6   6.6   6.6   6.6
XBISA<16>.D         12.4  12.4        11.6  11.6  11.6  11.6  11.6   7.4   7.4
XBISA<17>.D         12.4  12.4        10.8   6.6   6.6   6.6   6.6   6.6   6.6
XBISA<18>.D         12.4  12.4        11.6  11.6  11.6  11.6  11.6   7.4   7.4
XBISA<19>.D         12.4  12.4        10.8  10.8  10.8  10.8  10.8  10.8  10.8
XBISA<1>.D          12.4  12.4
XBISA<20>.D         12.4  12.4        10.8  10.8  10.8  10.8  10.8  10.8  10.8
XBISA<21>.D         12.4  12.4        10.8  10.8  10.8  10.8  10.8  10.8  10.8
XBISA<22>.D         12.4  12.4        11.6  11.6  11.6  11.6  11.6  11.6  11.6
XBISA<23>.D         12.4  12.4        10.8  10.8  10.8  10.8  10.8  10.8  10.8
XBISA<24>.D         12.4  12.4        11.6  11.6  11.6  11.6  11.6  11.6  11.6
XBISA<25>.D         12.4  12.4        10.8  10.8  10.8  10.8  10.8  10.8  10.8
XBISA<26>.D         12.4  12.4        11.6  11.6  11.6  11.6  11.6  11.6  11.6
XBISA<27>.D         13.2  13.2        11.6  11.6  11.6  11.6  11.6  11.6  11.6
XBISA<28>.D         16.6  16.6        11.6  11.6  11.6  11.6  11.6  11.6  11.6
XBISA<29>.D         13.2  13.2        11.6  11.6  11.6  11.6  11.6  11.6  11.6
XBISA<2>.D          12.4  12.4
XBISA<30>.D         16.6  16.6        11.6  11.6  11.6  11.6  11.6  11.6  11.6
```

```
XBISA<31>.D         16.6  16.6        11.6  11.6  11.6  11.6  11.6  11.6  11.6
XBISA<3>.D          12.4  12.4
XBISA<4>.D          12.4  12.4
XBISA<5>.D          12.4  12.4
XBISA<6>.D          12.4  12.4
XBISA<7>.D          12.4  12.4
XBISA<8>.D          12.4  12.4
XBISA<9>.D          12.4  12.4
mstrDSP<0>.D         7.4   7.4
mstrDSP<1>.D         7.4   7.4
wt_slave_rdy.D       7.4   7.4
xblast_w.D           6.6   6.6
xfr_cmplt.D          6.6   6.6

--------------------------------------------------------------------------------
                          Clock to Setup (tCYC) (nsec)
                                (Clock: XCLKIN)
\ From                 X     X     X     X     X     X     X     X     X     X
 \                     B     B     B     B     B     B     B     B     B     B
  \                    I     I     I     I     I     I     I     I     I     I
   \                   S     S     S     S     S     S     S     S     S     S
    \                  A     A     A     A     A     A     A     A     A     A
     \                 <     <     <     <     <     <     <     <     <     <
      \                1     1     1     1     2     2     2     2     2     2
       \               7     8     9     >     0     1     2     3     4     5
        \              >     >     >     .     >     >     >     >     >     >
         \             .     .     .     Q     .     .     .     .     .     .
          \            Q     Q     Q           Q     Q     Q     Q     Q     Q
           \
            \
             \
              \
               \
                \
                 \
  To              \------------------------------------------------------------

Arbiter_State<0>.D
Arbiter_State<1>.D
Arbiter_State<2>.D
Arbiter_State<3>.D
Arbiter_State<4>.D
Arbiter_State<5>.D
Arbiter_State<6>.D
Arbiter_State<7>.D
Arbiter_State<8>.D
N1429.D
N1433.D
N1434.D
XBISA<0>.D
XBISA<10>.D
XBISA<11>.D
XBISA<12>.D
XBISA<13>.D
XBISA<14>.D
XBISA<15>.D
XBISA<16>.D
XBISA<17>.D          6.6
XBISA<18>.D          7.4   7.4
```

