HyperTransport™ Consortium

"HTX™ BIOS"

"Generic HTX BIOS Guidelines"

The HyperTransport Consortium

www.hypertransport.org

 Rev 1.01 May 4, 2007

The HyperTransport Technology Consortium disclaims all warranties and liability for the use of this document and the information contained herein and assumes no responsibility for any errors that may appear in this document, nor does the HyperTransport Technology Consortium make a commitment to update the information contained herein.

DISCLAMER

This document is provided “AS IS” with no warranties whatsoever, including any warranty of merchantability, non-infringement, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification or sample. The HyperTransport Technology Consortium disclaims all liability for infringement of property rights relating to the use of information in this document. No license, express, implied, by estoppels, or otherwise, to any intellectual property rights is granted herein.

TRADEMARKS

“HyperTransport” and “HTX” are licensed trademarks of the HyperTransport Technology Consortium.

Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

Table of contents

1. Goals of this document

With an increasing number of companies leveraging the benefits of HyperTransport technology, the HTX connector is being implemented on an increasing number of compute platforms and peripheral / accelerator add-in cards. Existing solutions require modification of platform specific BIOS (Basic Input Output System) in order to recognize and configure the function(s) provided by the HTX card – initial implementations have been performed in a 'fixed' way to address 'point' solutions. This does nothing to address interoperability of existing HTX devices or to reduce time to market and increase adoption of newly developed HTX devices. This fundamental problem is impeding the progress of the HTX ecosystem. The HyperTransport consortium and its members have recognized this problem and launched a working group to understand and address it. This document is intended to capture and communicate guidelines to developers of HyperTransport devices, motherboards and BIOS in order to reduce time to market and significantly improve 'out of the box' interoperability leading to a healthy and growing HTX ecosystem.

The guidelines in this document are intended to describe "what" should happen between platforms, BIOS and HTX devices as opposed to "how" it should happen. This leaves IBV's (Independent BIOS Vendors) and platform manufacturers to innovate and implement the guidelines in a way that's effective and 'compatible' with their existing code base – it also ensures no single implementation is favored over another for a level playing field. The initial guidelines have been developed to address the most common implementation to date ensuring we don't burden the process with unnecessary complexities that would impede the near term progress and success of this effort. Initial goals of the guidelines are as follows: -

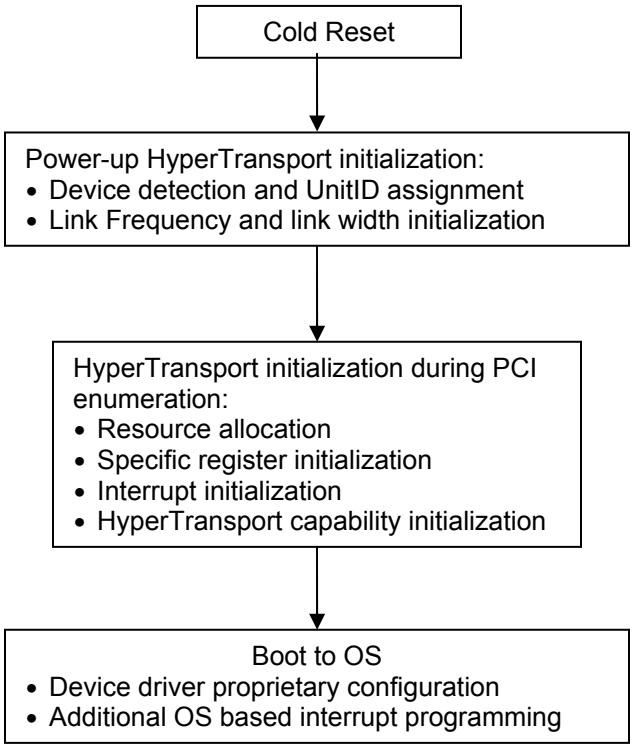
1. Identify / analyze current problem and HTX landscape
2. Determine supported configurations (HTX card based tunnel support, a HTX slot downstream of a motherboard based HyperTransport tunnel etc…)
3. Capture configuration mechanisms needed: -
    a. Early link initialization (Reset delay, link width, link frequency, slot limitations etc…)

b. POST (Power On Self Test) PCI enumeration
    i. Memory window base address programming
    ii. I/O window base address programming
    iii. Interrupt allocation – 3 possible scenarios: -
        1. No interrupts needed - supported
        2. Devices ONLY needing interrupts during OS operation – no POST legacy interrupt support) – supported as follows: -
            a. Native HyperTransport Interrupts
            b. MSI interrupt delivery
        3. Devices needing legacy interrupts during POST and OS operation – **NOT SUPPORTED** due to complexities with legacy interrupt delivery in POST
    iv. HyperTransport memory remap capability block – no known examples using this exist to date so it's outside the scope of this initial implementation – **NOT SUPPORTED**
c. OS based configuration
    i. Proprietary register configuration under device driver control
    ii. OS based interrupt configuration beyond BIOS programming in POST

