Wireless Host Controller Interface Specification for Certified Wireless Universal Serial Bus

Date: June 16, 2006

Revision: 0.95

Revision History

Revision	Issue Date	Comments
0.84	8/8/2005	Initial Revision
0.91	2/3/2006	Addressed multiple issues from the initial revision
0.92	2/23/2006	Added support to correctly reactivate an inactive Queue Set
0.94	4/24/2006	Modified the DRP Notification Send Command Frame command modified to accept data to be sent in a command frame as opposed to the raw command frame itself. Minor editorial changes Added an a new Event that is returned for Unknown Commands for Command Types sent to the URC Added a Channel Change Notification Added the ability to selectively receive notifications when an ASIE is sent by a peer device. Removed "Unsecure Control Transfer" from the list of Transfer Types

Revision	Issue Date	Comments
0.94a	5/7/2006	Added the URC Scan State when the Start and Stop Beaconing commands can be sent to the URC. Clarified that the only command that may be sent to the URC when its state is transitioning is the Reset Command Clarified the Set ASIE notification command in the case multiple Sets are received with the same ASIE Identifier but with different addresses.
0.95RC	6/2/2006	The URCD must preserve any bits it does not want to modify when making changes to URCCMD register. The WHCD must preserve any bits it does not want to modify when making changes to WUSBCMD register. Added 2 control bits in the WUSBCMD register to indicate that a QSet was removed from the schedule. The URC must have internal storage for the results of commands for any command that it accepts if the command completes when event processing is disabled. The WHC must deactivate a QHead when the I-Bit is set by the WHCD regardless of whether it is at the burst boundary or not. The WHC must update DWords 5 and when it retires a qTd/iTD. Editorial changes to Chapter 4.

Please send comments via electronic mail to: whci@Intel.com

This page intentionally left blank.

TABLE OF CONTENTS

This page intentionally left blank.

1. Introduction

The Wireless Host Controller Interface (WHCI) specification describes the register-level hardware/software interface between system software and the UWB Multi-Interface Controller (UMC) for the Wireless Universal Serial Bus (Wireless USB) host system. The UMC is a hardware functional component for the WiMedia UWB radio platform and may provide one or more Protocol Adaptation Layer (PAL) functions such as Wireless USB and WiNET.

This specification is intended for hardware component designers, system builders and device driver (software) developers. The reader is expected to be familiar with the *Wireless Universal Serial Bus Specification, Revision 1.0*. In spite of due diligence, there may exist conflicts between this specification and the *Wireless Universal Serial Bus Specification*. The *Wireless Universal Serial Bus Specification* takes precedence on all issues of conflict.

This specification also defines an interface for control and management of the WiMedia UWB radio, i.e. the UWB Radio Controller Interface (URCI). The reader is expected to be familiar with the WiMedia MAC layer protocol defined in the *WiMedia MAC Specification* (*Distributed Medium Access Control (MAC) for Wireless Networks, Release 1.0*). If there are conflicts between this specification and the *WiMedia MAC Specification*, then the *WiMedia MAC Specification* takes precedence on all issues of conflict.

Some key features of the WHCI specification are:

- **Full, Robust Support for all Wireless USB Features.** This specification describes the Wireless USB Host Controller function that correctly supports all compliant Wireless USB devices.
- **Provides a Generic Mechanism to expose Multiple PAL Interfaces.** The UMC can provide one or more PAL functions for the UWB radio platform. This specification defines a mechanism how the hardware exposes multiple PAL interfaces which allows system software to load an appropriate device driver for each interface.
- **Common Control and Management Interface of the UWB Radio.** The UMC exposes a common interface to control the UWB radio. System software can request the hardware to scan UWB channels, transmit beacons, make bandwidth reservation, etc. This interface also provides a mechanism for the hardware to send asynchronous event notifications to system software. This interface can be commonly used for Wireless USB and all the other PALs exposed by the UMC hardware.
- **System Power Management**. Current PC architectures are providing ubiquitous support for aggressive power management. Wireless USB is a critical component in delivering a consistent, coherent and robust user experience. If the implementation includes PCI configuration registers, then the UMC is required to implement a PCI Power Management Interface.
- **Support for 64-bit Addressing.** Over the implementation lifetime of this specification, it is expected that UMC hardware will be used increasingly in architectures that support more than 32-bits of addressable memory space. For this reason this specification defines all data structures with 64-bits of addressing.

This specification presents two chapters of pure structural definitions of the register space, commands, notifications and schedule interface data structures. These definition chapters contain little or no operational requirements or usage models. The definition chapters are followed by a detailed description of the operational model requirements of the UMC, using the previously defined registers, commands, notifications and schedule interface data structures. The following list summarizes the organization of this specification:

- Section 1.1 provides an overview of the architecture of the UMC.
- Chapter 2 defines the register spaces of the UMC.
- Chapter 3 defines the schedule data structures, command and notification structures used to communicate with the UMC.
- Chapter 4 defines the details of the operational model for the UMC. It uses example behavioral abstractions to describe the operational requirements.

- Appendix A defines the details of the PCI Power Management Interface for the UMC.

1.1 Architectural Overview

A Wireless USB Host System is composed of a number of hardware and software layers. Figure 1-1 illustrates a conceptual block diagram of the building block layers in a host system that work in concert to support Wireless USB.

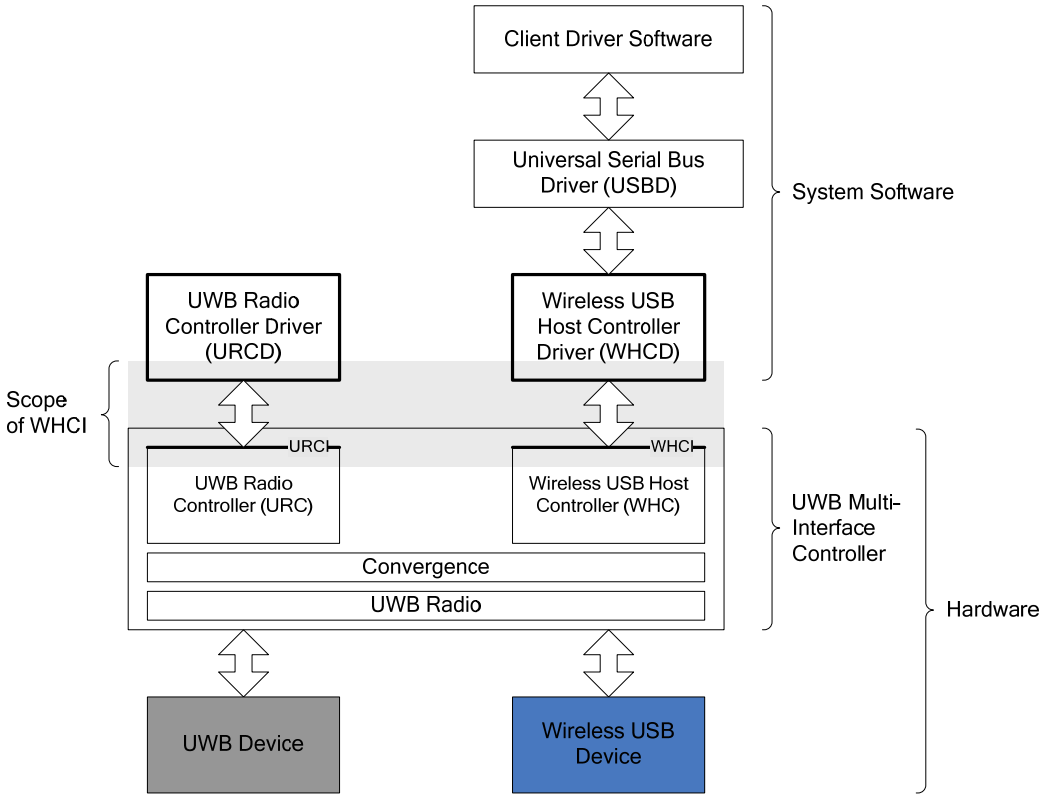


Figure 1-1. Wireless Universal Serial Bus, System Block Diagram

The component layers are:

- **Client Driver Software.** This software executes on the host PC corresponding to a particular Wireless USB device. Client software is typically part of the operating system or provided with the Wireless USB device.
- **USB Driver (USBD).** The USBD is a system software component that abstracts the details of the particular host controller driver for a particular operating system.
- **Wireless USB Host Controller Driver (WHCD).** The WHCD provides the software layer between a specific Host Controller hardware and the USBD.
- **UWB Radio Controller Driver (URCD).** The URCD is a system software component that provides control functions of the UWB Radio. It is required in every implementation of a UMC.
- **Wireless USB Host Controller (WHC).** This is the specific hardware implementation of the Wireless USB host controller.
- **UWB Radio Controller (URC).** This is the standard UWB radio control function that is required in every implementation of a UMC.

- **Wireless USB Device.** This is a hardware device that performs a useful end-user function. Interactions with Wireless USB devices flow from the applications through the software and hardware layers to the Wireless USB devices.
- **UWB Device.** This is another external hardware device with a UWB radio interface. This device may have one or more PAL functions, e.g. WiNET.

1.1.1 Interface Architecture

The UMC exposes three interface spaces (see Figure 1-2):

- **PCI Configuration Space**. If the implementation includes PCI registers, they are used for system component enumeration and PCI power management.
- **Register Space.** Implementation-specific parameters and capabilities, plus operational control and status registers. This space, normally referred to as I/O space, must be implemented as memory-mapped I/O space.
- **Schedule Interface Space.** This is typically memory allocated and managed by the WHCD for asynchronous and periodic schedules.

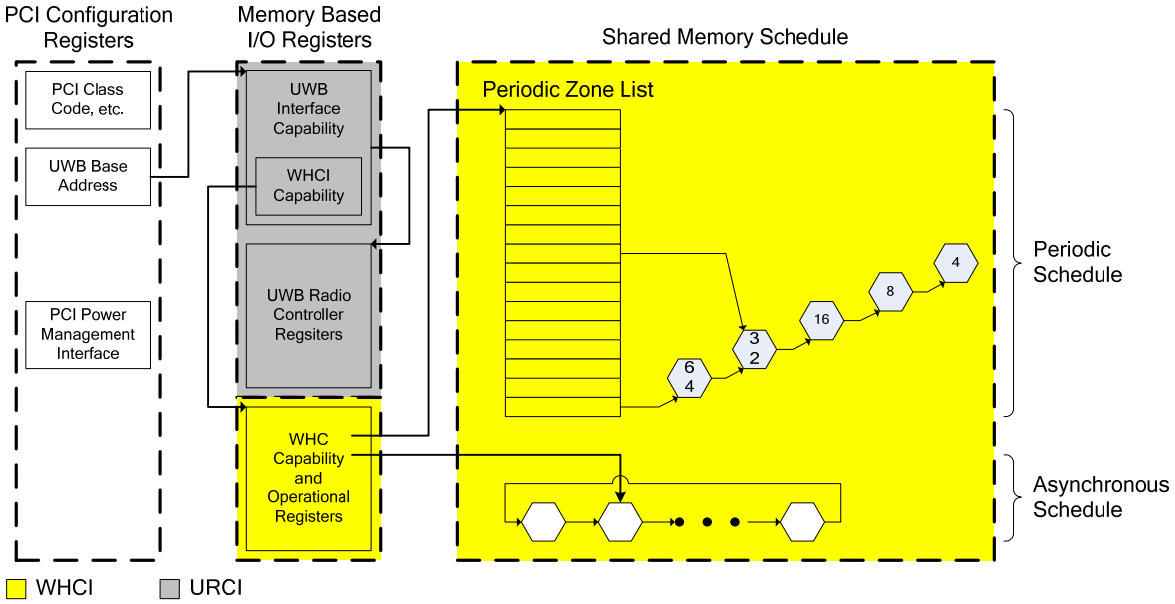


Figure 1-2. General Architecture of UWB Radio Controller and Wireless USB Host Controller Interface

The UMC is a PCI device function that exposes several functional interfaces. The interfaces supported are exposed via capability structures (similar to the standard PCI Capability structure) at the beginning of memory-mapped I/O register space. The UWB Interface Capability Register space identifies the type of each interface and is also used to determine the register space for each interface.

