PCI Arbiter Workarounds for the NS9775

Table of Contents

PCI Arbiter Workarounds for the NS9775

Overview

This application note describes:

- NS9775 PCI arbiter errata
- Software workaround for the errata
- Hardware workaround for the errata

Description of errata

When a PCI master negates its `REQ#` and `IRDY#` on the same clock cycle, as shown in Figure 1, and no other PCI master is requesting the bus, the PCI arbiter determines that this master is still requesting the bus and grants the bus to it.

Because the master does not take ownership of the bus within 16 PCI clocks, it is considered "broken" by the arbiter and taken out of service. As a result, the `PCIBRK_Mx` bit in the PCI Arbiter Interrupt Status register is set, indicating a "broken" master, and all further bus requests are ignored from this master until the `PCIEN_Mx` bit in the PCI Arbiter Configuration register is toggled from low to high by software.

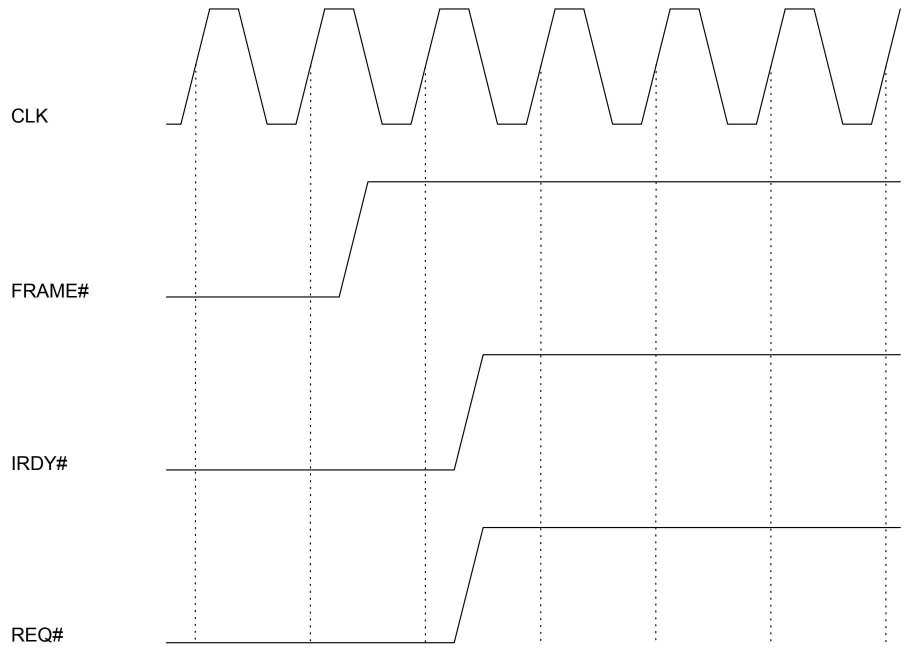


Figure 1: REQ# and IRDY# negated on same clock edge

Software workaround

Some PCI devices have a bit that controls when `REQ#` is negated. If this bit exists, this is the preferred workaround.

Otherwise, enable the `PCIBRK_Mx` interrupts for all masters present in the system by setting the associated `EN_PCIBRK_Mx` bits in the PCI Arbiter Interrupt Enable register. When a "broken" master interrupt is received, the master can be re-enabled by toggling the `PCIEN_Mx` bit for the broken master from low to high in the PCI Arbiter Configuration register.

Hardware workaround

Description

The workaround negates the `REQ#` signal when the master takes control of the bus so that the NS9775 does not see `REQ#` active when `IRDY#` is negated at the end of the cycle. This does not affect the master's ability to be granted the bus again at the end of the current PCI cycle because:

- The only way the current master can be granted the next bus cycle is if no other masters are requesting the bus.
- If no other masters are requesting the bus, the PCI arbiter parks ownership on the last master.

The hardware workaround is implemented as a 1-bit state machine that generates a gating signal that enables and disables the `PCI REQ#` from the master to the associated input of the NS9775. Figure 2 provides a state diagram.