4. Clearly communicate 'what is' and 'what is not' expected for the first Generic HTX BIOS implementation – but not how!
5. Drive adoption and adherence to the guidelines to ensure newly developed devices can plug into a HTX enabled motherboard and participate in a predictable configuration scheme and that the BIOS knows what it can expect from HTX devices designed to participate in a generic configuration mechanism. There may still be exceptions that require custom development, however we hope to at least minimize and ideally eliminate that need as this mechanism evolves.
6. Welcome participation / feedback from all vested consortium members to improve the capabilities of this generic configuration mechanism as the HTX ecosystem matures and increases in size.

2. HyperTransport Device Initialization

During a cold reset sequence (See HyperTransport I/O Link Specification for definition of cold vs. warm reset), the HyperTransport device on the HTX card will be initialized through the hardware-initiated link-width negotiation sequence and is running at its default width and link frequency potential *. In addition, the HyperTransport device will present itself as device 0 on the primary bus assigned to the HyperTransport link on which it resides. The BIOS must go through the software initialization process to detect the HyperTransport device and program the link width and link frequency according to the capability of the HyperTransport devices on both ends of the HyperTransport link and any system board limitations. Below is an example of a typical BIOS initialization flow: -



* Note: HyperTransport devices are expected to be capable of operating at minimum frequency in support of hardware based link capability negotiation. At least one case of a HyperTransport device being incapable of operating at the MINIMUM frequency has been observed – this case would not be handled in a generic manner and would require specific debug efforts.

3. Power-up HyperTransport initialization

3.1. Reset sequence – hardware initialization timing delay

Note: The reset timing requirements can be found in the specification as follows: -

HyperTransport™ I/O Link Specification Revision 2.00b – Section 12

In addition there are often device specific timing specifications / diagrams as part of specific vendor documentation.

Some HTX implementations may need to delay the reset process allowing an FPGA (Field Programmable Gate Array) based solution to be ready to respond to initial HyperTransport packets. The need for the delay mechanism will depend on the type of reset that occurs and whether the FPGA gets reset requiring re-initialization / programming. The preferred way to achieve this delay in a HTX environment is as follows: -

1. **Back Drive Reset#** - this is the ONLY supported hardware mechanism for allowing HTX based devices to delay reset until they are ready to respond. While there are references to this need in the HyperTransport I/O Link Specification (See: Sections “12.2 System Power up, Reset, and Low-Level Link Initialization” and “2 Signaling – Table 2”) it is somewhat weak and as such has not been implemented on some of the current generation HTX enabled motherboards, there is a proposal to add emphasis for this requirement in the latest generation of HyperTransport / HTX specifications. As systems increase in complexity the need to buffer RESET# will increase, as a result system designers should consider a CPLD approach to allow support of the open-drain functionality while enabling better control of transition times and more complex RESET# distribution models.

3.2. Reset sequence – alternate design considerations to allow for FPGA Initialization time

The hardware mechanism described in Section 3.1 is the only official mechanism promoted by these guidelines; however it is acknowledged that the current generation platforms do not support the mechanism. As a result developers may need to use one or more of the following interim workarounds in order to delay the reset process long enough to allow FPGA based implementations to initialize: -

Workaround 1: In at least one implementation a ~0.5 second delay has been observed between PWROK and RESET#, by keying off of PWROK assertion an FPGA should have enough extra time to achieve programming ahead of RESET# getting de-asserted. Where flexible control over PWROK → RESET# timing is possible a delay of >=0.5 seconds is *highly recommended* for platforms needing to support HTX based FPGA implementations.

Workaround 2: In another implementation the clock to the FPGA flash was increased to improve initialization load times. This should also be considered in conjunction with other suggestions to improve timing margins for any delay mechanisms implemented.

Workaround 3: It **may** be possible to implement an ***optional* BIOS Delay** mechanism – With the HTX slot and device specific table entries it could be possible to add an early BIOS delay / reset mechanism if a HTX device is present to allow FPGA's to be 'ready'. This implementation ***requires a software mechanism capable of generating a COLD reset*** (See: HyperTransport™ I/O Link Specification Revision 2.00b – Section 12.1 Definition of Reset). This approach should only be considered if the hardware mechanism controlling RESET# is not available.