The first interface exposed by the UMC is the UWB Radio Controller Interface (URCI) which is used for the control and operation of the UWB Radio and is mandatory for all hardware implementations. This interface provides a mechanism to exchange message-based commands and notifications between system software and the URC. The location of the UWB Radio Controller Registers is determined from the information in the UWB Interface Capability Register space.

The UMC may also expose one or more PAL interfaces. This specification defines one of these PAL interfaces used to control the Wireless USB Host Controller. The Wireless USB Host Controller Interface (WHCI) provides support for two categories of transfer types: asynchronous and periodic. Periodic transfer types include both isochronous and interrupt. Asynchronous transfer types include control and bulk. Figure 1-2 illustrates that the WHCI schedule interface for Asynchronous and Periodic transfer types.

The asynchronous schedule is a circular list of schedule work items that provides a round-robin service opportunity for all asynchronous transfers. The periodic schedule is based on a time-oriented Periodic Zone[1] List that represents a sliding *window* of time of host controller work items. Software links schedule data structures to the periodic zone list to produce a graph of data structures. The graph represents an appropriate sequence of transactions on Wireless USB.

The WHCI allows software to enable or disable each schedule as well as provides the ability for software to inform the WHC when it has modified either schedule.

1.1.2 WHCI Schedule Data Structures

The WHCI manages all transfer types using a buffer queuing data structure. The queuing data structure provides automatic, in-order streaming of data transfers. Software can asynchronously add data buffers to a queue and maintain streaming. Wireless USB-defined short packet semantics are fully supported on all processing boundary conditions without software intervention.

[1] A Zone refers to 16 continuous Media Access Slots in a superframe. The first Zone starts at MAS 0 and ends at MAS 15, the second Zone starts at MAS 16 and ends at MAS 31 and so on so forth with the last Zone starting at MAS 242 and ending at MAS 255

2. Register Interface

This section defines the register interfaces for system software to control the UWB Multi-Interface Controller (UMC). This includes the memory-mapped register set definitions for the UWB Radio Controller (URC) and the Wireless USB Host Controller (WHC) interfaces. It also specifies how to identify other register interfaces exposed by the UMC, e.g. a Network Interface. This specification does not provide any other register set definitions.

The URC contains three register sets. Memory-mapped UWB Interface Capability Registers, memory-mapped UWB Radio Controller Registers and the PCI configuration registers. Note that the PCI configuration registers are only required for PCI devices that implement the UMC.

- **PCI Configuration Registers (For PCI devices).** The standard PCI header and device specific registers are beyond the scope of this document (The CLASSC register is shown in this document). Note that the URCD does not interact with the PCI configuration space. This space is used only by the PCI enumerator to identify the URC, and assign the appropriate system resources.
- **Memory-mapped UWB Hardware Capability Registers.** This block of registers is memory-mapped into non-cacheable memory. This memory space must begin on a DWord (32-bit) boundary. This register space contains a set of read-only capability registers.
- **Memory-mapped UWB Radio Controller Registers.** This block of registers is memory-mapped into non-cacheable memory. This memory space must begin on a DWord (32-bit) boundary. This register space contains a set of read-only capability registers.

The WHC contains only a set of Memory-mapped Host Controller registers.

- **Memory-mapped Wireless USB Host Controller Registers.** This block of registers is memory-mapped into non-cacheable memory. This memory space must begin on a DWord (32-bit) boundary. This register space contains two types of registers: a set of read-only capability registers and a set of read/write operational registers.

Note that the URC and any other register interfaces that the UMC may expose are not required to support exclusive-access mechanisms (such as PCI LOCK) for accesses to the memory-mapped register space. Therefore, if system software attempts exclusive-access mechanisms to any of the interfaces' memory-mapped register space, the results are undefined.

The reserved bits defined in this revision of the specification may be allocated in later revisions. System software should not assume reserved bits are always zero and should preserve these bits when writing to modifiable registers. The following notation is used to describe register access attributes:

RO **Read Only**. If a register is read only, writes have no effect.

WO **Write Only**. If a register is write only, reads return a zero for all bit positions.

R/W **Read/Write**. A register with this attribute can be read and written. Note that individual bits in some read/write registers may be read only.

R/WC **Read/Write Clear**. A register bit with this attribute can be read and written. However, a write of a 1 clears (sets to 0) the corresponding bit and a write of a 0 has no effect.

Registers in the auxiliary well are reset under different conditions than the registers in the core well. The auxiliary well, memory-space registers are initialized to their default values in the following cases:

- initial power-up of the auxiliary power well, or
- a value of 1b in *UWBRESET* (see Section 2.3.1)

The core well, memory-space registers are initialized to their default values in the following cases:

- assertion of the system (core-well) hardware reset, or
- a value of 1b in *UWBRESET* (see Section 2.3.1), or

- transition from the PCI Power Management D3hot state to the D0 state

Exceptions to these reset conditions will be defined in the associated register section.

2.1 PCI Configuration Registers

Table 2-1 lists the PCI configuration space register for a UMC. The following subsections describe details about each set of registers defined.

Table 2-1. PCI Configuration Space Registers

Configuration Offset	Mnemonic	Register	Power Well	Register Access
00–08h	—	Register implementation as needed for specific PCI device	Core	—
09–0Bh	CLASSC	Class Code	Core	RO
0C–0Fh	—	Register implementation as needed for specific PCI device	Core	—
10–17h	UWBBASE	64 bit Base Address to Memory-mapped UWB Interface Capability Register Space	Core	R/W
18–33h	—	Register implementation as needed for specific PCI device	Core	—
34h	CAP POINTER	Must be non zero as all UMC implementations must support the PCI Power Management Capability and possibly an MSI capability	Core	RO
35–FFh	—	Register implementation as needed for specific PCI device	Core	—

2.1.1 CLASSC — CLASS CODE REGISTER

Address Offset: 09–0Bh
Default Value: 0D1010h
Attribute: RO
Size: 24 bits

This register contains the device programming interface information related to the Sub-Class Code and Base Class Code definition. This register also identifies the Base Class Code and the function sub-class in relation to the Base Class Code.

Table 2-2. Class Code Register Bit Definitions

Bit	Description
23:16	**Base Class Code (BASEC).** 0Dh= Wireless Controller.
15:8	**Sub-Class Code (SCC)**. 10h=RF Controller.
7:0	**Programming Interface (PI).** 10h=UMC that conforms to this specification.

2.1.2 UWBBASE — Register Space Base Address Register

Address Offset: 10–17h
Default Value: Implementation Dependent
Attribute: R/W
Size: 64 bits

This register contains the base address of the DWord-aligned memory-mapped UWB Interface Capability Register space. The number of writable bits in this register determines the actual size of the required memory space window. The minimum required is specified in this specification. Individual implementations may vary.

Table 2-3. Register Space Base Address Register Bit Definitions

Bit	Description
63:8	**Base Address — R/W.** Corresponds to memory address signals [63:8].
7:3	**Reserved — RO.** These bits are read only and must be set to zero.
2:1	**Type — RO.** This field has two valid values: **Value** — **Description** 00b — May only be mapped into 32-bit addressing space. 10b — May be mapped into 64-bit addressing space (Recommended).
0	**Reserved — RO.** This bit is read only and must be set to zero.

2.1.3 PWRMGT — PCI Power Management Interface

UMC implementations are required to implement the PCI Power Management registers as defined in the *PCI Bus Power Management Interface Specification Revision 1.1*. Refer to Appendix A for the UMC operational requirements for PCI Power Management.

2.1.4 MSI Capability

Address Offset: Implementation Dependent
Default Value: Implementation Dependent
Attribute: R/W
Size: 10/14 bytes

Message Signaled Interrupts (MSI) is an optional feature that a UMC may use in place of line based interrupts. A minimum of one vector is required for proper operation of the UMC.

Table 2-4. MSI Capability Bit Definitions

DWord	Bit	Description
0	31:25	**Reserved — RO.** These bits are read only and must be set to zero.
0	24	**Per-vector masking capable — RO.** This field must be set to zero. UMCs must not support per vector masking
0	23	**64 bit address capable — RO.** If this bit is set then DWord 2 will be set to the *Message Upper Address* otherwise only the lower 16 bits of DWord 2 will be valid and will contain the *Message*Table 2-4. MSI Capability Bit Definitions *Data*. 64-bit support is recommended in an implementation of the UMC.

Table 2-4. MSI Capability Bit Definitions (cont.)

DWord	Bit	Description
0	22:20	**Multiple Message Enable — R/W.** The PCI enumerator writes to this field to indicate the number of allocated vectors (equal to or less than the number of requested vectors). The number of allocated vectors is aligned to a power of two. E.g. If the UMC requests four vectors (indicated by a Multiple Message Capable encoding of “010”), the PCI enumerator can allocate either four, two, or one vector by writing a “010”, “001, or “000” to this field, respectively. When MSI is enabled, a function will be allocated at least 1 vector. The encoding is defined as: **Value** — **Number of Vectors allocated** 000b — 1 001b — 2 010b — 4 011b — 8 100b — 16 101b — 32 110b — Reserved 111b — Reserved This field’s state after reset is “000”.
0	19:17	**Multiple Message Capable — RO.** The PCI enumerator reads this field to determine the number of requested vectors. The number of requested vectors must be aligned to a power of two (if the UMC requires three vectors, it requests four by initializing this field to “010”). The encoding is the same as for the *Multiple Message Enable* field.
0	16	**MSI Enable — R/W.** If the PCI enumerator set’s this field to a 1 the UMC is permitted to use MSI to request service and is prohibited from using its INTx# pin (if implemented). System configuration software sets this bit to enable MSI. If 0, the UMC hardware is prohibited from using MSI to request service. This bit’s state after reset is 0 (MSI is disabled).
0	15:8	**Next Pointer — RO.** Pointer to the next item in the capabilities list.
0	7:0	**Capability ID — RO.** The value of 05h in this field identifies the UMC hardware as being MSI capable.
1	31:2	**Message Address — R/W.** System-specified message address. If the *Message Enable* bit is set, the contents of this register specify the DWord-aligned address for the MSI memory write transaction.
1	1:0	**Reserved — RO.** These bits are read only and must be set to zero.
2	31:0	**Message Upper Address — R/W.** System-specified message upper address. This register is optional and is implemented only if the UMC supports 64-bit message addresses. If the *Message Enable* bit is set, the contents of this register specify the upper 32-bits of a 64-bit message address.
3	15:0	**Message Data — R/W.** System-specified message data. If the *Message Enable* bit is set, the message data is driven onto the lower Word of the memory write transaction’s data phase. The *Multiple Message Enable* field defines the number of low order message data bits the UMC is permitted to modify to generate its PCI enumerator allocated vectors. For example, a *Multiple Message Enable* encoding of “010” indicates the UMC has been allocated four vectors and is permitted to modify message data bits 1 and 0 (the UMC modifies the lower message data bits to generate the allocated number of vectors). If the *Multiple Message Enable* field is “000”, the UMC is not permitted to modify the message data.

The number of vectors that the UMC requests must be equal to the total number of interfaces exposed by the UMC plus 1 rounded off to the next higher power of two. For example if 2 interfaces are exposed, then the *Multiple Message Capable* field must be set to 010b. See Section 2.2.1 for details on how to determine the number of interfaces exposed by the UMC.