When the state machine is in the `GATE_ON` state, the `REQ#` from the master is sent to the PCI arbiter in the NS9775. The state machine will not exit the `GATE_ON` state until this master has taken ownership of the PCI bus. Ownership of the bus is detected on the first clock cycle that this master drives `FRAME#.`

When the bus takes ownership, there is no reason to keep the `REQ#` from this master asserted with the bus-parking arbitration scheme described above, and it is forced inactive to the NS9775 when the state machine transitions to the `GATE_OFF` state. The state machine remains in the `GATE_OFF` state until the PCI bus goes idle, when it is safe to enable the `REQ#` again.

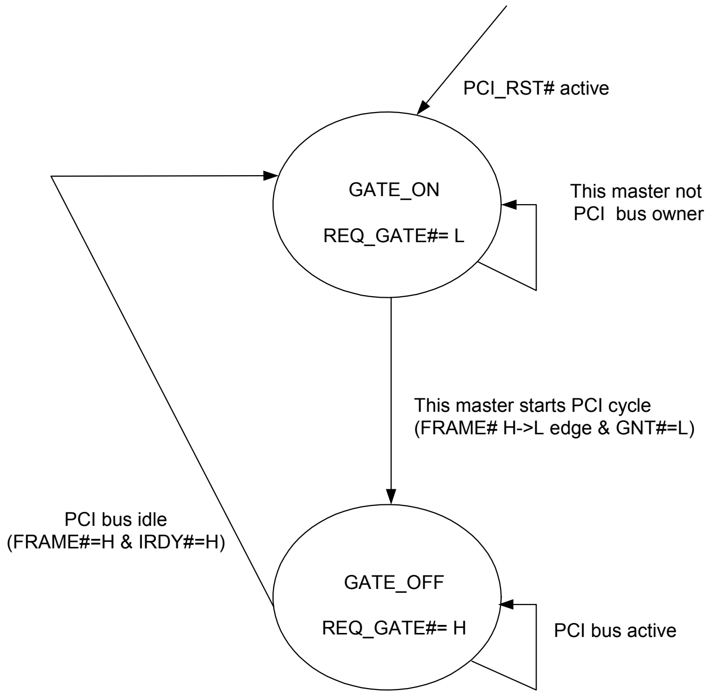

PCI_RST# active
GATE_ON
REQ_GATE#= L
This master not PCI bus owner
This master starts PCI cycle (FRAME# H->L edge & GNT#=L)
PCI bus idle (FRAME#=H & IRDY#=H)
GATE_OFF
REQ_GATE#= H
PCI bus active

Figure 2: Hardware workaround state diagram

Sample implementation

Figure 3 shows a sample implementation. This implementation uses only 2-input logic gates. Implementations that use gates with more inputs would result in fewer levels of logic. Also, all the combinatorial logic could be implemented using spare resources in an already existing PLD or FPGA in the target application. Timing must be met in whatever implementation is chosen.

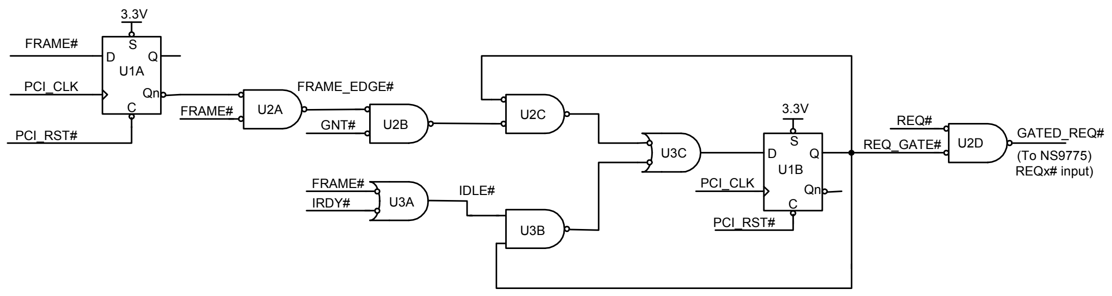
A sample timing analysis for this implementation using ALVC and LVC logic families is provided in the tables that follow to illustrate the timing parameters that must be considered in any implementation. In all these tables, the maximum PCI clock of 33Mhz is assumed.

