APPLICATION NOTE AN-91

Multi-GT Mode in the GT-6426xx

1. Introduction

The Marvell Technology GT-6426xx CPU interface supports multi-GT mode.

In multi-GT mode, multiple slave agents can be connected on the same CPU bus. This mode is useful for connecting on the CPU bus several GT-6426xx devices or user specific FPGA/ASIC chips (non-GT devices) sitting in parallel to the GT-6426xx device.

This application note outlines the system considerations when connecting multiple GT-6426xx devices and/or multiple slaves, non-GT devices on the CPU interface.

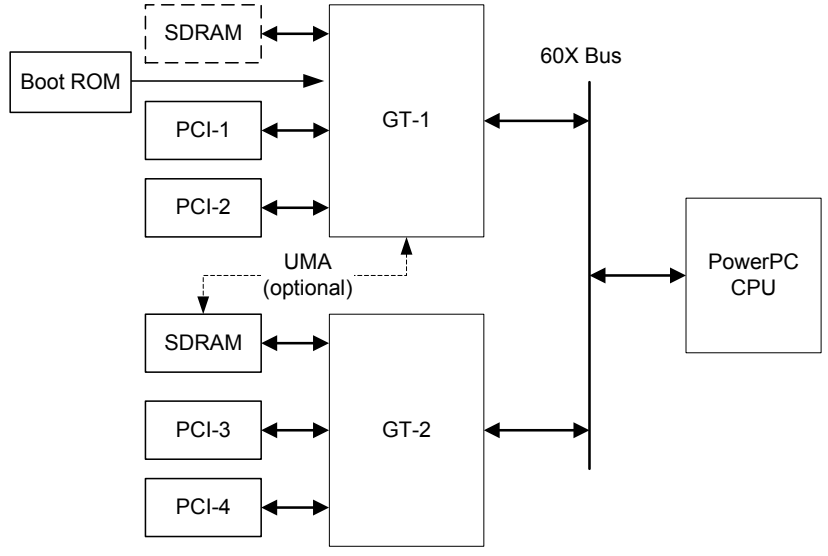
Notes

- For further information, refer to the multi-GT section in the appropriate data sheet's CPU interface section.
- In this document 'non-GT' refers to a user specific FPGA/ASIC device.

2. Multiple GT-6426xx Device Design Guidelines

This section outlines the considerations when connecting multiple GT-6426xx devices on the CPU 60x bus.

Figure 1: Multiple GT-6426xx Devices on the CPU Bus



In Figure 1, two GT-6426xx devices are connected to a single CPU. Each GT-6426xx has two PCI adapters on each of its PCI interfaces (total of 4 PCI adapters).

Data can be transferred between all of the adapters using one of the following methods:

- Use one SDRAM that is shared between the two GT-6426xxs using UMA (if single DRAM bandwidth is sufficient).
- Use the internal IDMA to transfer data between two separate SDRAMs, over the 60x bus.

Note

The GT-6426xx does not support MPX bus mode in multi-GT configuration.

2.1 Address Map Setting

Refer to the multi-GT section on the GT-6426xx data sheet.

2.2 AACK Delays

In multi-GT mode, the GT-6426xx always asserts the AACK* two cycles, or later, after TS*. The precise timing of the AACK* assertion depends on the condition of the GT-6426xx queues.

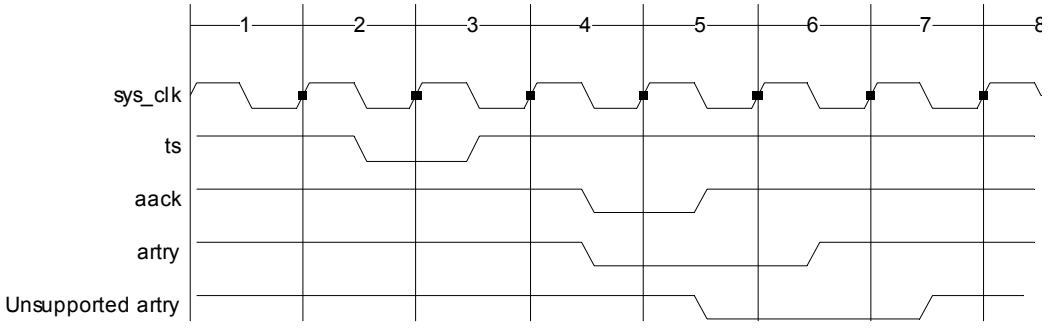
Note

In multi-GT mode, the following bits in the CPU configuration register (offset 0x000) are ignored: FastClk [23], AACK Delay [11], and AACK Delay_2 [25].