3.3. Device detection and UnitID assignment

After power-up reset, the HyperTransport device presents itself as device 0 on the primary bus on which it resides. The BIOS must detect the HyperTransport device by reading the vendor and device ID at device 0 and then assigning a

non-zero value to its base UnitID. The allocated UnitID then becomes the device number of the HyperTransport device. The HyperTransport device can be accessed with this new device number after this process.

Since the BIOS may not be able to obtain all necessary information regarding the HyperTransport device on a HTX card, the HyperTransport device detection process will assume: -

1. Only one HTX slot is supported on a given HyperTransport link (physical limitation anyway)
2. The generic HTX BIOS implementation must support the following HTX based configurations: -
   a. A single HyperTransport cave or tunnel device
   b. Two exactly identical HyperTransport tunnel devices in a chain
3. When a tunnel device is present it may be connected to the host CPU via either 'side' and the HTX BIOS will dynamically handle this (see section 3.6 for more information).
4. A valid (non 0xFFFF) vendor/device ID must be present

3.4. Link frequency and link width initialization

After device detection and UnitID assignment, the BIOS will maximize the link width and frequency to achieve the highest performance of HyperTransport link operation. The maximum link width and frequency will be determined by the following factors:

1) The width and frequency reported in the HyperTransport Interface Capability blocks
2) The highest capability supported by a given motherboard implementation as reported via the HTX slot descriptor structure in the BIOS (this will typically reflect the maximum supported capabilities from the HTX specification being implemented unless there are overriding motherboard limitations as a result of a design / layout / manufacturing issue)

3) The highest capability supported by a given HyperTransport device as reported in the device specific slot descriptor structure in the BIOS.
4) User defined options selected via a BIOS set-up option that overrides the automatic configuration (**STRONGLY recommended**).

If the HyperTransport device on a HTX card requires a specific setting of the link width and frequency, or there is a limitation on the system board, the BIOS should allow OEM (Original Equipment Manufacturers) or BIOS vendors to override the information detected from the HyperTransport device (a mechanism to provide this data in the slot and device specific descriptors is expected to support this requirement). An example of the HTX slot and device specific descriptors is shown below: -

Today's static table *example*: -

```
<HyperTransport Directory Start>
        ;
        ; Note – HT Frequency capabilities are specified as a 16-bit field
        ; indicating supported frequencies as follows: -
        ;
        ; Bit positions as defined in section 7.5.7 of the HyperTransport I/O Link Spec
        ;
        ; Bit 0 = 200MHz, Bit 1 = 300MHz, Bit 2 = 400MHz … bit 6 = 1GHz
        ; Bits [10:14] reserved for future use, Bit 15 = Vendor Specific
        ;
        <CPU0_Link0, Device_ID, 0x0075, 16-bit, OTHER, ...>
        <CPU0_Link1, Device_ID, 0x0075, 16-bit, OTHER, ...>
        <CPU0_Link2, Device_ID, 0x0035, 16-bit, OTHER, … >  ← FIXED HTX device
        <CPU1_Link0, Device_ID, 0x0075, 16-bit, OTHER, … >
<HyperTransport Directory End>
```