After the system configuration software (PCI Bus enumerator) configures the UMC, it will use MSI if the *MSI Enable* bit is set otherwise it must use line based interrupts. The following rules determine how the UMC uses the vectors assigned to it when the *MSI Enable* bit is set:

- If the number of vectors assigned to the UMC is less than the number of interfaces exposed plus 1, then the UMC will only use one message vector (and hence only one Message Data value) for all interrupts.
- If the number of vectors assigned is greater than the number of interfaces exposed by the UMC, then the first vector will be used to indicate that the URC generated an interrupt. The remaining vectors will be assigned in ascending order with the lowest vector assigned to the first interface exposed, the next vector assigned to then next interface exposed etc. If more vectors are assigned to the UMC than the number of interfaces exposed plus 1, then those vectors must not be used.

2.2 UWB Interface Capability Registers

Table 2-5. UWB Interface Capability Registers

Offset	Size	Mnemonic	Power Well	Register Name
00h	8	UWBCAPINFO	Core	UWB Interface Capability Information
08h	8 * *N_CAP*	UWBCAPDATA	Core	UWB Interface Capability Data

2.2.1 UWBCAPINFO — UWB Interface Capability Information Register

Address: UWBBase + (00h)
Default Value Implementation Dependent
Attribute: RO
Size: 64 bits

This register contains the number of interface capabilities exposed by the UMC and the information required to access the URC operational registers (see Section 2.3).

Table 2-6. UWBCAPINFO — UWB Interface Capability Information Register Bit Definitions

Bit	Description
63:48	**Reserved.** These bits are reserved and must be set to zero.
47:32	**URCI Version.** This field contains a BCD encoding of the URCI revision number supported by this UMC. The most significant byte of this field represents a major revision and the least significant byte is the minor revision. This field must be set to 0095h to conform to this version of the specification.
31:18	**URC Offset.** Offset from the Bar where the first register of the URC Registers is present. The value is expressed in byte units.
17:16	**URC Bar to Use.** Bar to use to access the URC Registers. The legal values are 0, 1 and 2. BAR 0 refers to Base Address Register at offset 10h, BAR 1 refers to Base Address Register at offset 18h, and BAR 2 refers to Base Address Register at offset 20h in the PCI Configuration space.
15:4	**Reserved.** These bits are reserved and must be set to zero.
3:0	**Number of Capabilities (N_CAPS).** This field specifies the total number of other interfaces exposed by the UMC. This indicates the number of UWBCAPDATA registers following this register. The maximum number of capabilities is restricted to 8.

2.2.2 UWBCAPDATA — UWB Interface Capability Data Register

Address: UWBBase + (08h + (8 * (Capability Number-1)))
where: *Capability Number* is 1, 2, 3, … N_CAPS
Default Value Implementation Dependent
Attribute: RO
Size: 64 bits * N_CAPS

The UMC must implement one or more UWB Interface Capability Data registers. These registers identify the register interfaces exposed by this UMC and are used by the host enumerator software to load the appropriate software for each interface. The location of the first capability is immediately after the UWBCAPINFO register (see Section 2.2.1) and the total number of the registers is indicated by the *N_CAPS* field in the UWBCAPINFO register.

Table 2–7. UWBCAPDATA — UWB Interface Capability Data Register Bit Definitions

Bit	Description
63:48	**Reserved.** These bits are reserved and must be set to zero.
47:32	**Version ID.** Version of the specification that this capability conforms to.
31:18	**Offset.** Offset from the Bar where the first register for this capability is present. The value is expressed in byte units.
17:16	**Bar to use.** Bar to use to access this capability's register space. The legal values are 0, 1 and 2. BAR 0 refers to Base Address Register at offset 10h, BAR 1 refers to Base Address Register at offset 18h, and BAR 2 refers to Base Address Register at offset 20h in the PCI Configuration space.
15:8	**Size.** The size in DWords of this capabilities register space.
7:0	**Capability ID.** This field identifies the UMC capability. See Table 2–8 for a list of the valid UMC capability codes.

Table 2–8. UMC Capability Codes

ID	Name	Description
00h	Reserved	This value is reserved and must not be used.
01h	WHCI Capability	Wireless USB Host Controller Interface support capability. The WUSBBase address is determined by the *Bar to use* and *Offset* fields of this capability. See Section 2.4 for details of this register interface. The Host enumerator may load the class driver for this interface based on the UMC Class Code and this capability ID. *Version ID* field for the WHCI Capability defined must be set to the same value as WHCIVERSION register (see Section 2.4.1).
02-7Fh	Reserved	These values are reserved for future extension.
80-FFh	Vendor Specific Capability	Vendor specific interface capability. This specification does not specify these register interfaces. The Host enumerator may load the vendor specific drivers for these interfaces based on the UMC Vendor ID, Device ID and these capability IDs.

2.3 UWB Radio Controller Registers

This section defines the URC operational registers. The location of these registers, i.e. URCBase, is determined by *URC Bar to use* field and *URC Offset* field in UWBCAPINFO register. All the URC registers are DWord aligned and the URCD should read and write the URC registers using DWord accesses only. Figure 2-1 shows how the URC exposes the registers defined in this specification.

Table 2-9. UWB Radio Controller Registers

Offset	Size	Mnemonic	Register Name	Power Well	Section
00h	4	URCCMD	UWB Radio Controller Command	Core	2.3.1
04h	4	URCSTS	UWB Radio Controller Status	Core	2.3.2
08h	4	URCINTR	UWB Radio Controller Interrupt Enable	Core	2.3.3
10h	8	URCCMDADDR	UWB Radio Controller Command Address	Core	2.3.4
18h	8	URCEVTADDR	UWB Radio Controller Event Address	Core	2.3.5

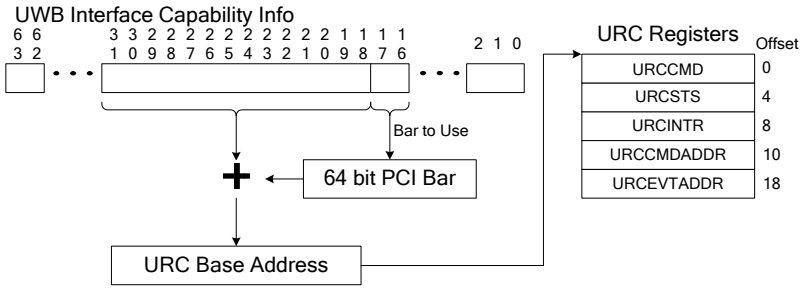


Figure 2-1. URC Register Organization

2.3.1 URCCMD — URC Command Register

Address: URCBase + (00h)
Default Value: 00000000h
Attribute RO, R/W (field dependent)
Size 32 bits

The URCD must use this register to reset the UMC hardware. On completion of a reset, the UMC and all associated registers must be reset to the default state. This register is also used to send radio control commands (e.g. Scan, Start Beaconing etc) to the URC.

Note that the URCD must take care to preserve all the bits in this register that it does not want to modify when it performs a write operation to this register.

Table 2-10. URCCMD – URC Command Register Bit Definitions

Bit	Description
31	**UWB Hardware Reset (UWBRESET) — R/W.** This control bit is used by the URCD to reset the UMC. The effects of this are similar to a Chip Hardware Reset. When the URCD writes a one to this bit, the UMC resets its internal pipelines, timers, counters, state machines, etc. to their default values. Any transactions currently in progress are immediately terminated. PCI Configuration registers are not affected by this reset. This bit is set to zero by the UMC when the reset process is complete. The URCD cannot terminate the reset process early by writing a zero to this register.
30	**Run/Stop (RS) — R/W.** Default 0b. 1=Run. 0=Stop. When set to a 1, the URC proceeds with execution of the radio control operations (commands and notifications). The URC continues execution as long as this bit is set to a 1. When this bit is set to 0, the URC cancels any pending radio control commands and then halts. The URC also disables further event processing. The *Halted* bit in the status register indicates when the URC has finished its pending commands and has entered the stopped state. None of the other interfaces (see Section 2.2.2) exposed by the UMC can operate when the URC is halted.
29	**Event Address Register Valid — R/W.** Default 0b. The URCD must set this bit after it has updated the URCEVTADDR register. This indicates to the URC that the Event Address is valid. The URC must set the *Event Processing Status* bit in the URCSTS register and clear this bit after it has read the contents of the Event Address Register. The URCD can only update the URCEVTADDR register when this bit is set to zero and *Event Processing Status* bit in the URCSTS register is set to zero. A write of zero to this field has no effect.
28:16	**Reserved — RO**. These bits are reserved and must be set to zero.
15	**Active — R/W**. The URCD writes a 1 to this bit to indicate to the URC that the buffer contents pointed to by the URCCMDADDR register are valid. The URC must set this bit to a zero when it completes reading the command and is ready to accept the next command. Note that the URC must only set this bit to a zero if it has enough internal storage to temporarily store the result of the next command. A write of zero to this field has no effect. See Section 4.13 for additional operational details.
14	**Interrupt when Ready — R/W.** The URCD must set this bit to a 1 if it wants the URC to set the *Ready for Command Interrupt* bit in the URCSTS register when it completes reading the command from the buffer pointed to by the URCCMDADDR register and is ready to accept the next command. In addition if the *Ready for Command Interrupt Enable* bit is set in the URCINTR register then the URC will generate an interrupt.
13	**Reserved — RO**. This bit is reserved and must be set to zero.
12:0	**Command Size — R/W.** This field indicates the size of the command (in bytes) that is present in the buffer pointed to by the URCCMDADDR register.

2.3.2 URCSTS — URC Status Register

Address: URCBase + (04h)
Default Value: 00010000h
Attribute RO, R/WC (field dependent)
Size 32 bits

This register indicates pending interrupts and various states of the URC. The URCD sets a bit to 0 in this register by writing a 1 to it. See Section 4.13.11.6 for additional information concerning URC interrupt conditions.

Table 2-11. URCSTS – URC Status Register Bit Definitions

Bit	Description
31:18	**Reserved — RO**. These bits are reserved and must be set to zero.
17	**Event Processing Status — RO.** 0=Default. This bit reports the current real status of the Event Processing. When this bit is set to zero then the status of the Event Processing is disabled. When this bit is set to a one then Event Processing is enabled, i.e. a valid event buffer address exists in the URCEVTADDR register and the URC is ready to place any events that may occur into the buffer. When the URC completes the event buffer and sets the *Event Ready* bit, it must reset this bit to a zero. In addition, when the URCD clears the *Run/Stop* bit in the URCCMD register to a zero, the URC must clear this bit to a zero in addition to setting the *Halted* bit.
16	**Halted — RO.** 1=Default. This bit is a zero whenever the *Run/Stop* bit is a one. The URC sets this bit to one after it has stopped executing as a result of the *Run/Stop* bit being set to 0, either by the URCD or by the URC hardware (e.g. internal error).
15:11	**Reserved — RO**. These bits are reserved and must be set to zero.
10	**Host System Error — R/WC.** The URC sets this bit to 1 when a serious error occurs during a host system access involving the URC module. In a PCI system, conditions that set this bit to 1 include PCI Parity error, PCI Master Abort, and PCI Target Abort. When this error occurs, the URC clears the *Active* bit in the URCCMD register to stop further execution of the commands.
9	**Event Ready (URCER) — R/WC.** 0=Default. The URC sets this bit to 1 when it completes writing one or more events to the event buffer pointed to by URCEVTADDR register. In addition, the URC must reset the *Event Processing Status* bit to a zero.
8	**Ready for Command Interrupt (URCRCI) — R/WC.** 0=Default. The URC sets this bit to 1 after it sets the *Active* bit to a zero in URCMD register and if the *Interrupt when Ready* bit in URCCMD register is set to one.
7:0	**Interrupt Source Identification (URCISI) — RO.** The URC sets this field to identify the source of the interrupt. Each bit is used to identify the interface (e.g. WHCI, NIC etc) on which an interrupt occurred. The ordering of the bits depends on the order that the UMC exposes its capabilities as defined in Section 2.2. For example if the WHCI capability is exposed first, then Bit 0 will indicate that an interrupt was caused by the WHC. This is a read only field and each bit must be reset by the URC when the original interrupt source is cleared. For example, in the case where the WHC is the source, the corresponding bit in this register will be reset (to zero) when the WUSBSTS register is cleared.