In this mode, it is impossible to interface with CPUs in which the ARTRY* window is delayed. For example, if a system is using MPC7450 CPU, the CPU core clock ratio must be selected to be higher than or equal to 1:5. Otherwise, the MPC7450 will assert ARTRY* three cycles after the TS*, the GT-6426xx will not sample it, and the system will hang.

Figure 2 is a waveform showing two cases of ARTRY* response.

Figure 2: ARTRY* Response



2.3 TA Delay Mode

In multi-GT mode, every GT-6426xx must be programmed to the same TA delay mode setting, either enabled or disabled.

The TA Delay mode is enabled when the CPU Configuration register's TADelay bit [15] is set to '1'. In this mode, the GT-6426xx earliest TA* assertion is two cycles after AACK*. Otherwise, with this bit set to '0', the GT-6426xx earliest TA* assertion is in the same cycle as AACK*.

2.4 Pipeline Mode

In multi-GT mode, there is support for Pipeline mode as long as all of the GT-6426xx devices are programed to the same pipeline mode, enabled or disabled.

The default value for the CPU Configuration register's Pipeline bit [13] is '0' (disabled). This means that the GT-6426xx uses a single queue entry and will not respond with AACK* to a new transaction before the previous transaction data phase completes.

Special care must be taken when programing the CPU Configuration register. Changing the Pipeline bit [13] setting to '1' (enable) appears to create a period in which one GT-6426xx is in Pipeline mode (using all of it's queue entries) and the other GT-6426xx is not in Pipeline mode. To prevent the system from hanging, follow the procedure described in Section 2.5.

2.5 Programming the CPU Configuration Register

Programing the CPU Configuration register (offset 0x000) must occur after completing the multiple GT-6426xx systems initialization sequence. In the datasheet, see the "Initializing a Multiple GT-642xx System" section.

To change the CPU Configuration register, the following steps are required.

Note

GT-1 and GT-2 are two GT-6426xx devices. GT-1 ID is '11' (the boot GT) and GT-2 ID is '00').

1. Read the GT-1 and GT-2 CPU Configuration registers. This "dummy" read, guarantees that all previous transactions in both CPU interface pipes are flushed.
2. Program the GT-2 register to its new value. Do not access GT-2 until step 4 is completed.
3. Program the GT-1 register to its new value.
4. Read polling of the GT-1 register until the new data is read.
5. Read polling of the GT-2 register until the new data is read.

Note

Program each of the GT-6426xx devices' CPU Configuration register once. Programing the register multiple times or not following the procedure described above (steps 1-5), may cause the system to hang.

2.6 60x Arbiter

The GT-6426xx internal, 60x arbiter supports up to two external CPU's (Masters) and its internal Master (usually used only for snoop transaction). If more masters are used, then an external arbiter must be used. Enable the external arbiter by strapping the AD[8] signal low at reset. For more details, refer to the data sheet's "Pins Sample configuration" section.

The following examples are some of the multi-GT optional configurations:

- Dual GT-6426xx in one system in which both GT-6426xx's internal Masters are in use plus one external CPU. Use the boot device's internal arbiter (GT-1 AD[8] strapped high, GT-2 AD[8] strapped low).
- Up to four GT-6426xx in one system in which one GT-6426xx, or none of the GT-6426xx internal Masters, are in use, plus one or two external CPU's. Use the boot device's internal arbiter (GT-1 AD[8] strapped high, GT-2/3/4 AD[8] strapped low).
- Up to four GT-6426xx in one system in which all of the GT-6426xx internal Masters are in use, plus one or more external CPU's. Use an external arbiter (GT-1/2/3/4 AD[8] strapped low).