```
XBISA<19>.D         10.8  10.8   6.6
XBISA<1>.D                             6.6
XBISA<20>.D         10.8  10.8  10.8         6.6
XBISA<21>.D         10.8  10.8   6.6         6.6   6.6
XBISA<22>.D         11.6  11.6   7.4         7.4   7.4   7.4
XBISA<23>.D         10.8  10.8   6.6         6.6   6.6   6.6   6.6
XBISA<24>.D         11.6  11.6  11.6        11.6  11.6  11.6   7.4   7.4
XBISA<25>.D         10.8  10.8   6.6         6.6   6.6   6.6   6.6   6.6   6.6
XBISA<26>.D         11.6  11.6  11.6        11.6  11.6  11.6   7.4   7.4   7.4
XBISA<27>.D         11.6  11.6  11.6        11.6  11.6  11.6  11.6  11.6  11.6
XBISA<28>.D         11.6  11.6  11.6        11.6  11.6  11.6  11.6  11.6  11.6
XBISA<29>.D         11.6  11.6  11.6        11.6  11.6  11.6  11.6  11.6  11.6
XBISA<2>.D
XBISA<30>.D         11.6  11.6  11.6        11.6  11.6  11.6  11.6  11.6  11.6
XBISA<31>.D         11.6  11.6  11.6        11.6  11.6  11.6  11.6  11.6  11.6
XBISA<3>.D
XBISA<4>.D
XBISA<5>.D
XBISA<6>.D
XBISA<7>.D
XBISA<8>.D
XBISA<9>.D
mstrDSP<0>.D
mstrDSP<1>.D
wt_slave_rdy.D
xblast_w.D
xfr_cmplt.D

--------------------------------------------------------------------------------
                          Clock to Setup (tCYC) (nsec)
                                (Clock: XCLKIN)
\ From                 X     X     X     X     X     X     X     X     X     X
 \                     B     B     B     B     B     B     B     B     B     B
  \                    I     I     I     I     I     I     I     I     I     I
   \                   S     S     S     S     S     S     S     S     S     S
    \                  A     A     A     A     A     A     A     A     A     A
     \                 <     <     <     <     <     <     <     <     <     <
      \                2     2     2     2     2     3     3     3     4     5
       \               6     7     8     9     >     0     1     >     >     >
        \              >     >     >     >     .     >     >     .     .     .
         \             .     .     .     .     Q     .     .     Q     Q     Q
          \            Q     Q     Q     Q           Q     Q
           \
            \
             \
              \
               \
                \
                 \
  To              \------------------------------------------------------------

Arbiter_State<0>.D
Arbiter_State<1>.D
Arbiter_State<2>.D
Arbiter_State<3>.D
Arbiter_State<4>.D
Arbiter_State<5>.D
Arbiter_State<6>.D
Arbiter_State<7>.D
```

```
Arbiter_State<8>.D
N1429.D
N1433.D
N1434.D
XBISA<0>.D
XBISA<10>.D                                 10.8              10.8  10.8  10.8
XBISA<11>.D                                  6.6               6.6   6.6   6.6
XBISA<12>.D                                 10.8              10.8  10.8  10.8
XBISA<13>.D                                  6.6               6.6   6.6   6.6
XBISA<14>.D                                 11.6              11.6  11.6  11.6
XBISA<15>.D                                 10.8              10.8  10.8  10.8
XBISA<16>.D                                 11.6              11.6  11.6  11.6
XBISA<17>.D                                 10.8              10.8  10.8  10.8
XBISA<18>.D                                 11.6              11.6  11.6  11.6
XBISA<19>.D                                 10.8              10.8  10.8  10.8
XBISA<1>.D
XBISA<20>.D                                 10.8              10.8  10.8  10.8
XBISA<21>.D                                 10.8              10.8  10.8  10.8
XBISA<22>.D                                 11.6              11.6  11.6  11.6
XBISA<23>.D                                 10.8              10.8  10.8  10.8
XBISA<24>.D                                 11.6              11.6  11.6  11.6
XBISA<25>.D                                 10.8              10.8  10.8  10.8
XBISA<26>.D          7.4                    11.6              11.6  11.6  11.6
XBISA<27>.D         11.6   7.4              11.6              11.6  11.6  11.6
XBISA<28>.D         11.6   7.4   7.4        11.6              11.6  11.6  11.6
XBISA<29>.D         11.6   7.4   7.4   8.2  11.6              11.6  11.6  11.6
XBISA<2>.D                                   7.4
XBISA<30>.D         11.6  11.6  11.6  11.6  11.6   7.4        11.6  11.6  11.6
XBISA<31>.D         11.6  11.6  11.6  11.6  11.6  11.6   7.4  11.6  11.6  11.6
XBISA<3>.D                                   6.6               6.6
XBISA<4>.D                                   6.6               6.6   6.6
XBISA<5>.D                                   6.6               6.6   6.6   6.6
XBISA<6>.D                                   7.4               7.4   7.4   7.4
XBISA<7>.D                                   6.6               6.6   6.6   6.6
XBISA<8>.D                                  11.6              11.6  11.6  11.6
XBISA<9>.D                                   6.6               6.6   6.6   6.6
mstrDSP<0>.D
mstrDSP<1>.D
wt_slave_rdy.D
xblast_w.D
xfr_cmplt.D