Proposed 2-tier generic table *example*: -

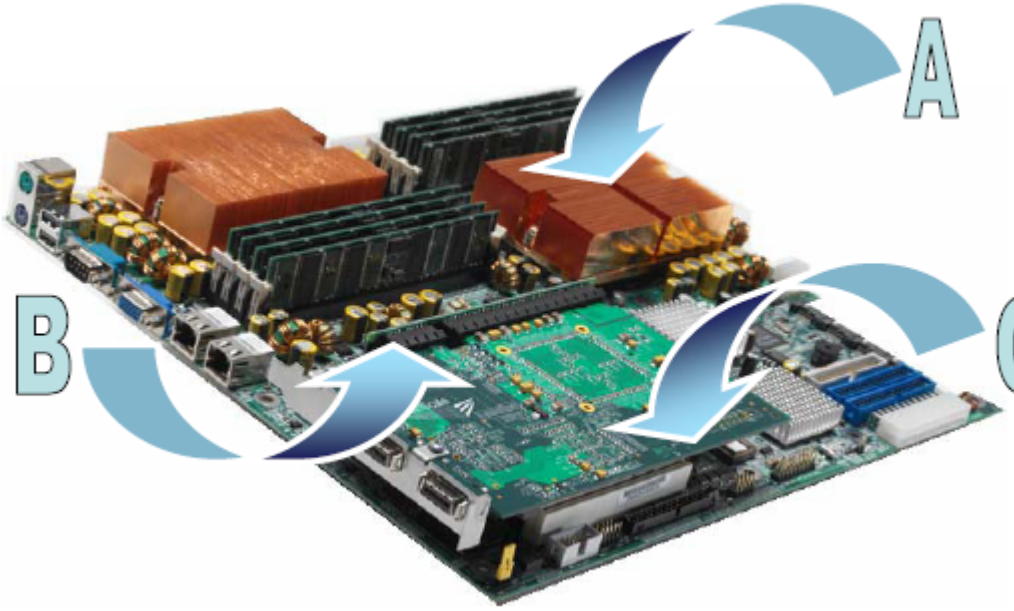
```
<HyperTransport Directory Start>
        ;
        ; Note – HT Frequency capabilities are specified as a 16-bit field
        ; indicating supported frequencies as follows: -
        ;
        ; Bit positions as defined in section 7.5.7 of the HyperTransport I/O Link Spec
        ;
        ; Bit 0 = 200MHz, Bit 1 = 300MHz, Bit 2 = 400MHz … bit 6 = 1GHz
        ; Bits [10:14] reserved for future use, Bit 15 = Vendor Specific
        ;
        <CPU0_Link0, Device_ID, 0x0075, 16-bit, OTHER, ...>
        <CPU0_Link1, Device_ID, 0x0075, 16-bit, OTHER, ...>
        <CPU0_Link2, HTX, 0x0035, 16-bit, OTHER, … >  ← GENERIC HTX SLOT
        <CPU1_Link0, Device_ID, 0x0075, 16-bit, OTHER, … >
<HyperTransport Directory End>

<HTX Exception Table_Start>
        HTX specific table (use for device specific exceptions)
<HTX Exception Table End>
```

In the example above the fundamental shift is from FIXED device specific descriptors describing a HTX slot and it's capabilities to one where the 'generic' configuration described in this document will be applied. There are also provisions for an **Optional Exception** table to handle device specific details that absolutely cannot be handled in a generic manner. The goal is to have no motherboard specific limitations placed in the HTX slot descriptor and no device specific entries needed in an exception table, however these mechanisms are detailed to cover possible exceptions where a slot or device requires special consideration (for example: a motherboard layout error results in a 600MHz capable HTX slot).

3.5. Example HTX scenarios

Below are a couple of example scenarios with fictitious BIOS descriptors showing the resulting 'lowest common denominator' configuration for each case.

In each scenario the following HyperTransport link parameters will be used: -



Key: -

A = Host CPU HyperTransport link capabilities
B = HTX slot HyperTransport capabilities
C = HTX based HyperTransport device capabilities

3.5.1. Scenario 1

The senior design team at company 'X' developed their latest generation motherboard with HTX enablement and followed all the motherboard design guidelines and HTX specification requirements closely. They completed bring-up and have a customer interested in using a HTX based FPGA for acceleration of a financial application by off-loading some specific math algorithms. During motherboard validation they have determined their HTX

implementation is working correctly and they are able to support the full HTX capabilities as specified – 800MHz 16-bit HyperTransport link. They are using an FPGA capable of running at 400MHz with an 8-bit link width. As a result their BIOS would implement the following HyperTransport directory: -

```
<HyperTransport Directory Start>
        ;
        ; Note – HT Frequency capabilities are specified as a 16-bit field
        ; indicating supported frequencies as follows: -
        ;
        ; Bit positions as defined in section 7.5.7 of the HyperTransport I/O Link Spec
        ;
        ; Bit 0 = 200MHz, Bit 1 = 300MHz, Bit 2 = 400MHz … bit 6 = 1GHz
        ; Bits [10:14] reserved for future use, Bit 15 = Vendor Specific
        ;
        <CPU0_Link0, Device_ID, 0x0075, 16-bit, OTHER, ...>
        <CPU0_Link1, Device_ID, 0x0075, 16-bit, OTHER, ...>
        <CPU0_Link2, HTX, 0x0035, 16-bit, OTHER, … > ← GENERIC HTX SLOT
        <CPU1_Link0, Device_ID, 0x0075, 16-bit, OTHER, … >

<HyperTransport Directory End>

;-------------------------------------------------------------------------
; Exception table not needed – only compliant devices
; have been seen so far ☺
;-------------------------------------------------------------------------
;<HTX Exception Table_Start>
;HTX specific table (use for device specific exceptions)
;<HTX Exception Table End>
```

The effective capability parameters for this scenario are as follows: -

A = AMD Opteron™ = 16-bit 1GHz capable
B = HTX slot = 16-bit 800MHz capable
C = HTX based FPGA = 8-bit 400MHz capable

In this scenario the final HTX HyperTransport link configuration should be **8-bits wide at 400MHz** in order to accommodate the slower FPGA capabilities as reported by the FPGA HyperTransport implementation during early link initialization.