Note

The external arbiter must assert DBG* at the earliest window allowed in the 60x bus protocol. The GT-6426xx, when using an external arbiter, does not qualify TA* assertion with DBG*. The GT-6426xx assumes DBG* is asserted to the 60x master in the cycle after TS* for idle state (no data transactions are pending), or the cycle after TA* assertion for data bus busy state (data transactions are pending). For full details, go to www.marvell.com for the EV-Board external arbiter PLD code.

2.7 Mismatch Errors

In cases where the CPU attempts to access an address that misses all address windows in all of the GT-6426xx devices, no device asserts AACK* and the system might hang.

If the NoMatchCntEn bit [8] in the boot GT-6426xx CPU Configuration register is set to '1', after a time out period defined in the same register's NoMatchCnt bits [7:0] and NoMatchCntExt bit [9], the boot GT-6426xx completes the transaction by asserting the right amount of TA*s (one or four). This transaction returns "dummy" data ('ffffffff'), parity error (if enabled), and interrupt (if enabled).

If the CPU interfaces with very "slow" devices, the NoMatchCnt value must be carefully defined. If the counter expires before the target device responds, there will be duplicate data tenures (TA*s) and the system hangs. If the access time to the slow device is greater than the maximum value allowed in the NoMatchCnt and NoMatchCntExt fields ('1ff'), NoMatchEn must be set to '0'.

Note

A mismatch error is not expected in normal operation. If it occurs, it indicates a software error occurred.

2.8 TA* and AACK* Hardware Connections

In multi-GT mode, the AACK* and TA* signals function as sustained tri-state outputs requiring pull-up resistors.

All TA* outputs from the GT-6426xx devices must be tied together to drive the CPU TA* input. All AACK* outputs from the GT-6426xx devices must be tied together to drive the CPU AACK* input.

AACK* and TA* are only driven by the target GT-6426xx or by the boot GT-6426xx in the case of mismatch address. After the last TA* and AACK* assertion, the target device drives these signals high for another half cycle and then tri-states them.

2.9 Interrupts

Each GT-6426xx has one CPU interrupt (CPUInt*).

Since the CPU has only one interrupt input, connect the boot GT-6426xx CPUInt* to the CPU input and the CPUInt* signals of the other devices to the boot GT-6426xx MPP inputs.

Once an interrupt event occurs, the interrupt handler first reads the boot GT-6426xx Main Cause register to identify the interrupt source. If the interrupt source is in the local device, the handler reads the local unit's Main Cause register to identify the exact cause for the interrupt. However, if the interrupt source is the GPP input, the interrupt handler reads the GPP Interrupt Cause register to identify which interrupt pin was asserted. The interrupt handler then reads the exact interrupted pin and the appropriate GT-6426xx device Main Cause register to identify the source of the interrupt.

2.10 AC timing parameters

The GT-6426xx AC timing values are specified for 10-20 pf load and not for higher loads, e.g., more than devices. Any multi-GT design must simulate the board traces using the GT-6426xx IBIS models.

In multi-GT mode, the following signals require a longer setup time: TS*, TT[0:4], ARTRY*, AACK* & TA*. This is why the maximum frequency in this mode is only 100-110 MHz instead of 133 MHz. For more details, refer to the data sheet and the latest Documentation Update and Changes (DCU) document.