2.3.3 URCINTR — URC Interrupt Enable Register

Address: URCBase + (08h)
Default Value: 00000000h
Attributes R/W
Size 32 bits

This register enables and disables reporting of the corresponding interrupt to the URCD. When a bit in this register is set to a one and the corresponding interrupt is active, an interrupt is generated to the host. Interrupt sources that are disabled in this register (have a value of zero) still report interrupt conditions via the URCSTS to allow the URCD to poll for events.

Table 2-12. URCINTR - URC Interrupt Enable Register

Bit	Interrupt Source	Description
31:11	**Reserved**.	These bits are reserved and must be set to zero.
10	**Host System Error Enable**	When this bit is a one, and the *Host System Error* bit in the URCSTS register is a one, the URC will issue an interrupt. The interrupt is acknowledged by the URCD clearing the *Host System Error* bit.
9	**Event Ready Enable**	When this bit is a one, and the *Event Ready* bit in the URCSTS register is a one, the URC will issue an interrupt. The interrupt is acknowledged by the URCD clearing the *Event Ready* bit.
8	**Ready for Command Interrupt Enable**	When this bit is a one, and the *Ready for Command Interrupt* bit in the URCSTS register is a one, the URC will issue an interrupt. The interrupt is acknowledged by the URCD clearing the *Ready for Command Interrupt* bit.
7:0	**Interrupt Source Identification Enable**.	If a bit in this field is a one and the corresponding bit in the *Interrupt Source Identification* in the URCSTS register is a one, the URC will issue an interrupt.

Note: for all enable register bits, 1= Enabled, 0= Disabled

2.3.4 URCCMDADDR— URC Command Address Register

Address: URCBase + (10h)
Default Value: Undefined
Attribute R/W
Size 64 bits

This 64 bit register contains the address of the command buffer into which the URCD has placed the URC command it wants the URC to execute. See Section 3.1.1 for the actual contents in this buffer.

Table 2-13. URCCMDADDR – URC Command Address Register Bit Definitions

Bit	Description
63:2	**Command Buffer Address**. These bits correspond to memory address signals [63:2], respectively. This is the main memory address of the physically contiguous command buffer. The size of the command in this buffer is specified by the *Command Size* field in the URCCMD register.
1:0	**Reserved.** These bits are reserved and must be set to zero.

2.3.5 URCEVTADDR — URC Event Address Register

Address: URCBase + (18h)
Default Value: Undefined
Attribute R/W
Size 64 bits

This 64 bit register contains the main memory address of the event buffer. The event buffer must be 4K bytes in length and page aligned. See Section 3.1.2 for the actual contents in this buffer and Section 4.13 for the operational details of this register.

Table 2-14. URCEVTADDR – URC Event Address Register Bit Definitions

Bit	Description
63:12	**Event Buffer Address**. This is the location of the physically contiguous event buffer. The URC does not modify this field.
11:0	**Current Offset.** The URC updates this field by incrementing it by the event size so that this register points to the buffer location for the next event. This field finally indicates the total size of the events written to the event buffer when the buffer is returned to the URCD. The URCD must set this field to zero when it writes an event buffer address to this register.

2.4 Wireless USB Host Controller Capability and Operational Registers

This section defines the WHC Capability and Operational registers. The Capability registers specify the limits, restrictions and capabilities of the host controller implementation. The operational registers are used by the WHCD to control and monitor the operational state of the host controller. The location of the WHC registers is determined by parsing the WHCI Capability in the list of UMC capabilities (see Section 2.2.2). All the WHC registers must be DWord aligned and the WHCD should read and write the WHC operational registers using only DWord accesses except for the WHCIVERSION register which is Word accessed. Figure 2-2 shows how the WHC exposes the registers defined in this specification.

Table 2-15. WHC Capability and Operational Registers

Offset	Size	Mnemonic	Register Name	Power Well	Section
00h	2	WHCIVERSION	Interface Version Number	Core	2.4.1
04h	4	WHCSPARAMS	Structural Parameters	Core	2.4.2
08h	4	WUSBCMD	WUSB Command	Aux	2.4.3
0Ch	4	WUSBSTS	WUSB Status	Core	2.4.4
10h	4	WUSBINTR	WUSB Interrupt Enable	Core	2.4.5
14h	4	WUSBGENCMDSTS	WUSB Generic Command Status	Core	2.4.6
18h	4	WUSBGENCMDPARAMS	WUSB Generic Command Parameters	Core	2.4.7
20h	8	WUSBGENADDR	WUSB Generic Address	Core	2.4.8
28h	8	WUSBASYNCLISTADDR	WUSB Current Asynchronous List Address	Core	2.4.9
30h	8	WUSBDNTSBUFADDR	WUSB DNTS Buffer Address	Core	2.4.10
38h	8	WUSBDEVICEINFOADDR	WUSB Device Info Address	Core	2.4.11
40h	4	WUSBSETSECKEYCMD	WUSB Set Security Key Command	Core	2.4.12
44h	4	WUSBTKID	WUSB Temporal Key ID	Core	2.4.13
48h	16	WUSBSECKEY	WUSB Security Key	Core	2.4.14

Table 2-15. WHC Capability and Operational Registers (cont.)

Offset	Size	Mnemonic	Register Name	Power Well	Section
58h	8	WUSBPERIODICLISTBASE	WUSB Periodic List Base	Core	2.4.15
60h	4	WUSBMASINDEX	WUSB MAS Index	Core	2.4.16
64h	4	WUSBDNTSCTRL	WUSB DNTS Control	Core	2.4.17
68h	4	WUSBTIME	WUSB Channel Time	Aux	2.4.18
6Ch	4	WUSBBPST	WUSB Beacon Period Start Time	Core	2.4.19
70h	16	WUSBDIBUPDATED	WUSB Device Information Buffer Updated	Core	2.4.20

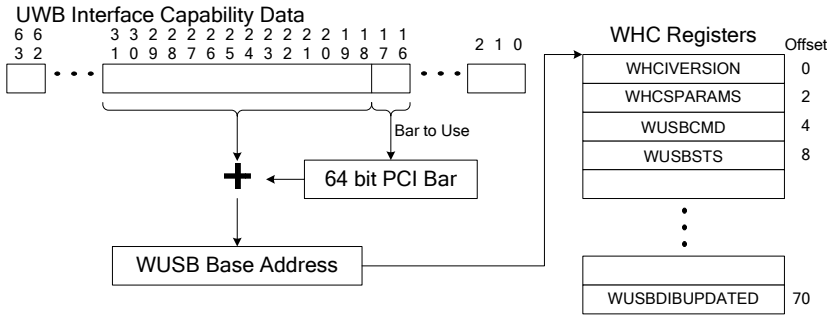


Figure 2-2. WHC Register Organization

2.4.1 WHCIVERSION — WUSB Host Controller Interface Version Number

Address: WUSBBase + (00h)
Default Value: 0095h
Attribute RO
Size: 16 bits

This is a two-byte register containing a BCD encoding of the WHCI revision number supported by this WHC implementation. The most significant byte of this register represents a major revision and the least significant byte is the minor revision.

2.4.2 WHCSPARAMS — WHC Structural Parameters

Address: WUSBBase + (04h)
Default Value Implementation Dependent
Attribute RO
Size: 32 bits

This is a set of fields that are structural parameters: e.g. number of devices supported by the WHC.

Table 2-16.WHCSPARAMS — WUSB Host Controller Structural Parameters

Bit	Description
31:24	**Reserved.** These bits are reserved and must be set to zero.
23:16	**N_MMC_IES.** This field specifies the number of MMC IE blocks that this WHC can support at the same time. Valid values are in the range of 3H to 10H. Any other value in this field is undefined.
15:8	**N_KEYS.** This field specifies the number of security key tables that this WHC can store at the same time. Valid values are in the range of 3H to 80H. A zero, one or two in this field is undefined.
7	**Reserved.** This bit is reserved and must be set to zero.
6:0	**N_DEVICES.** This field specifies the number of Wireless USB devices that this WHC can support at the same time. Valid values are in the range of 2H to 7FH. A zero or one in this field is undefined.

2.4.3 WUSBCMD — WUSB Command Register

Address: WUSBBase + (08h)
Default Value: 00000000h
Attribute RO, R/W (field dependent)
Size 32 bits

The Command Register indicates the command to be executed by the WHC. Writing to the register causes a command to be executed.

Note that the WHCD must take care to preserve all the bits in this register that it does not want to modify when it performs a write operation to this register.

Table 2-17. WUSBCMD – WUSB Command Register Bit Definitions

Bit	Description
31:24	**Reserved — RO.** These bits are reserved and must be set to zero.
23:16	**Broadcast Cluster ID — R/W.** This field is used by the WHCD to set the WUSB broadcast DEVID.
15:13	**Reserved — RO.** These bits are reserved and must be set to zero.
12	**Asynchronous Schedule QSet Removed — R/W.** Default 0b. This field is set to a 1B by the WHCD to indicate that the WHCD has removed a Queue Set from the Asynchronous schedule. The WHCD must also set the *Asynchronous Schedule Updated* bit to a 1B when it sets this bit. The WHC must set this bit to a 0B when it sets the *Asynchronous Schedule Updated* to a 0B. See Section 4.6 for operational details.
11	**Periodic Schedule QSet Removed — R/W.** Default 0b. This field is set to a 1B by the WHCD to indicate that the WHCD has removed a Queue Set from the Periodic schedule. The WHCD must also set the *Periodic Schedule Updated* bit to a 1B when it sets this bit. The WHC must set this bit to a 0B when it sets the *Periodic Schedule Updated* to a 0B. See Section 4.6 for operational details.
10:8	**WUSB Stream Index (WUSBSI) — R/W.** This field is used by the WHCD to set the WUSB stream index.

Table 2-17. WUSBCMD – WUSB Command Register Bit Definitions (cont.)