3.5.2. Scenario 2

Company 'Y' developed their latest generation server motherboard with HTX enablement. They completed bring-up and have a customer interested in using a HTX based Infiniband card to enable high performance cluster configurations. During motherboard validation they have determined their HTX implementation has a layout problem and they are unable to support the full HTX capabilities as specified – they can only support a **600 MHz** 16-bit HyperTransport link. As a result their BIOS would implement the following HyperTransport directory: -

```
<HyperTransport Directory Start>
        ;
        ; Note – HT Frequency capabilities are specified as a 16-bit field
        ; indicating supported frequencies as follows: -
        ;
        ; Bit positions as defined in section 7.5.7 of the HyperTransport I/O Link Spec
        ;
        ; Bit 0 = 200MHz, Bit 1 = 300MHz, Bit 2 = 400MHz … bit 6 = 1GHz
        ; Bits [10:14] reserved for future use, Bit 15 = Vendor Specific
        ;
        <CPU0_Link0, Device_ID, 0x0075, 16-bit, OTHER, ...>
        <CPU0_Link1, Device_ID, 0x0075, 16-bit, OTHER, ...>
        <CPU0_Link2, HTX, 0x0015, 16-bit, OTHER, … > ← MHz LIMITED HTX SLOT
        <CPU1_Link0, Device_ID, 0x0075, 16-bit, OTHER, … >
<HyperTransport Directory End>

;-------------------------------------------------------------------------
; Exception table not needed – only compliant devices
; have been seen so far ☺
;-------------------------------------------------------------------------
;<HTX Exception Table_Start>
;HTX specific table (use for device specific exceptions)
;<HTX Exception Table End>
```

The effective capability parameters for this scenario are as follows: -

A = AMD Opteron™ = 16-bit 1GHz capable
B = HTX slot = 16-bit **600MHz** capable
C = HTX based IB Device = 16-bit 800MHz capable

In this scenario the final HTX HyperTransport link configuration should be 16-bits wide at **600MHz** in order to accommodate the HTX slot limitations

as a result of a design error and reported via the motherboard specific HTX slot descriptor.

3.5.3. Scenario 3

Company 'Z' developed their latest generation single CPU motherboard with HTX enablement. They completed bring-up and have a customer very interested in using a HTX based Network Processor (NP). During motherboard validation they determined the motherboards HTX slot is able to support the full HTX capabilities as specified – 800Mhz 16-bit HyperTransport link, however the selected network processor (with Device / Vendor ID: 0xFDAA 0x09FA) has an errata specific to the HyperTransport link that means it only supports an 8-bit wide 400MHz link even though *it claims to be 16-bit 800MHz capable*. As a result their BIOS would implement the following HyperTransport directory *with a device specific exception entry* describing this limitation: -

```
<HyperTransport Directory Start>
        ;
        ; Note – HT Frequency capabilities are specified as a 16-bit field
        ; indicating supported frequencies as follows: -
        ;
        ; Bit positions as defined in section 7.5.7 of the HyperTransport I/O Link Spec
        ;
        ; Bit 0 = 200MHz, Bit 1 = 300MHz, Bit 2 = 400MHz … bit 6 = 1GHz
        ; Bits [10:14] reserved for future use, Bit 15 = Vendor Specific
        ;
        <CPU0_Link0, Device_ID, 0x0075, 16-bit, OTHER, ...>
        <CPU0_Link1, HTX, 0x0035, 16-bit, OTHER, … > ← GENERIC HTX SLOT
        ;<CPU0_Link2, Device_ID, 0x0075, 16-bit, OTHER, ...> ← Not connected
<HyperTransport Directory End>

;-------------------------------------------------------------------------
; Exception table needed because: -
; 1) Network Processor model N, revision X.y found to be non-compliant
;-------------------------------------------------------------------------
<HTX Exception Table_Start>
        ; Note – HT Frequency capabilities are specified as a 16-bit field
        ; indicating supported frequencies as follows: -
        ;
        ; Bit positions as defined in section 7.5.7 of the HyperTransport I/O Link Spec
        ;
        ; Bit 0 = 200MHz, Bit 1 = 300MHz, Bit 2 = 400MHz … bit 6 = 1GHz
        ; Bits [10:14] reserved for future use, Bit 15 = Vendor Specific
        ;
        HTX Exception <0xFDAA_09FA, 0x0005, 8-bit, HTX.ValidateDevice (DevID, True)>
<HTX Exception Table End>
```