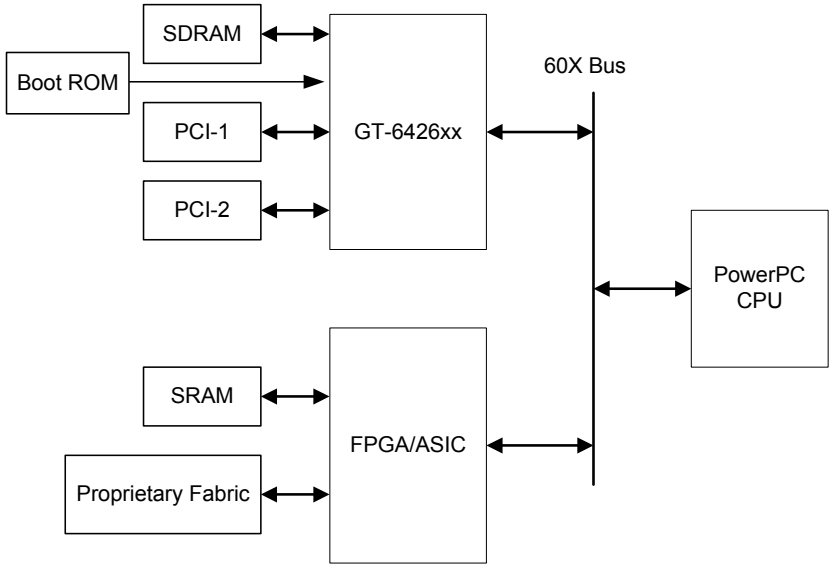
3. User Specific 60x FGPA/ASIC Design Guidelines

This section outlines the considerations and guidelines for a non-GT devices (CPLD/FPGA) connected on the CPU bus.

Note

In this case, the GT-6426xx device or devices must be configured to Multi-GT mode. All the restrictions described in Section 2. Multiple GT-6426xx Device Design Guidelines still apply to all the GT-6426xx and the non-GT devices.

Figure 3: Multiple GT-6426xx and Non-GT Devices on the CPU Bus



In this example, a single CPU is connected on the 60x bus to a GT-6426xx and a user defined FPGA/ASIC (non-GT device). The FPGA implements both master and slave on the 60x bus. It is used to interface proprietary fabric (cell or packet based) and has local SRAM for buffering. The CPU manages the FPGA (read/write configuration registers). The FPGA also has its own DMA to transfer data/descriptors between its local SRAM and the main SDRAM.

As an alternative, the FPGA is a slave only and uses the GT-6426xx IDMA to transfer data from the FPGA SRAM to the main SDRAM.

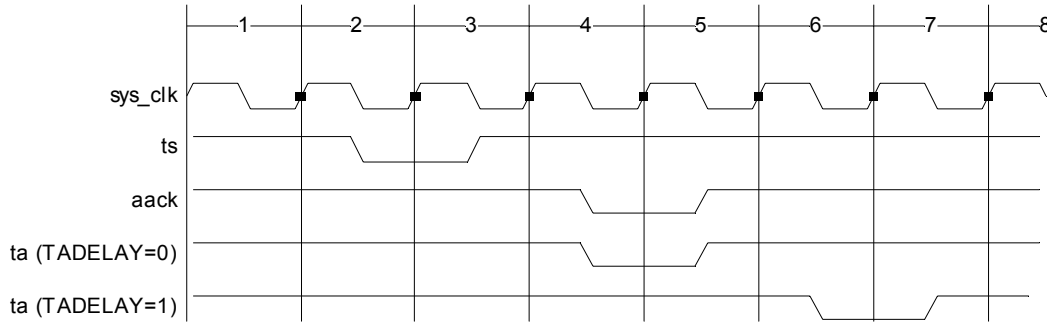
3.1 AACK* and TA* Delays Requirements

The non-GT device must assert AACK* two cycles, or later, after TS*.

The non-GT device must assert TA* according to the TA delay mode programmed in the GT-6426xx devices (the CPU Configuration register's TADelay bit [15]).

If the GT-6426xx devices are programmed to assert TA* on the same cycle as AACK (TADelay is set to '0'), the non-GT device must assert TA* in the same cycle as AACK* or later. If the GT-6426xx devices are programed to assert TA* two cycles after AACK* (TADelay is set to '1'), the non-GT device must assert TA* two cycles after AACK* or later. Figure 4 shows the earliest TA* assertion in both TA delay modes.

Figure 4: AACK* and TA* Delays Response



3.2 Pipeline and Queue Depth Requirements

In multi-GT mode with pipeline enabled, each GT-6426xx can queue up to 14 transactions and "observe" up to 16 transactions (including its own). With pipeline disabled, it can queue or observe only one transaction. For this reason, special care should be taken to maintain 60x bus ordering. For example, if there are two transactions targeted to the GT-6426xx followed by a transaction targeted to the non-GT device, the non-GT device must wait until the prior two transactions complete their data tenure before completing its own transaction. In general, the non-GT device must have the same queuing capabilities as the GT-6426xx, even though, in many cases, such a "deep" queue is not necessary. For more details on implementing the 60x Slave, in the different PIPELINE options, see Section 3.6 60x Bus Slave Non-GT Design Guidelines.