Bit	Description
7	**Interrupt on Asynch Schedule Synched Doorbell — R/W.** Default 0b. This bit is used as a doorbell by the WHCD to tell the WHC to issue an interrupt when its internal cached state of the asynchronous schedule is synched with the changes made to the asynchronous schedule. The WHCD must write a 1 to this bit to *ring* the doorbell. When the WHC has evicted all appropriate cached schedule state, it sets the *Interrupt on Asynch Schedule Synched* status bit in the WUSBSTS register. If the *Interrupt on Asynch Schedule Synched Enable* bit in the WUSBINTR register is a one then the WHC will assert an interrupt. See Section 4.6 for operational details. The WHC sets this bit to a zero after it has set the *Interrupt on Asynch Schedule Synched* status bit in the WUSBSTS register to a one. The WHCD should not write a one to this bit when the asynchronous schedule is disabled. Doing so will yield undefined results.
6	**Interrupt on Periodic Schedule Synched Doorbell — R/W.** Default 0b. This bit is used as a doorbell by the WHCD to tell the WHC to issue an interrupt when its internal cached state of the periodic schedule is synched with the changes made to the periodic schedule. The WHCD must write a 1 to this bit to *ring* the doorbell. When the WHC has evicted all appropriate cached schedule state, it sets the *Interrupt on Periodic Schedule Synched* status bit in the WUSBSTS register. If the *Interrupt on Periodic Schedule Synched Enable* bit in the WUSBINTR register is a one then the WHC will assert an interrupt. See Section 4.6 for operational details. The WHC sets this bit to a zero after it has set the *Interrupt on Periodic Schedule Synched* status bit in the WUSBSTS register to a one. The WHCD should not write a one to this bit when the periodic schedule is disabled. Doing so will yield undefined results.
5	**Asynchronous Schedule Updated — R/W.** Default 0b. This field is set to a 1B by the WHCD to indicate that the WHCD has updated the Asynchronous Schedule. The WHC will reset this bit (to a 0B) to acknowledge the change. The WHCD must not update the Asynchronous Schedule while this bit is set. See Section 4.6 for operational details.
4	**Periodic Schedule Updated — R/W.** Default 0b. This field is set to a 1B by the WHCD to indicate that the WHCD has updated the Periodic Schedule. The WHC will reset this bit (to a 0B) to acknowledge the change. The WHCD must not update the Periodic Schedule while this bit is set. See Section 4.6 for operational details.
3	**Asynchronous Schedule Enable — R/W.** Default 0b. This bit controls whether the WHC skips processing the Asynchronous Schedule. **Value** — **Description** 0b — Do not process the Asynchronous Schedule 1b — Use the WUSBASYNCLISTADDR register to access the Asynchronous Schedule. This bit must only be modified if its current value is equal to the value of the *Asynchronous Schedule Status* bit in the WUSBSTS register.
2	**Periodic Schedule Enable — R/W.** Default 0b. This bit controls whether the WHC skips processing the Periodic Schedule. **Value** — **Description** 0b — Do not process the Periodic Schedule 1b — Use the WUSBPERIODICLISTBASE register to access the Periodic Schedule. This bit must only be modified if its current value is equal to the value of the *Periodic Schedule Status* bit in the WUSBSTS register.

Table 2-17. WUSBCMD – WUSB Command Register Bit Definitions (cont.)

Bit	Description
1	**Host Controller Reset (WHCRESET) — R/W.** This control bit is used by the WHCD to reset the WHC. When the WHCD writes a one to this bit, the WHC resets its internal pipelines, timers, counters, state machines, etc. to their initial values. Any transaction currently in progress on the Wireless USB channel is immediately terminated. PCI Configuration registers and UWB Radio Controller Registers are not affected by this reset. All operational registers are set to their initial values. The WHCD must reinitialize the WHC as described in Section 4.1 in order to return the WHC to an operational state. This bit is set to zero by the WHC when the reset process is complete. The WHCD cannot terminate the reset process early by writing a zero to this register. The WHCD should not set this bit to a one when the *HCHalted* bit in the WUSBSTS register is a zero. Attempting to reset an actively running WHC will result in undefined behavior.
0	**Run/Stop (RS) —R/W.** Default 0b. 1=Run. 0=Stop. When set to a 1, the WHC proceeds with execution of the schedule. The WHC continues execution as long as this bit is set to a 1. When this bit is set to 0, the WHC completes the current and any actively pipelined transactions on the Wireless USB channel and then halts. The *HCHalted* bit in the status register indicates when the WHC has finished its pending pipelined transactions and has entered the stopped state. In addition, the WHC will stop processing any commands issued through the generic command mechanism, will not support setting/erasing security keys and will not process any device notifications.

2.4.4 WUSBSTS — WUSB Status Register

Address: WUSBBase + (0Ch)
Default Value: 00001000h
Attribute RO, R/WC (field dependent)
Size 32 bits

This register indicates pending interrupts and various states of the WHC. The status resulting from a transaction on the Wireless USB channel is not indicated in this register. The WHCD sets a bit to 0 in this register by writing a 1 to it. See Section 4.12 for additional information concerning WHC interrupt conditions.

Table 2-18. WUSBSTS – WUSB Status Register Bit Definitions

Bit	Description
31:16	**Reserved — RO**. These bits are reserved and must be set to zero.
15	**Asynchronous Schedule Status — RO.** 0=Default. This bit reports the current real status of the Asynchronous Schedule. If this bit is a zero then the status of the Asynchronous Schedule is disabled. If this bit is a one then the status of the Asynchronous Schedule is enabled. The WHC is not required to *immediately* disable or enable the Asynchronous Schedule when the WHCD transitions the *Asynchronous Schedule Enable* bit in the WUSBCMD register.
14	**Periodic Schedule Status — RO.** 0=Default. The bit reports the current real status of the Periodic Schedule. If this bit is a zero then the status of the Periodic Schedule is disabled. If this bit is a one then the status of the Periodic Schedule is enabled. The WHC is not required to *immediately* disable or enable the Periodic Schedule when the WHCD transitions the *Periodic Schedule Enable* bit in the WUSBCMD register.
13	**DNTS Schedule Status — RO.** 0=Default. The bit reports the current real status of the DNTS Schedule. If this bit is a zero then the status of the DNTS Schedule is disabled. If this bit is a one then the status of the DNTS Schedule is enabled. The WHC is not required to *immediately* disable or enable the DNTS Schedule when the WHCD transitions the *Active* bit in the WUSBDNTSCTRL register.

Table 2-18. WUSBSTS – WUSB Status Register Bit Definitions (cont.)

Bit	Description
12	**HCHalted — RO.** 1=Default. This bit is a zero whenever the *Run/Stop* bit is a one. The WHC sets this bit to one after it has stopped executing as a result of the *Run/Stop* bit being set to 0, either by the WHCD or by the WHC hardware (e.g. internal error).
11:10	**Reserved — RO**. These bits are reserved and must be set to zero.
9	**Generic Command Completion — R/WC.** The WHC sets this bit to 1 when it completes execution of a command specified in the WUSBGENCMDSTS register.
8	**Channel Time Rollover — R/WC.** The WHC sets this bit to 1 when the Wireless USB Channel Time value in WUSBTIME register has wrapped around to zero.
7	**Device Notification Buffer Overflow — R/WC.** The WHC sets this bit to 1 when there is no device notification block available to place a device notification received in the Device Notification Buffer. The WHC clears the *Active* bit in the WUSBDNTSCTRL register and stops DNTS scheduling.
6	**BPST Adjustment Changed — R/WC.** The WHC sets this bit to 1 when the adjustment value of the Beacon Period Start Time has been changed.
5	**Host System Error — R/WC.** The WHC sets this bit to 1 when a serious error occurs during a host system access involving the WHC module. In a PCI system, conditions that set this bit to 1 include PCI Parity error, PCI Master Abort, and PCI Target Abort. When this error occurs, the WHC clears the *Run/Stop* bit in the Command register to prevent further execution of the scheduled TDs.
4	**Interrupt on Asynch Schedule Synched — R/WC.** 0=Default. The WHCD can force the WHC to issue an interrupt the next time the WHC synchs its internal asynchronous schedule cache with the actual asynchronous schedule by writing a one to the *Interrupt on Asynch Schedule Synched Doorbell* bit in the WUSBCMD register. This status bit indicates the assertion of that interrupt source.
3	**Interrupt on Periodic Schedule Synched — R/WC.** 0=Default. The WHCD can force the WHC to issue an interrupt the next time the WHC synchs its internal periodic schedule cache with the actual periodic schedule by writing a one to the *Interrupt on Periodic Schedule Synched Doorbell* bit in the WUSBCMD register. This status bit indicates the assertion of that interrupt source.
2	**WUSB DNTS Interrupt (WUSBDNTSINT) — R/WC.** The Host Controller sets this bit to a one when one or more device notification messages are received in a DNTS period.
1	**WUSB Error Interrupt (WUSBERRINT) — R/WC.** The WHC sets this bit to 1 when completion of a Wireless USB transaction results in an error condition (e.g., error counter underflow). If the Transfer Descriptor (TD) on which the error interrupt occurred also had its IOC bit set, both this bit and WUSBINT bit are set. See Section 4.12.1 for a list of the Wireless USB transaction errors that will result in this bit being set to a one.
0	**WUSB Interrupt (WUSBINT) — R/WC.** The WHC sets this bit to 1 on the completion of a Wireless USB transaction, which results in the retirement of a TD that had its IOC bit set. The WHC also sets this bit to 1 when a qTD was completed by receiving the last packet of a transfer.

2.4.5 WUSBINTR — WUSB Interrupt Enable Register

Address: WUSBBase + (10h)
Default Value: 00000000h
Attributes R/W
Size 32 bits

This register enables and disables reporting of the corresponding interrupt to the WHCD. When a bit is set and the corresponding interrupt is active, an interrupt is generated to the host. Interrupt sources that are disabled in this register still appear in the WUSBSTS to allow the WHCD to poll for events.

Table 2-19. WUSBINTR - WUSB Interrupt Enable Register Bit Definitions

Bit	Interrupt Source	Description
31:10	**Reserved**.	These bits are reserved and must be set to zero.
9	**Generic Command Completion Enable**	When this bit is a one, and the *Generic Command Completion* bit in the WUSBSTS register is a one, the WHC will issue an interrupt. The interrupt is acknowledged by the WHCD clearing the *Generic Command Completion* bit.
8	**Channel Time Rollover Enable**	When this bit is a one, and the *Channel Time Rollover* bit in the WUSBSTS register is a one, the WHC will issue an interrupt. The interrupt is acknowledged by the WHCD clearing the *Channel Time Rollover* bit.
7	**Device Notification Buffer Overflow Enable**	When this bit is a one, and the *Device Notification Buffer Overflow* bit in the WUSBSTS register is a one, the WHC will issue an interrupt. The interrupt is acknowledged by the WHCD clearing the *Device Notification Buffer Overflow* bit.
6	**BPST Adjustment Changed Enable**	When this bit is a one, and the *BPST Adjustment Changed* bit in the WUSBSTS register is a one, the WHC will issue an interrupt. The interrupt is acknowledged by the WHCD clearing the *BPST Adjustment Changed* bit.
5	**Host System Error Enable**	When this bit is a one, and the *Host System Error* bit in the WUSBSTS register is a one, the host controller will issue an interrupt. The interrupt is acknowledged by software clearing the *Host System Error* bit.
4	**Interrupt on Asynch Schedule Synched Enable**	When this bit is a one, and the *Interrupt on Asynch Schedule Synched* bit in the WUSBSTS register is a one, the WHC will issue an interrupt. The interrupt is acknowledged by the WHCD clearing the *Interrupt on Asynch Schedule Synched* bit.
3	**Interrupt on Periodic Schedule Synched Enable**	When this bit is a one, and the *Interrupt on Periodic Schedule Synched* bit in the WUSBSTS register is a one, the WHC will issue an interrupt. The interrupt is acknowledged by the WHCD clearing the *Interrupt on Periodic Schedule Synched* bit.
2	**WUSB DNTS Interrupt Enable.**	When this bit is a one, and the *WUSB DNTS Interrupt* bit in the WUSBSTS register is a one, the WHC will issue an interrupt. The interrupt is acknowledged by the WHCD clearing the *WUSB DNTS Interrupt* bit.

Table 2-19. WUSBINTR - WUSB Interrupt Enable Register Bit Definitions (cont)

Bit	Interrupt Source	Description
1	**WUSB Error Interrupt Enable.**	When this bit is a one, and the WUSBERRINT bit in the WUSBSTS register is a one, the WHC will issue an interrupt. The interrupt is acknowledged by the WHCD clearing the *WUSBERRINT* bit.
0	**WUSB Interrupt Enable.**	When this bit is a one, and the WUSBINT bit in the WUSBSTS register is a one, the WHC will issue an interrupt. The interrupt is acknowledged by the WHCD clearing the *WUSBINT* bit.