Table 1 shows the worst-case setup timing path to the D-input of U1B.

Table 2 and Table 3 show the worst-case setup timing through both paths of U2D to any of the REQx# inputs of the NS9775.

Table 4 shows the worst-case hold timing path to any of the REQx# inputs of the NS9775 through U2D. Only the REQ_GATE# path is shown because the REQ# path meets the 0ns hold requirement by default because all PCI signals have a hold time of 2ns, per the PCI specification, and that compensates for the 2ns of allowable PCI clock skew. For the same reason, no table is provided for the hold time analysis for the D-input of U1B using any of the PCI signals.

Another path exists from `U1A-Qn`, but this is not a concern because the `PCI_CLOCK` skew between U1A and U1B is insignificant. Also the feedback path from `U1B-Q` is not a concern because it would satisfy the 0ns hold requirement of the 74LVC74 without any gate delays.



Notes:

U1A, U2A, and U3A are common to additional gating circuits.
FRAME_EDGE# detects the H->L edge of FRAME#.
U2 is a quad-OR gate; U3 is a quad-NAND gate.
PCI_CLK for U1A and U1B is from the same driver.

Figure 3: Hardware workaround sample implementation

This table describes the timing parameters:

Timing parameter	Time (ns)	Notes
PCI clock period	30	33Mhz max PCI clock
PCI_CLOCK valid to FRAME# valid (max)	(11)	PCI specification
Delay through U2A (max)	(2.8)	74ALVC32
Delay through U2B (max)	(2.8)	74ALVC32
Delay through U2C (max)	(2.8)	74ALVC32
Delay through U3C (max)	(3.0)	74ALVC00
U1B setup (max)	(2.0)	74LVC74 (ALVC not used because it is sole-sourced)
PCI clock skew (max)	(2.0)	PCI specification
Trace delay (max)	(2.0)	Very conservative estimate
Margin to U1B	1.6	

Table 1: Worst-case setup timing path to D-input of U1B

Timing parameter	Time (ns)	Notes
PCI clock period	30	33Mhz max PCI clock
PCI_CLK valid to REQ_GATE# (max)	(5.4)	74LVC74 (ALVC not used because it is sole-sourced)
Delay through U2D (max)	(2.8)	74ALVC32
REQx# setup requirement to NS9775 (max)	(5)	NS9775 timing specifications
PCI clock skew (max)	(2.0)	PCI spec
Trace delay (max)	(2.0)	Conservative estimate
Margin to NS9775	12.8	

Table 2: Worst-case setup timing to REQx# input of NS9775 using REQ_GATE#

Timing parameter	Time (ns)	Notes
PCI clock period	30	33Mhz max PCI clock
PCI_CLK valid to REQ# valid (max)	(12)	PCI specification
Delay through U2D (max)	(2.8)	74ALVC32
REQ# setup requirement to NS9775 (max)	(5)	NS9775 timing specifications
PCI clock skew (max)	(2.0)	PCI specification
Trace delay (max)	(2.0)	Conservative estimate
Margin to NS9775	6.2	

Table 3: Worst-case setup timing to REQx# input of NS9775 using REQ#

Timing parameter	Time (ns)	Notes
PCI_CLK valid to REQ_GATE#(min)	1.0	74LVC74 (ALVC not used because it is sole-sourced)
Delay through U2D (min)	1.0	74ALVC32
REQx# hold requirement to NS9775	0	NS9775 timing specifications
PCI clock skew (max)	(2.0)	PCI specification
Trace delay (min)	0	Conservative estimate
Margin to NS9775	0	

Table 4: Worst- case hold timing to REQx# input of NS9775 using REQ_GATE#