Note

"Observe up to 16 transactions" means that the GT-6426xx device counts the number of times TA* was asserted by other GT-6426xx or non-GT devices. The device uses this time/cycle to estimate when it can return its own TA*s with respect to 60X bus protocol.

3.3 Mismatch Errors

The same restriction, as described in Section 2.7 Mismatch Errors, applies to a non-GT device, when it interfaces with very "slow" devices on its "side". If the boot GT-6426xx NoMatchCnt counter expires before the non-GT target device responds, duplicate data tenure (TA*) occurs and the system hangs. If the slower device access time is higher than the maximum value, i.e, NoMatchCnt='ff' and NoMatchCntExt=1, NoMatchCntEn bit [8] must be set to '0'.

Note

Instead of using the GT-6426xx mismatch mechanism, it is possible that the non-GT device will implement a mismatch time-out mechanism. In this case, set NoMatchCntEn to ’0’.

3.4 TA* and AACK* Hardware Requirements

The non-GT device must implement the AACK* and TA* signals to function as sustained tri-state outputs.

After the last TA* and AACK* assertion, the non-GT device must drive these signals high for another half cycle and then tri-states them.

All TA* and AACK* outputs from the GT-6426xx and the non-GT devices must be tied together to drive the CPU TA* and AACK* inputs.

3.5 60x Bus Master Non-GT Design Requirements

The non-GT master must have a similar behavior to the CPU connected on the bus. If non-GT master implements a retry mechanism like a CPU, it must assert ARTRY* the second cycle after TS* and de-assert ARTRY* the second cycle after AACK*.

3.6 60x Bus Slave Non-GT Design Guidelines

The following sections contain recommendations and examples showing how to design a 60x bus slave with minimum logic.

Note

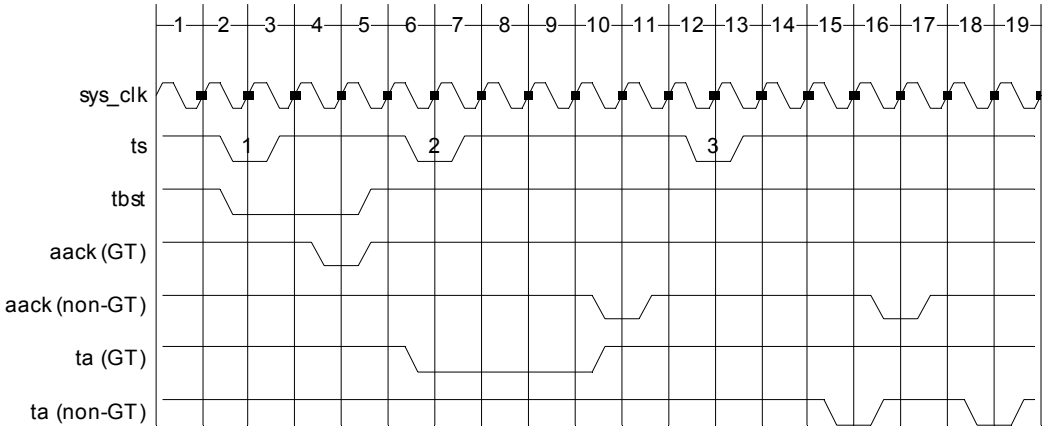
The examples do not take into account the case of ARTRY* assertion. In this case, the transaction is canceled and the counters are disabled.

3.6.1 GT-6426xx and non-GT Device in Non-Pipeline Mode

The non-GT device must only implement one sampling stage to be able to save one address and attribute, handle it or observe it. If an address is targeted to the GT-6426xx, the non-GT device must observe it, and not respond with AACK* to any following transactions before the GT-6426xx completes the prior transaction’s data tenure (its last TA*). If an address is targeted to the non-GT device, and no other transaction is pending, the non-GT device can handle the transaction and assert the TA*s whenever ready. At this point, the non-GT device must not respond to any following transactions with AACK* until completing its current transaction data tenure.