Note: for all enable register bits, 1= Enabled, 0= Disabled

2.4.6 WUSBGENCMDSTS — WUSB Generic Command Status Register

Address: WUSBBase + (14h)
Default Value: 00000000h
Attributes R/W
Size 32 bits

This register is used by the WHCD when it wants to modify the type of data identified by the *Command Type* field in the register. The register bit definitions of the WUSBGENCMDPARAMS register depends on value of the *Command Type* field. See Section 4.8 for the details on the operational model of this register.

Table 2-20. WUSBGENCMDSTS - WUSB Generic Command Status Register Bit Definitions

Bit	Description
31:24	**Status**. This field is used by the WHC to communicate individual command execution states back to the WHCD. This field contains the status of the last command performed. The bit encodings are: **Bit** — **Meaning** 7 — **Active**. Set to 1 by the WHCD to enable the execution of the command by the WHC. The WHC will reset this bit when it completes execution of the command 6:1 — **Reserved**. These bits are reserved and must be set to zero 0 — **Error**. The WHC encountered an error while processing the command The WHCD can reset the *Error* bit by writing a zero to that bit.
23	**Interrupt on Complete — R/W.** If this bit is set then the WHC will set the *Generic Command Completion* bit in the WUSBSTS register when it completes this command. In addition if the *Generic Command Completion Enable* bit is set in the WUSBINTR register then the WHC will generate an interrupt.
22:8	**Reserved**. These bits are reserved and must be set to zero.

Table 2-20. WUSBGENCMDSTS - WUSB Generic Command Status Register Bit Definitions (cont)

Bit	Description
7:0	**Command Type**. The type of command that the WHCD wants the WHC to perform. The format of the bits in the WUSBGENCMDPARAMS register depends on the value of this field. **Value** — **Command Type** 0 — Reserved. 1 — Add MMC IE (also uses WUSBGENADDR register) 2 — Remove MMC IE 3 — Set WUSB MAS (also uses WUSBGENADDR register) 4 — Channel Stop 5 — Remote Wake Poll Enable 6-255 — Reserved

2.4.7 WUSBGENCMDPARAMS — WUSB Generic Command Parameters Register

Address: WUSBBase + (18h)
Default Value: Undefined
Attributes R/W
Size 32 bits

This register is used in conjunction with the WUSBGENCMDSTS register when it wants to modify the type of data identified by the *Command Type* field in the WUSBGENCMDSTS register. The bit definitions for this register depend on the type of command.

2.4.7.1 WUSB Generic Command Add MMC IE

If the *Command Type* field in the WUSBGENCMDSTS register is set to **Add MMC IE**, then this register is used to add or modify the list of IEs that are sent in an MMC. The actual IE block to be added or modified is in the buffer referenced by the value in the WUSBGENADDR register. Note that an IE Block consists of one or more IEs.

Table 2-21. WUSB Generic Command Parameters Register Bit Definitions for Add MMC IE

Bit	Description
31:24	**Interval.** The field specifies the period at which the WHC must include this IE block in an MMC. This field is expressed in milliseconds. **Value** — **Operational Requirement** 0 — The WHC must send this IE block in every MMC. The value in the *Repeat Count* field must be ignored. 1-254 — The WHC must send this IE block every *Interval* period. 255 — This value must be treated as an infinite period. The WHC must only send this IE block the number of times specified in the *Repeat Count* field. If this field is set to 255, then the WHC will complete this command only after sending the IE the number of times specified in the *Repeat Count* field.
23:16	**Repeat Count**. This field specifies the number of consecutive MMCs that this IE block must be sent in during each interval. This field is ignored when *Interval* is set to zero.
15:8	**Size of IE Block.** This field specifies the size of the IE block pointed to by the WUSBGENADDR register.
7:4	**Reserved**. These bits are reserved and must be set to zero.

Table 2-21. WUSB Generic Command Parameters Register Bit Definitions for Add MMC IE (cont)

Bit	Description
3:0	**IE_HANDLE**. This handle uniquely identifies an IE block. This is the handle to be used by the WHCD when it needs to modify or remove this IE block from subsequent MMCs. The total number of IE blocks supported by the WHC is specified by *N_MMC_IES* field in WHCSPARAMS register. The WHCD must ensure that the value is in the range of 0 to *N_MMC_IES* – 1. If an invalid value is set, the WHC operation is undefined.

2.4.7.2 WUSB Generic Command Remove MMC IE

If the *Command Type* field in the WUSBGENCMDSTS register is set to **Remove MMC IE**, then this register is used to specify the IE block to remove from the list of IEs that are sent in an MMC. The WUSBGENADDR register is not used with this command.

Table 2-22. WUSB Generic Command Parameters Register Bit Definitions for Remove MMC IE

Bit	Description
31:4	**Reserved**. These bits are reserved and must be set to zero.
3:0	**IE_HANDLE**. This handle specifies the IE block to be removed. The total number of IE blocks supported by the WHC is specified by *N_MMC_IES* field in WHCSPARAMS register. The WHCD must ensure that the value is in the range of 0 to *N_MMC_IES* – 1. If an invalid value is set, the WHC operation is undefined.

2.4.7.3 WUSB Generic Command Set WUSB MAS

If the *Command Type* field in the WUSBGENCMDSTS register is set to **Set WUSB MAS**, then this register is not used. This command is used to set/update the currently available Media Access Slots that the WHC can use. The WUSB MAS is a 32 byte array of 256 entries, each of which corresponds to one of the 256 MASs within a superframe. The zero entries identify MASs that cannot be used by the WHC while nonzero entries identify MASs in which a WHC may perform Wireless USB transactions. The actual 32 byte WUSB MAS array to be set or updated will be present at the buffer pointed to by the WUSBGENADDR register.

Table 2-23. WUSB Generic Command Parameters Register Bit Definitions for Set WUSB MAS

Bit	Description
31:0	**Reserved**. These bits are reserved and must be set to zero.

2.4.7.4 WUSB Generic Command Channel Stop

If the *Command Type* field in the WUSBGENCMDSTS register is set to **Channel Stop**, then this register is used to request the WHC to stop the Wireless USB Channel. On reception of this command, the WHC must stop the Wireless USB channel at the specified time. The WHC must send a Channel Stop IE in its MMC packets before stopping the channel (i.e. terminating transmission of MMCs). The WUSBGENADDR register is not used with this command.

Table 2-24. WUSB Generic Command Parameters Register Bit Definitions for Channel Stop

Bit	Description
31	**Cancel**. If this bit is set then the WHC must cancel the previously requested channel stop operation.
30:24	**Attributes**. This field is a bitmap of control parameters that effect the operation the channel stop sequencing or format of the Channel Stop IE. This field has the following encoding: **Bit** **Description** 6:1 **Reserved.** These bits are reserved and must be set to zero. 0 **Remote Wake.** The value of this bit is directly reflected (copied) into the *bmAttributes.Remote Wakeup* bit in the Channel Stop IE transmitted by the WHC during the Wireless USB channel shutdown sequence.
23:0	**Stop Time**. This field indicates the WUSB Channel Time at which the WHC must stop the WUSB channel. This field is only valid when the *Cancel* bit is not set.

2.4.7.5 WUSB Generic Command Remote Wake Poll Enable

When the *Command Type* field in the WUSBGENCMDSTS register is set to **Remote Wake Poll Enable**, the format of this register is defined in Table 2-25.

Table 2-25. WUSB Generic Parameters Register Bit Definitions for Remote Wake Poll Enable

Bit	Description
31	**Enabled.** The value of this field enables/disables the WHC polling for remote wake functionality. A value of zero (0B) disables remote wake polling. A value of one (1B) enables remote wake polling.
30:12	**Reserved**. This field is reserved and must be set to zero by the WHCD.
11:8	**DNTS Window Width.** The value of this field indicates the minimum number of notification slots the WHC must provide in each MMC that is used to poll for remote wake.
7:0	**Remote Wake Poll Interval**. The value of this field indicates the interval at which the WHC will poll for remote wake notifications. The units of this field are number of WiMedia super-frames. This field must be set to a value in the range [0AH (10) to 3CH (60)]. The results are not defined for any value outside of this range. This field is ignored if the value of the *Enabled* field (above) is a zero (0B).

This command has no externally visible effects on the operation of the WHC while the channel is active (which coincides with the run/stop bit value of 1B). It does effect the operation of the WHC when the value of the run/stop bit is a zero. See Section 4.11 for full operational details.

The first interval begins at the first superframe boundary after the WHC has reported being halted (*HCHalted* bit set to one in the WUSBSTS register).

2.4.8 WUSBGENADDR — WUSB Generic Address Register

Address: WUSBBase + (20h)
Default Value: Undefined
Attribute: Read/Write (Writes must be DWord Writes)
Size: 64 bits

This 64-bit register contains the address of the data that the WHCD wants to set. The type and format of this data is dependent on the contents of the WUSBGENCMD register. The buffer referred to by the value in this register must be physically contiguous, less than 4K in length and cannot cross a page boundary.

Table 2-26. WUSBGENADDR — WUSB Generic Address Register Bit Definitions

Bit	Description
63:0	**Address Pointer.** These bits correspond to memory address signals [63:0], respectively.

2.4.9 WUSBASYNCLISTADDR — WUSB Current Asynchronous List Address Register

Address: WUSBBase + (28h)
Default Value: Undefined
Attribute: R/W, RO (field dependent, Writes must be DWord Writes)
Size: 64 bits

This 64-bit register contains the address of the next asynchronous Queue Head (QHead). Bits [5:4] and Bit [0] of this register cannot be modified by the WHCD and will always return a zero when read. The memory structure referenced by this physical memory pointer is assumed to be 64-byte aligned. This register also contains the number of qTDs (see Section 3.2.4) following the QHead pointed to by the *Link Pointer* field.

Table 2-27. WUSBASYNCLISTADDR — WUSB Current Asynchronous List Address Register Bit Definitions

Bit	Description
63:6	**Link Pointer (LP) — R/W.** These bits correspond to memory address signals [63:6], respectively. This field may only reference a QHead, see Section 3.2.6.
5:4	**Reserved — RO.** These bits are reserved and must be set to zero.
3:1	**Number of qTDs in Queue Set (nTDs) — R/W.** This field indicates to the WHC the number of qTDs associated with the referenced QHead. The actual number of qTDs is one more than the raw value in this field.
0	**Reserved — RO.** This bit is reserved and must be set to zero.

2.4.10 WUSBDNTSBUFADDR — WUSB DNTS Buffer Address Register

Address: WUSBBase + (30h)
Default Value: Undefined
Attribute: R/W, RO (field dependent, Writes must be DWord Writes)
Size: 64 bits

This 64-bit register contains the address pointer to the Device Notification Buffer. This buffer must be 4K bytes in length and page aligned. The WHC will write a device notification it receives from the Wireless USB beginning at the buffer location referenced by this register and update the *Current Offset* field so that this register indicates the buffer location for the next device notification message. If a notification is received and the WHC does not have enough buffer space, it will discard the data, stop DNTS scheduling by clearing the *Active* bit in WUSBDNTSCTRL register and set the *Device Notification Buffer Overflow* bit in the WUSBSTS register. See Section 3.2.9 for the detailed format of the Device Notification Buffer.

Table 2-28. WUSBDNTSBUFADDR — WUSB DNTS Buffer Address Register Bit Definitions

Bit	Description
63:12	**Address Pointer — R/W.** These bits correspond to memory address signals [63:12], respectively. This field only references a page aligned Device Notification Buffer.
11:6	**Current Offset — R/W.** This field indicates the current offset in the Device Notification Buffer in 64 bytes units. The WHCD must set this field to zero. The WHC increments this field by one when it writes a device notification to the buffer.
5:0	**Reserved — RO.** These bits are reserved and must be set to zero.