The effective capability parameters for this scenario are as follows: -

A = AMD Opteron™ = 16-bit 1GHz capable
B = HTX slot = 16-bit 800MHz capable
C = HTX based NP Device = **claims** 16-bit 800MHz capable

In this scenario the final HTX HyperTransport link configuration should be **8-bits wide at 400MHz** in order to accommodate the HTX based network processor errata that limit its HyperTransport capability. This is communicated via a device specific exception entry as the device itself claims to be 800MHz 16-bit capable which would result in the HTX HyperTransport link being programmed incorrectly, potentially causing a system failure (although the HTX generic BIOS should handle this gracefully if at all possible).

Note the addition of a fictitious 'HTX.ValidateDevice(DevID, True)' function parameter in the device specific exception entry – this could allow the execution of a 'validate device' function that further determines the need to apply the capability limitations based on a specific revision of a chip that has the same Device and Vendor ID thus allowing more granular support of exception cases and a potential future workaround should the chip get fixed but maintain the same device and vendor ID.

3.6. HyperTransport tunnel on a HTX card support

If a HyperTransport tunnel device is populated on a HTX card it is possible for the link port implementation to vary (either side of the device could face the host CPU). The BIOS must implement a dynamic mechanism to check the 'Master Host (Bit 10)' and set the 'Default Direction (Bit 11)' of the Command Register (Offset 02h) before proceeding with link width and frequency initialization.

In addition to handling the dynamic configuration of the upstream vs. downstream link the generic HTX BIOS must support a configuration that

allows two exactly identical HyperTransport tunnels to be connected downstream of each other on a single HTX card. By limiting the configuration to two exactly identical tunnel devices we hope to simplify the implementation of this requirement. This will allow HTX developers better leverage of the bandwidth provided by HyperTransport. By supporting two exactly identical devices on a single HTX slot (which is all most systems offer) it allows a doubling of functional density and further increases the performance and value of HTX based functions.

4. HyperTransport initialization during PCI enumeration

4.1. Resource allocation

After the device number is assigned and the link width and frequency are initialized, the HyperTransport device will be initialized by the standard BIOS PCI enumeration process. There are no special resource allocation requirements at this point regardless of whether the HyperTransport device is a bridge or device.

Note: There is an assumption that the BIOS is expected to build any ACPI (Advanced Configuration and Power Interface) and / or MP (Multi-Processor) tables required to support the HTX card in the same way it would for a standard PCI based device. If there are device specific capabilities implemented by the HTX card then custom BIOS development may be required – custom ASL (ACPI Source Language) methods for example.

4.2. Special register initialization

Any device specific register initialization (beyond the first 64 bytes of PCI configuration address space) required by the HyperTransport device on a HTX card can be handled by: -

1. PCI Option ROM (Read Only Memory)

This is a standard approach used by PCI devices requiring specific configuration during BIOS POST initialization. Note: this will only be needed for devices that participate in the boot process before the OS is available.

2. Specific BIOS porting work provide by OEM or BIOS vendors

   The specific register setting can be done by OEM or BIOS vendors during the BIOS building time. The vendor and device ID can be used for device presence test in this case.

3. OS based configuration

   For devices needing specific configuration that do not need to function ahead of the OS booting it's better to configure them via OS based software (device driver / application programming as identified in item 3 below).

4.3. Interrupt initialization

Three potential interrupt requirement scenarios have been identified as follows: -

1. No interrupts needed - supported
2. Interrupts needed during OS present time only - supported
3. Legacy interrupts needed in POST and during OS present – **NOT SUPPORTED**

Note: Future HyperTransport 3.0 devices are expected to support legacy INTx messaging – this is outside of the scope of this initial implementation but is noted here to keep it on the radar for future consideration as needed.

Within each of these scenarios there are implementation specifics as described in the subsequent scenario specific sections: -

4.3.1. No interrupts needed - supported

This scenario covers a HTX based device that needs no interrupt support during the pre-OS or OS present time. This is an easy configuration to support as there's no interrupt specific configuration needed, however it is a valid and supported configuration.