Figure 5 shows an example of three transactions. The first transaction is targeted to the GT-6426xx. The second and third transactions are targeted to the non-GT device. In the second transaction, the non-GT device waits for the first transaction data tenure to complete (last TA* asserted) before responding with AACK*. The same thing happens in the third transaction. The non-GT device waits for its first transaction data tenure to complete before it responds with AACK*.

Figure 5: Non-Pipeline Mode Transactions



3.6.2 The GT-6426xx in Pipeline Mode and the Non-GT Device in Non-pipeline Mode

The non-GT device can work in non-pipeline mode while the GT works in pipeline mode. The non-GT device must implement one sampling stage to be able to save one address and attributes. However, it must also track the data tenures of the transactions coming before and after the non-GT device address acknowledge, to maintain the proper ordering of the 60x bus protocol. The non-GT device must respond with AACK* for only one transaction and should not respond to a following transaction (TS*), before completing the first transaction data tenure (its last pending TA*s).

The following section describes an example meeting these requirements. This example uses two counters.

Ta_to_aack_cntr Updown Counter

This counter counts the expected number of TA*s for the transactions coming before the non-GT device's address acknowledge. It increments by one or four, depending on the transaction size (burst or partial), with every AACK* assertion for transactions targeted to devices other then itself. It decrements by one with every TA* assertion coming in response to transactions targeted to devices other then itself. The non-GT device must only start its data tenure when this counter is cleared.

Ta_from_aack_cntr Counter

This counter counts the expected number of TA*s for the transactions coming after the non-GT device's AACK* response. It increments by one or four, depending on the transaction size (burst or partial), with every AACK* assertion. The incremention occurs only between the non-GT device AACK* assertion (including) and the completion of its current transaction data tenure (the final pending TA*). When the non-GT device asserts its last TA*, the updated value of ta_from_aack_cntr is written into the ta_to_aack_cntr and cleared. An edge case, where the non-GT device asserts its last TA* together with the GT-6426xx's AACK* assertion, must be taken into account (update the right value to ta_to_aack_cntr).

Figure 6 shows how the counters can be used. There are five transactions involved.

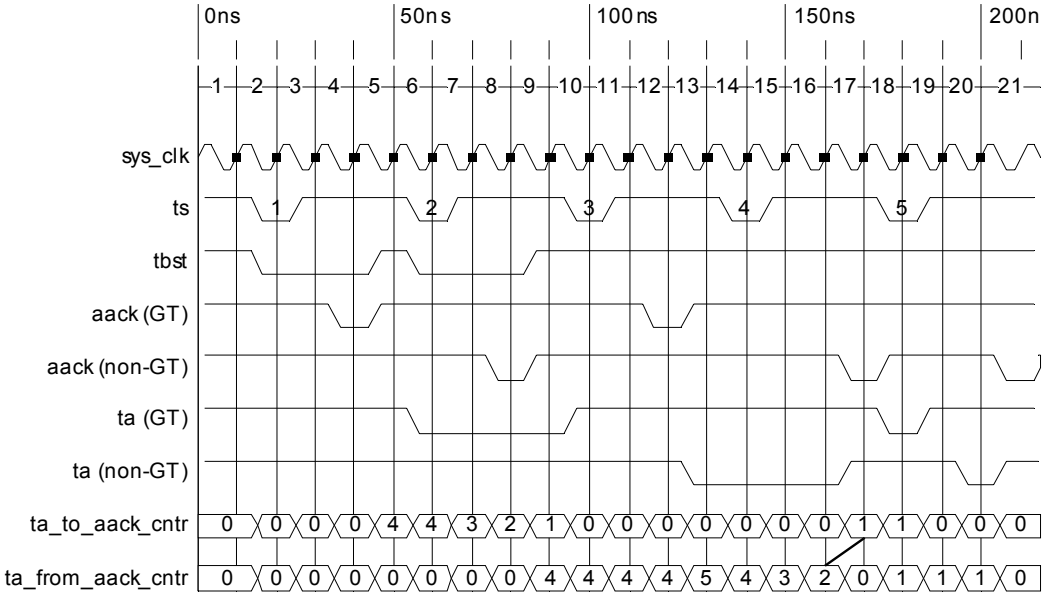
- Transaction 1 is targeted to the GT-6426xx
- Transaction 2 is targeted to the non-GT device
- Transaction 3 is targeted to the GT-6426xx
- Transactions 4 and 5 are targeted to the non-GT device.