--------------------------------------------------------------------------------
                          Clock to Setup (tCYC) (nsec)
                                (Clock: XCLKIN)
\ From                 X     X     X     X     m     m     w     x     x
 \                     B     B     B     B     s     s     t     b     f
  \                    I     I     I     I     t     t     _     l     r
   \                   S     S     S     S     r     r     s     a     _
    \                  A     A     A     A     D     D     l     s     c
     \                 <     <     <     <     S     S     a     t     m
      \                6     7     8     9     P     P     v     _     p
       \               >     >     >     >     <     <     e     w     l
        \              .     .     .     .     0     1     _     .     t
         \             Q     Q     Q     Q     >     >     r     Q     .
          \                                    .     .     d           Q
           \                                   Q     Q     y
            \                                                .
             \                                               Q
```

```
              \
               \
                \
                 \
  To              \------------------------------------------------------

Arbiter_State<0>.D                          16.6  16.6  12.4        12.4
Arbiter_State<1>.D                           7.4   7.4               6.6
Arbiter_State<2>.D                          17.4  17.4  13.2        13.2
Arbiter_State<3>.D                           6.6   6.6
Arbiter_State<4>.D                           6.6   6.6
Arbiter_State<5>.D                           6.6   6.6
Arbiter_State<6>.D                                       7.4
Arbiter_State<7>.D                                                   6.6
Arbiter_State<8>.D                           6.6   6.6   6.6
N1429.D
N1433.D                                      6.6   6.6
N1434.D                                      7.4   7.4
XBISA<0>.D                                  13.2  13.2
XBISA<10>.D         10.8  10.8  10.8  10.8  12.4  12.4
XBISA<11>.D          6.6   6.6   6.6   6.6  12.4  12.4
XBISA<12>.D         10.8  10.8  10.8  10.8  12.4  12.4
XBISA<13>.D          6.6   6.6   6.6   6.6  12.4  12.4
XBISA<14>.D         11.6  11.6  11.6  11.6  12.4  12.4
XBISA<15>.D         10.8  10.8  10.8  10.8  12.4  12.4
XBISA<16>.D         11.6  11.6  11.6  11.6  12.4  12.4
XBISA<17>.D         10.8  10.8  10.8  10.8  12.4  12.4
XBISA<18>.D         11.6  11.6  11.6  11.6  12.4  12.4
XBISA<19>.D         10.8  10.8  10.8  10.8  12.4  12.4
XBISA<1>.D                                  12.4  12.4
XBISA<20>.D         10.8  10.8  10.8  10.8  12.4  12.4
XBISA<21>.D         10.8  10.8  10.8  10.8  12.4  12.4
XBISA<22>.D         11.6  11.6  11.6  11.6  12.4  12.4
XBISA<23>.D         10.8  10.8  10.8  10.8  12.4  12.4
XBISA<24>.D         11.6  11.6  11.6  11.6  12.4  12.4
XBISA<25>.D         10.8  10.8  10.8  10.8  12.4  12.4
XBISA<26>.D         11.6  11.6  11.6  11.6  12.4  12.4
XBISA<27>.D         11.6  11.6  11.6  11.6  13.2  13.2
XBISA<28>.D         11.6  11.6  11.6  11.6  15.8  15.8
XBISA<29>.D         11.6  11.6  11.6  11.6  12.4  12.4
XBISA<2>.D                                  12.4  12.4
XBISA<30>.D         11.6  11.6  11.6  11.6  15.8  15.8
XBISA<31>.D         11.6  11.6  11.6  11.6  15.8  15.8
XBISA<3>.D                                  12.4  12.4
XBISA<4>.D                                  12.4  12.4
XBISA<5>.D                                  12.4  12.4
XBISA<6>.D           7.4                    12.4  12.4
XBISA<7>.D           6.6   6.6              12.4  12.4
XBISA<8>.D           7.4   7.4   7.4        12.4  12.4
XBISA<9>.D           6.6   6.6   6.6   6.6  12.4  12.4
mstrDSP<0>.D                                 7.4
mstrDSP<1>.D                                       7.4
wt_slave_rdy.D                                           7.4
xblast_w.D                                                     6.6
xfr_cmplt.D                                                          6.6
```

```
Path Type Definition:

Pad to Pad (tPD) –                        Reports pad to pad paths that start
                                          at input pads and end at output pads.
                                          Paths are not traced through
                                          registers.

Clock Pad to Output Pad (tCO) –           Reports paths that start at input
                                          pads trace through clock inputs of
                                          registers and end at output pads.
                                          Paths are not traced through PRE/CLR
                                          inputs of registers.

Setup to Clock at Pad (tSU) –             Reports external setup time of data
                                          to clock at pad. Data path starts at
                                          an input pad and end at register D/T
                                          input. Clock path starts at input pad
                                          and ends at the register clock input.
                                          Paths are not traced through
                                          registers.

Clock to Setup (tCYC) –                   Register to register cycle time.
                                          Include source register tCO and
                                          destination register tSU.
```

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265