4.3.2. Interrupts needed during OS present time only - supported

This scenario covers a HTX based device that DOES NOT need legacy interrupts during the BIOS POST process but does require interrupt delivery during the OS present time. There are currently Four ways defined for software to interact with HyperTransport interrupts on non-bridge HyperTransport devices as follows: -

1. The HyperTransport Interrupt Discovery and Configuration capability block.
2. The HyperTransport MSI Mapping Capability paired with a PCI MSI Capability or a PCI MSI-X Capability.
3. The memory mapped IOAPIC interface to Interrupt Discovery and Configuration Capability Block. (See: HyperTransport™ I/O Link Specification, Appendix F.1.4)
4. The INTx messages that implement virtual wire support (introduced in the HyperTransport™ I/O Link Specification Revision 1.05 – Section 8.4 INTx Virtual Wire Messages)

The Interrupt Discovery and Configuration capability block is unusable as a generic configuration mechanism using legacy interrupt mode (where everything uses the legacy 8259 PIC) as there is no INTx signal or packet support in today's implementations (future HyperTransport 3.0 devices are expected to introduce INTx capabilities). Instead this mechanism would require programming by the OS to generate the interrupt in an implementation specific way and is thus outside the scope of this document. The BIOS can safely ignore this capability.

The HyperTransport MSI Mapping capability defines a mapping from the PCI MSI 4 byte interrupt packet to the 12-byte HyperTransport interrupt packet. The base address of the interrupt target window is substituted and the low order bits are moved around so they fit in the HyperTransport packet (See: HyperTransport™ I/O Link Specification Revision 2.00b, Section B.5 Message Signaled Interrupts for specific mapping details). On the PC architecture the base addresses and the meaning of all bits are well defined for both MSI and HyperTransport interrupts, and the mapping preserves the well defined meaning on a PC. The definition of the packet mapping is in appendix (B.5 Message Signaled Interrupts).

A PC BIOS needs to ensure the base address is set to its default value of 0x0000_0000_FEEx_xxxx. If a fixed x86 implementation is present this will be the default (identified by 'Fixd' bit 17 in the MSI mapping capability register). For non-fixed or non-x86 architectures this default value should be programmed (See: HyperTransport™ I/O Link Specification Revision 1.05c (or later) section 7.12 - MSI Mapping Capability). Then BIOS must set the enable bit in the HyperTransport MSI Mapping Capability (bit 16).

Once configured, a HyperTransport device with a standard PCI MSI or PCI MSI-X capability will perform logical mapping between the PCI MSI interrupt packet and the HyperTransport interrupt packet before transmitting the HyperTransport packet. It is strongly recommended that the PCI MSI mapping method be implemented as there is much broader operating system support for standard PCI MSI interrupts than for the HyperTransport interrupt discovery and configuration capability.

Note: Based on recent experience with a current implementation it has been strongly recommended the BIOS not attempt to share interrupts allocated to a HTX based device with any other device in the system.

Note: If the BIOS doesn't enable the PCI compatible MSI mapping capability it can be programmed later by operating system, device driver or application code.

4.3.3. Legacy interrupts needed in POST and during OS present – NOT SUPPORTED

This scenario covers a HTX based device that REQUIRES legacy interrupt delivery during BIOS POST, this requirement is **NOT SUPPORTED** by the initial implementation and *may* be addressed by a future implementation as needed. By not supporting legacy interrupt delivery we preclude HTX based devices from participating in the OS boot process – a HTX based network or storage device could only be used to load OS code if it could work in a polled mode (sub-optimal implementation). This is not considered a significant limitation for today's solutions and is thus not supported due to the complications and delays it would introduce near term. If there is an identified near term or future need for HTX legacy interrupt support then please contact the HyperTransport technology consortium to flag the need and provide specific details.

Note: Revision 1.05 of the HyperTransport I/O Link Specification introduced INTx Virtual Wire messages. As of today we're not aware of devices capable of handling these interrupt packets. With the introduction of HyperTransport 3.0 devices there will likely be support for these new packets. INTx packet support will allow interrupts to be delivered in legacy interrupt mode, which will enable interrupt handling during POST / boot. This legacy interrupt delivery mechanism would allow all OS's to handle interrupts from HTX cards.