Transactions #1 to #3 have no wait states, i.e., AACK* is asserted in the second cycle after TS* assertion.

Transaction #4 is delayed because the non-GT device waits for its first transaction (#2) data to complete (last TA*) and only then responds with AACK*.

Similarly in transaction #5, the non-GT device waits for its second transaction (#4) data to complete (last TA*) before it responds with AACK*.

Figure 6: Pipeline Mode Transactions



3.6.3 GT-6426xx and non-GT Device in Pipeline Mode

The non-GT device can use the same method described above to implement a queue with one to 16 entries, to be able to register several addresses and attributes.

To minimize its logic, the non-GT device can only queue transactions targeted to itself. It does not queue transactions targeted to the GT-6426xx. After responding to a transaction with AACK*, the non-GT device can respond to another following transaction only if it is targeted to itself, however, it must continue tracking (using the counters) the other GT-6426xx device transactions. In case that the next transaction is targeted to the GT-6426xx, the non-GT device must only respond to the following transaction after completing the data tenure of its current transaction.

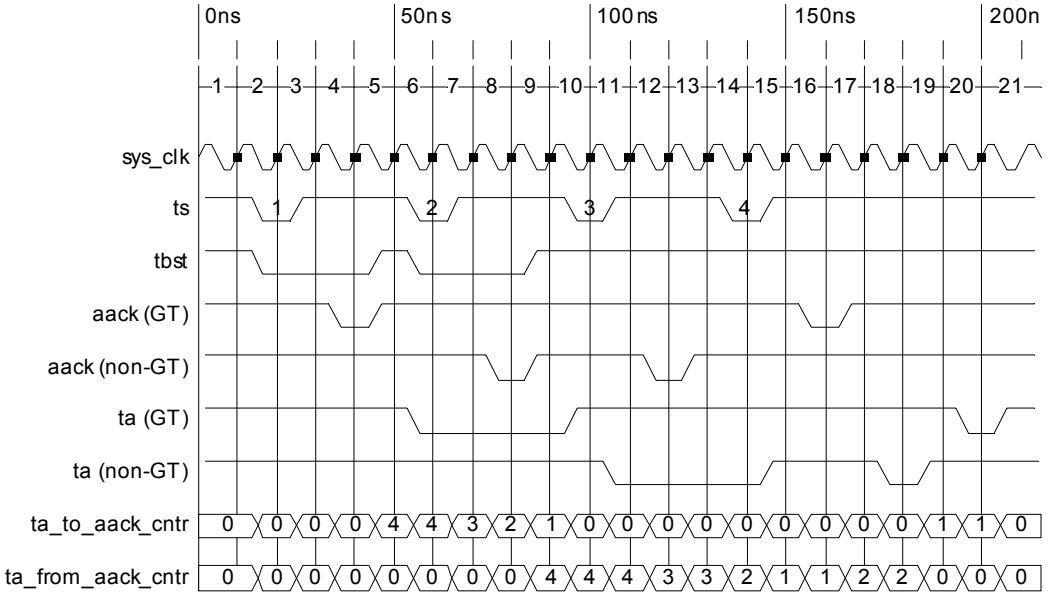
Following are two examples of non-GT device implementing two or more entries queue. This implementation only queues the non-GT transactions but continues tracking (using the counters) the GT-6426xx transactions.

Example One

Four transactions are involved, see Figure 7.

- Transaction 1 is targeted to the GT-6426xx.
- Transaction 2 is targeted to the non-GT device.
- Transaction 3 is targeted to the non-GT device.
- Transaction 4 is targeted to the GT-6426xx.