2.4.11 WUSBDEVICEINFOADDR — WUSB Device Info Address Register

Address: WUSBBase + (38h)
Default Value: Undefined
Attribute: R/W, RO (field dependent, Writes must be DWord Writes)
Size: 64 bits

This 64-bit register contains the address of the Device Information buffer in main memory. This buffer must be 64 byte aligned and can be a maximum of 8K in length. This buffer must be physically contiguous. The WHC will use this information to determine whether it can perform a transaction to a particular device based on its device availability information. The Device Address is also stored in this location. In addition, the Device Info Buffer stores the Security Key Index to be used to encrypt/decrypt data to and/or from the device.

The WHCD is only allowed to update the contents of this buffer when the corresponding bit in the WUSBDIBUPDATED register is set to zero.

Table 2-29. WUSBDEVICEINFOADDR — WUSB Device Info Address Register Bit Definitions

Bit	Description
63:6	**Address Pointer (LP) — R/W.** These bits correspond to memory address signals [63:6], respectively. This field may only reference a 64 byte aligned Device Information Buffer, see Section 3.2.8.
5:0	**Reserved — RO.** These bits are reserved and must be set to zero.

2.4.12 WUSBSETSECKEYCMD — WUSB Set Security Key Command Register

Address: WUSBBase + (40h)
Default Value: 00000000h
Attribute: R/W, RO (field dependent)
Size: 32 bits

The Set Security Key Command Register is used to set or erase the encryption key. The WHC gets the encryption key from the 128-bit WUSBSECKEY register.

Table 2-30. WUSBSETSECKEYCMD — WUSB Set Security Key Command Register Bit Definitions

Bit	Description
31	**Set Encryption Key Bit — R/W.** When the WHCD writes a one to this bit, the WHC will read the contents of the WUSBSECKEY register and use it as the encryption key for the 24 bit TKID present in the WUSBTKID register. The WHC will reset this bit to zero when it has successfully updated the Key Table. The results of writes to this register when this bit is set are undefined.
30	**Erase Encryption Key Bit — R/W.** When the WHCD writes a one to this bit, the WHC must erase the encryption key that it currently has stored in the Key Table at the given *Key Table Index*. The WHC will reset this bit to zero when it has successfully erased the encryption key.
29:9	**Reserved — RO.** These bits are reserved and must be set to zero.
8	**Group Key — R/W.** When the WHCD writes a one to this bit, the WHC must begin using the key value in the WUSBSECKEY register as the new Group Key in subsequent transmissions.
7:0	**Key Table Index — R/W.** This field specifies the index within the Key Table (in the WHC) where the new key is to be set (if this is a Set operation) or the index of the Key to be erased (if this is a Erase operation). The total number of keys that this WHC can store is specified by *N_KEYS* field in WHCSPARAMS register. The WHCD must ensure that the value is in the range of 0 to *N_KEYS* – 1. If an invalid value is set, the WHC operation is undefined.

2.4.13 WUSBTKID – WUSB Temporal Key ID

Address: WUSBBase + (44h)
Default Value: Undefined
Attribute: Read/Write
Size: 32 bits

The lower 24bits of this register contains the TKID to be used with the encryption key from the 128-bit WUSBSECKEY register. This register is only valid when the *Set Encryption Key* bit in the WUSBSETSECKEYCMD register is set.

2.4.14 WUSBSECKEY — WUSB Security Key

Address: WUSBBase + (48h)
Default Value: Undefined
Attribute: Read/Write (Writes must be DWord Writes)
Size: 128 bits

This register contains the 128-bit encryption key to be set. This register is only valid when *Set Encryption Key* bit in the WUSBSETSECKEYCMD register is set.

2.4.15 WUSBPERIODICLISTBASE — WUSB Periodic List Base Register

Address: WUSBBase + (58h)
Default Value: Undefined
Attribute: R/W, RO (field dependent, Writes must be DWord Writes)
Size: 64 bits

This 64-bit register contains the address of the Periodic Zone List buffer. The memory structure referenced by this physical memory pointer is assumed to be 128-byte aligned.

Table 2-31. WUSBPERIODICLISTBASE — WUSB Periodic List Base Register Bit Definitions

Bit	Description
63:7	**Address Pointer — R/W.** These bits correspond to memory address signals [63:7], respectively. This field references a 128-byte aligned buffer containing an array of link pointers. See Section 3.1.
6:0	**Reserved — RO.** These bits are reserved and must be set to zero.

2.4.16 WUSBMASINDEX — WUSB MAS Index Register

Address: WUSBBase + (60h)
Default Value: Undefined
Attribute: RO
Size: 32 bits

This read only register contains the 8 bit index value indicating the current MAS location in the superframe. The WHC resets this register to zero at the beginning of a superframe and increments it by one whenever it crosses a MAS boundary.

Table 2-32. WUSBMASINDEX — WUSB MAS Index Register Bit Definitions

Bit	Description
31:8	**Reserved.** These bits are read only and must be set to zero.
7:0	**MAS Index.** This field indicates the current MAS location in the superframe.

2.4.17 WUSBDNTSCTRL — WUSB DNTS Control Register

Address: WUSBBase + (64h)
Default Value: 00000000h
Attribute: R/W, RO (field dependent)
Size: 32 bits

This register is used to control the schedule of the Device Notification Time Slot (DNTS). The WHC starts DNTS scheduling when the WHCD sets the *Active* bit in this register.

Table 2-33. WUSBDNTSCTRL — WUSB DNTS Control Register Bit Definitions

Bit	Description
31	**Active (A) — R/W.** If this bit is set then the WHC will enable the scheduling of Device Notification Time Slot (DNTS) by including a W_{DNTS}CTA. If this bit is not set then the WHC does not schedule any DNTS. The WHC will reset this bit when it cannot place a received device notification message in the buffer and sets *Device Notification Buffer Overflow* bit in WUSBSTS register. This bit must only be modified if its current value is equal to the value of the *DNTS Schedule Status* bit in the WUSBSTS register.
30:16	**Reserved — RO.** These bits are reserved and must be set to zero.
15:8	**Interval — R/W.** This field specifies an interval value in milliseconds that the WHC must use to schedule a DNTS. If this value is set to zero then the WHC must schedule a DNTS in every transaction group.
7:0	**Number of Slots — R/W.** This field specifies the width of the DNTS (number of notification slots). This value is used as the *bNumSlots* field in the W_{DNTS}CTA.

2.4.18 WUSBTIME — WUSB Channel Time Register

Address: WUSBBase + (68h)
Default Value: 00000000h
Attribute: RO
Size: 32 bits

This read only register indicates the current 24-bit Wireless USB channel time.

Table 2-34. WUSBTIME — WUSB Channel Time Register Bit Definitions

Bit	Description
31:24	**Reserved.** These bits are read only and must be set to zero.
23:0	**Channel Time.** This field indicates the current 24-bit Wireless USB channel time. See Section 4.2 for a detailed explanation of this field. The WHC must keep updating this field while the *Run/Stop* bit in WUSBCMD register is set or at the remote wake poll interval (see Section 4.11.3), but it is not required to start with the value zero. When this field has wrapped around to zero, the WHC must set the *Channel Time Rollover* bit in WUSBSTS register and generate an interrupt if the *Channel Time Rollover Enable* bit in the WUSBINTR register is set.

2.4.19 WUSBBPST — WUSB Beacon Period Start Time Register

Address: WUSBBase + (6Ch)
Default Value: 00000000h
Attribute: RO
Size: 32 bits

This read only register indicates the Wireless USB channel time at the Beacon Period Start Time (BPST) of the next superframe and the current superframe adjustment value.

Table 2-35. WUSBBPST — WUSB Beacon Period Start Time Register Bit Definitions

Bit	Description
31:24	**BPST Adjustment.** This field indicates the current adjustment value of the Beacon Period Start Time in microseconds.
23:0	**BPST Channel Time.** This field indicates the 24-bit Wireless USB channel time at the Beacon Period Start Time of the next superframe.

2.4.20 WUSBDIBUPDATED — WUSB Device Information Buffer Updated Register

Address: WUSBBase + (70h)
Default Value: 00000000h
Attribute: Read/Write (Writes must be DWord Writes)
Size: 128 bits

The WHCD must perform DWord writes to this register. This register is used to signal the WHC that the device information buffer entries have been updated. The device information buffer is present at the main memory address pointed to by the *WUSBDEVICEINFOADDR* register. The WHCD can only set the bits in this register to one. The WHC will reset any bit set in a particular bit field to zero when it updates its internal cached state of the device information buffer for that particular entry.

The WHCD must not update a DIB entry when the corresponding bit for that entry in this register is set. Note that the WHC is only required to implement this register in DWord increments up to the number of devices it supports i.e. only implement (*N_DEVICES-1)*/32 + 1 DWords. The WHCD must not write to DWords that are not implemented by the WHC.

This page intentionally left blank

3. Data Structures

The general architecture of a UWB Multi-Interface Controller (UMC) implementation is given in Figure 1-2. This section presents the data structure definition for the main memory based interface to the UWB Radio Controller (URC) and the Wireless USB Host Controller (WHC) sub-functions in a UMC.

The first half (starting at Section 3.1) describes the data structures used to configure and control the URC. These include the command and event block structures. A URC command block is used to send commands to the URC. The commands and their buffer formats for all the standard commands are described in Section 3.1.3. In addition the event notifications and their buffer formats are described in Section 3.1.3.16.

The second half (starting at Section 3.1.4.12) defines the WHC data structures. These include the interface data structures used to communicate control, status and data between the WHCD and the WHC. The interface consists of a Periodic Schedule, an Asynchronous Schedule, a Device Information Buffer (see Section 3.2.8) and a Device Notification Buffer (see Section 3.2.9).

The data structure definitions in this chapter support a 64-bit memory address space (for interface data structures and data buffers). Note that software must ensure that no interface data structure reachable by the URC or the WHC spans a 4K page boundary except for the Device Information Buffer (see Section 3.2.8). The Device Information Buffer may be up to 8K in length and can span one Page boundary at most. In this specification, a page is equal to 4096 bytes.

The data structures defined in this chapter are (from the URC and the WHC's perspective) a mix of read-only and read/writeable fields. The hardware must preserve the read-only fields on all data structure writes.

3.1 UWB Radio Controller Commands and Events

This section specifies the commands and data structures used to configure and control the URC. The following list summarizes the organization of this section:

- Section 3.1.1 defines the format of the Command Block buffer used to send commands to the URC. It also defines the values for the standard control command codes as well as the result codes that are used to indicate the result of the operation specified by the Command Block back to the URCD.
- Section 3.1.2 defines the format of the Event Block buffer. This structure is used to send asynchronous notifications from the URC to the URCD. In addition results to commands sent to the URC are returned to the URCD using the same format. The standard Event Codes are also defined in this section.
- Section 3.1.3 defines the individual field values (where possible) and parameters of the Command Block buffers for the standard URC commands. It also defines individual field values (where possible) and parameters of the Event Block buffers that are returned as results to these commands.
- Section 3.1.3.16 defines the individual field values (where possible) and parameters of the Event Block buffers for the standard asynchronous notifications that a URC can send to the URCD.

3.1.1 UWB Radio Controller Command Block (RCCB)

This data structure is used to send commands to the URC. This data structure must be physically contiguous and DWord aligned.

The URCD prepares a Command Block buffer and loads the 64-bit physical memory address of this buffer into the URCCMDADDR register (see Section 2.3.4) to send a command to the URC. The Command Block buffer format is diagramed in Figure 3-1 and is described in Table 3-1. All commands for the URC use a Command Block buffer. The size and value of the parameters in a Command Block buffer is determined by the value in the *Command* field.