4.4. Unsupported HyperTransport Capability initialization blocks

Other HyperTransport capability blocks, such as the address remapping capability, have not been implemented on any HTX card that we're aware of to date. As a result, handling of this capability block is not supported by this initial implementation and is noted only here for future consideration as needed. If you have immediate or future needs then please contact the HyperTransport consortium to flag your need.

5. Future considerations on the radar

This section exists to capture additional items for future consideration to make sure they are not lost.

1. **STRONG RECOMMENDATION:** For debug purposes a HyperTransport link width / frequency set-up option should be provided to allow manual override of link configuration parameters. This configuration *could* also be provided in a production BIOS although there is concern that users could configure the system in a way that would stop it from booting.
2. Legacy interrupt delivery – INTx.
3. HyperTransport address remap capability block support.
4. This document is focused exclusively on HTX based devices but there are examples of 'other' HyperTransport connectivity (socket stuffers for example). Could this mechanism be extended to support HTX-like generic configuration of those devices?
5. Revision 3.0 of the HyperTransport I/O Link Specification introduces new capabilities – while not supported here, care should be taken so as to not preclude support for new features in the near future (AC coupled mode, split links, INTx delivery and link power management are examples of potential features needing support).

6. Conclusion

There exists a significant window of opportunity to exploit the clear advantages of HyperTransport technology in performance oriented applications. With the advent of the HTX connector it provides motherboard and chip developers a vehicle to closely couple 'solutions' in an interoperable way. Minimum investment is needed on the part of the motherboard developer to support a HTX slot – the mechanicals and electricals are fairly straight forward and well understood. From a silicon standpoint there are clearly applications benefiting from the distinct advantages of a low latency high bandwidth connection direct to the main system compute resources. Platform vendors have stepped up and provided an increasing number of HTX enabled motherboards – increasing silicon developments are underway. With the help of the HyperTransport technology consortium members, identifying and eliminating potential barriers to adoption opens up a significant opportunity for all involved to unleash the real potential of HyperTransport technology on the computing world at large – let's align our efforts towards executing on this common goal!

The HyperTransport technology consortium thanks you for your interest and looks forward to your continued support and participation.

Note: There is a parallel HyperTransport technology consortium effort to address LinuxBIOS as a development / debug vehicle for HTX based implementations. If you're interested in participating in either / both HTX related working groups then please contact Mario Cavalli at: -

E-mail: Mario.Cavalli@hypertransport.org
Phone: +1 (925) 968-0220

7. Generic HTX BIOS requirements matrix

The following table provides a matrix of required versus unsupported mechanisms to help manufacturers and BIOS developers quickly identify the needs communicated in these guidelines: -

Feature	BIOS Required	OS Required	Unsupported / Recommended / On RADAR
HTX reset delay feature – FPGA initialization mechanism (Hardware / Software) – See 3.1 and 3.2	☑		
Two tier HTX descriptor mechanism allowing deployment options for motherboard / device specific details – See 3.4	☑		
HyperTransport link width / frequency initialization mechanism resulting in lowest common denominator selection based on HTX slot and device capabilities from descriptor table entries and device enquiry mechanisms – See 3.5	☑		
Mechanism to detect and configure a HTX based tunnel device host interface direction dynamically and two exactly identical tunnels downstream of each other on a single HTX card – See 3.6	☑		
HyperTransport MSI remap register programming default checked against addendum in HyperTransport I/O Link Specification – default for x86 systems should be programmed, other architectures may require specific programming – See 4.3	☑		
Standard POST PCI enumeration scheme configuration of I/O and Memory base address registers, control register etc… to ensure HTX device is accessible to OS drivers / applications.	☑		
Native HyperTransport interrupt delivery mechanism supported – See 4.3		☑	
PCI compatible MSI interrupt delivery mechanism supported – See 4.3		☑	
Legacy interrupt delivery in POST for HTX devices – See 5			☒
HyperTransport address remap capability block support – See 5			☒
Non-HTX connected device support via generic configuration mechanism - See 5			☒
HyperTransport 3.0 specific capability configuration - See 5			☒
STRONGLY RECOMMENDED: BIOS set-up option to override HyperTransport link width and frequency settings for development / debug purposes - See sections 3.4 and 5			☑

For more information on HyperTransport technology please visit the HyperTransport Technology Consortium website at www.hypertransport.org where additional white papers, detailed specifications and information on becoming a member of the HyperTransport Consortium are available.

A low-cost membership in the Consortium enables member companies to have royalty-free access to HyperTransport IP and to participate in the Consortium's technical working groups that manage future extensions of the HyperTransport specifications and protocols.