Figure 7: Example One



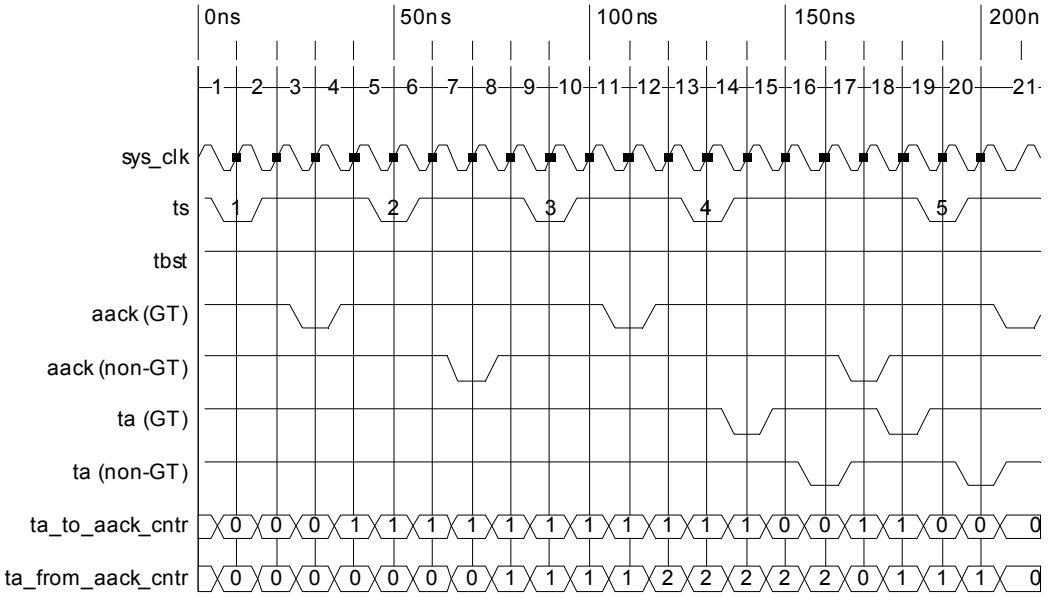
The non-GT device responds with AACK* to Transactions 1 and 3. While waiting for its second transaction (3) data to complete (last TA*), the non-GT device continues counting the expected TA*s of the last (or more) GT-6426xx transactions, for every AACK* assertion. When its data tenure is completed (last TA*), the non-GT device writes the value of ta_from_aack_cntr into the ta_to_aack_cntr, clears the value, and waits for a new transaction targeted at the non-GT device.

Example Two

Five transactions are involved, see Figure 8.

- Transaction 1 is targeted to the GT-6426xx.
- Transaction 2 is targeted to the non-GT device.
- Transaction 3 is targeted to the GT-6426xx.
- Transaction 4 is targeted to the non-GT device.
- Transaction 5 is targeted to the GT-6426xx.

Figure 8: Example Two



Transactions 1 to 3 have no wait states. AACK* is asserted in the second cycle after TS* assertion. While waiting for its first transaction (2) data tenure to complete (last TA*), the non-GT device continues to count the expected TA*s of the following GT-6426xx transaction (3) on the GT-6426xx AACK* assertion. The non-GT device does not assert AACK* for transaction 4 until transaction 2 data tenure is completed (last TA*). Then, the non-GT device writes the value of ta_from_aack_cntr into the ta_to_aack_cntr and clears it. Transaction 5 will not be issued by the CPU before transaction 4 AACK* assertion.

4. Summary

Multi-GT mode is useful for connecting multiple GT-6426xx devices to the CPU(s) as well as connecting a user specific device sitting on the 60x bus parallel to the GT-6426xx. Multi-GT mode is a powerful feature that enables implementing sophisticated applications.

Implementing a multi-GT system requires design considerations, both the in hardware aspects (logic design and AC timing) and software aspects (address decoding, registers configuration, interrupts). To guarantee proper system operation, use the guidelines described in this app note.

Marvell Semiconductor, Inc.
700 First Ave.
Sunnyvale, CA 94089
Phone: (408) 222-2500, Fax: (408) 752-9028
commsales@marvell