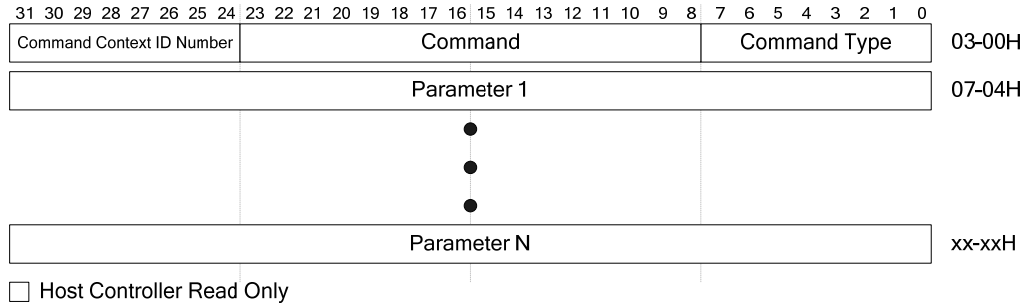


Figure 3-1. UWB Radio Controller Command Block Diagram

Table 3-1. UWB Radio Controller Command Block

Byte Offset	Description
0	**Command Type.** This field specifies the type of command. The different types of commands are defined in the table below. **Value** — **Meaning** 00h — GENERAL 01h-EFh — Reserved for future use. F0h-FFh — Vendor Specific commands
1	**Command.** This field specifies the command code.
3	**Command Context ID Number.** This field specifies the URCD assigned ID for this command. Valid values for commands are 1 through FEH. The value FFH in this field is reserved. A value of 00H in this field is invalid.
4	**Parameter1.** First parameter for this command. The size and value of this parameter is specific to the actual command,
XX	**ParameterN.** Last parameter for this command. The size and value of this parameter is specific to the actual command,

The list of valid values for the Command field in a Command Block when the Command Type field has a value of 00h (**GENERAL**) is given in Table 3-2.

Table 3-2. Standard Command Codes

Command	Value
CHANNEL_CHANGE	16
DEV_ADDR	17
GET_IE	18
RESET	19
SCAN	20
SET_BEACON_FILTER	21
SET_DRP_IE	22
SET_IE	23
SET_NOTIFICATION_FILTER	24
SET_TX_POWER	25
SLEEP	26

Table 3-2. Standard Command Codes (cont.)

Command	Value
START_BEACONING	27
STOP_BEACONING	28
BP_MERGE	29
SEND_COMMAND_FRAME	30
SET_ASIE_NOTIFICATION	31

The result of a command that is sent to the URC is returned in an RCEB (see Section 3.1.2). The list of Result Codes that can be returned in an RCEB is given in Table 3-3.

Table 3-3. Result Codes

Result Codes	Value
SUCCESS	0
FAILURE	1
FAILURE_HARDWARE	2
FAILURE_NO_SLOTS	3
FAILURE_BEACON_TOO_LARGE	4
FAILURE_INVALID_PARAMETER	5
FAILURE_UNSUPPORTED_PWR_LEVEL	6
FAILURE_INVALID_IE_DATA	7
FAILURE_BEACON_SIZE_EXCEEDED	8
FAILURE_CANCELLED	9
FAILURE_INVALID_STATE	10
FAILURE_INVALID_SIZE	11
FAILURE_ACK_NOT_RECEIVED	12
FAILURE_NO_MORE_ASIE_NOTIFICATION	13
FAILURE_TIME_OUT	255

3.1.2 UWB Radio Controller Event Block (RCEB)

This data structure is used to receive notifications from the URC. See Section 4.13 for a complete description of the behavioral model. This data structure must be physically contiguous.

The URC prepares an Event Block buffer and writes the contents of this buffer to the next free location in the Event buffer (see Section 2.3.5). The format of an Event Block buffer is diagramed in Figure 3-2 and is described in Table 3-4. The result of any command sent to the URC is returned in this format as well. The size and value of the parameters in an Event Block buffer is determined by the value in the *Event* field. Note that the URC can write multiple event blocks to the Event buffer before retiring the Event buffer.

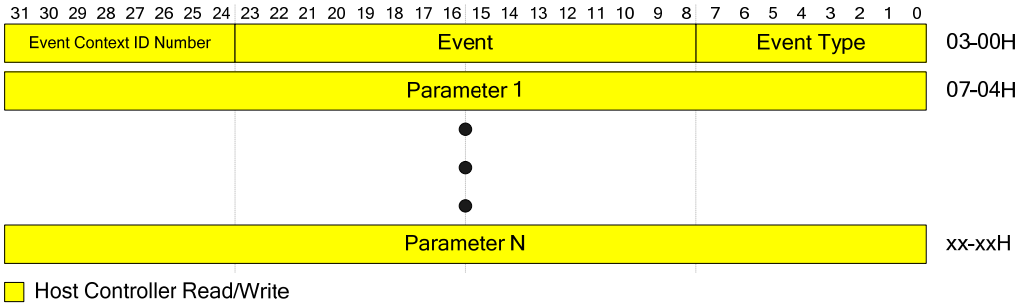


Figure 3-2. UWB Radio Controller Event Block Diagram

Table 3-4. UWB Radio Controller Event Block

Byte Offset	Description
0	**Event Type.** This field specifies the type of event. The different types of events are defined in the table below. **Value** — **Meaning** 00h — GENERAL 01h-EFh — Reserved for future use. F0h-FFh — Vendor Specific events
1	**Event.** This field specifies the event code.
3	**Event Context ID Number.** This field identifies this Event. If this event was a result of a URCD issued command then this must match the *Command Context ID Number* field value in the RCCB. A value of zero indicates an Event that occurred which is not a direct result of a Radio Control Command (Indication). A value of FFH in this field is invalid.
4	**Parameter1.** First parameter for this event. The size and value of this parameter is specific to the actual event.
XX	**ParameterN.** Last parameter for this event. The size and value of this parameter is specific to the actual event.

The list of valid values for the *Event* field in an Event Block when the *Event Type* field has a value of 00h (**GENERAL**) is given in Table 3-5.

Table 3-5. Standard Event Codes

Event	Value
IE_RECEIVED	0
BEACON_RECEIVED	1
BEACON_SIZE_CHANGE	2

Table 3-5. Standard Event Codes (cont.)

Event	Value
BPOIE_CHANGE	3
BP_SLOT_CHANGE	4
BP_SWITCH_IE_RECEIVED	5
DEV_ADDR_CONFLICT	6
DRP_AVAILABILITY_CHANGE	7
DRP	8
BP_SWITCH_STATUS	9
COMMAND_FRAME_RECEIVED	10
CHANNEL_CHANGE_IE_RECEIVED	11
Reserved	12-15
UNKNOWN_COMMAND_RECEIVED	65535

3.1.3 UWB Radio Controller Commands

The following sub-sections define the format of the Command Block buffers for each of the commands that the URCD can send to the URC. For each command, this specification also defines the format of the RCEB that provides the command completion status and any other information that the URC must return to the URCD. The URC operational model for each of the commands is given in Section 4.13.

3.1.3.1 Channel Change Command

This command is used to inform other devices that the URC is going to change the channel

The format of the RCCB and RCEB for this command is given in Table 3-6 and Table 3-7 respectively.

Table 3-6. Channel Change RCCB Format

Byte Offset	Size	Description
0	1	**Command Type.** GENERAL command
1	2	**Command.** CHANNEL_CHANGE
3	1	**Command Context ID Number.** URCD assigned ID for this command.
4	1	**Channel Change Countdown.** This field specifies the number of superframes before the URC changes to the new channel.
5	1	**New Channel Number.** This field specifies the channel number to which the device moves. The encoding of the channel number is specified in Table 4-9.

Table 3-7. Channel Change RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** CHANNEL_CHANGE
3	1	**Event Context ID Number.** This field must match the *Command Context ID Number* field value in the corresponding RCCB.
4	1	**Result Code.** This field contains the result of the CHANNEL_CHANGE command.

3.1.3.2 Device Address Management Command

This command is used to query or set the 16-bit device address or the 48-bit MAC address (EUI-48) that are used by the URC.

The format of the RCCB and RCEB for this command is given in Table 3-8 and Table 3-9 respectively.

Table 3-8. Device Address Management RCCB Format

Byte Offset	Size	Description
0	1	**Command Type.** GENERAL command
1	2	**Command.** DEV_ADDR
3	1	**Command Context ID Number.** URCD assigned ID for this command.
4	1	**Operation Type.** This field specifies the type of the device address management operation. **Bit** **Description** 7:1 **Address type** **Value** **Meaning** 0 16-bit device address 1 48-bit MAC address (EUI-48) 2-127 Reserved 0 **Set**. If this bit is a one (1B), this command causes the URC to **SET** the existing address indicated by *Address Type* with the value of *New Address*. If this bit is zero (0B), this command **GETS** the current *Address Type* address and returns it in a corresponding RCEB.
5	6	**New Address.** This field is ignored (and should be set to zero by the URCD) when the *Set* bit has a value of 0B. When the *Set* field has a value of 1B, this field specifies a replacement address value for the address indicated by the value of the *Address Type* field.

Table 3-9. Device Address Management RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** DEV_ADDR
3	1	**Event Context ID Number.** This field must match the *Command Context ID Number* field value in the corresponding RCCB.
4	6	**Existing Address.** This field contains the requested address value for the URC when this is a Get Device Address operation.
10	1	**Result Code.** This field contains the result of the DEV_ADDR command.

3.1.3.3 Get IE Command

This command is used to get the current IEs that the URC has locally stored.

The format of the RCCB and RCEB for this command is given in Table 3-10 and Table 3-11 respectively.

Table 3-10. Get IE RCCB Format

Byte Offset	Size	Description
0	1	**Command Type.** GENERAL command
1	2	**Command.** GET_IE
3	1	**Command Context ID Number.** URCD assigned ID for this command.

Table 3-11. Get IE RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** GET_IE
3	1	**Event Context ID Number.** This field must match the *Command Context ID Number* field value in the corresponding RCCB.
4	2	**IE Length.** This field contains the length of IE data that is following this field.
6	Var	**IE Data.** This field contains the variable size array containing IE data.

3.1.3.4 Reset Command

This command instructs the URC to reset the URC to the “Not Beaconing” sub-state (see Section 4.13).

The format of the RCCB and RCEB for this command is given in Table 3-12 and Table 3-13 respectively.

Table 3-12. Reset RCCB Format

Byte Offset	Size	Description
0	1	**Command Type.** GENERAL command
1	2	**Command.** RESET
3	1	**Command Context ID Number.** URCD assigned ID for this command.

Table 3-13. Reset RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** RESET
3	1	**Event Context ID Number.** This field must match the *Command Context ID Number* field value in the corresponding RCCB.
4	1	**Result Code.** This field contains the result of the RESET command.

3.1.3.5 Scan Command

This command instructs the URC to start/stop a scan operation.

The format of the RCCB and RCEB for this command is given in Table 3-14 and Table 3-15 respectively.

Table 3-14. Scan RCCB Format

Byte Offset	Size	Description
0	1	**Command Type.** GENERAL command
1	2	**Command.** SCAN
3	1	**Command Context ID Number.** URCD assigned ID for this command.
4	1	**Channel Number.** The physical channel to be scanned. The encoding of the channel number is specified in Table 4-9.
5	1	**Scan State.** This field specifies the type of scan to perform. **Value** — **Description** 0 — SCAN_ONLY Scan only. No other transmit or receive operation is performed until the scan is disabled. 1 — SCAN_OUTSIDE_BP Scan at all times except during the beacon period. 2 — SCAN_WHILE_INACTIVE Scan only when not scheduled to transmit or receive. 3 — SCAN_DISABLED Scanning is disabled. This is the default scanning state on power up or after a RESET command completes successfully. 4 — SCAN_ONLY_STARTTIME Scan only. No other transmit or receive operation is performed until the scan is disabled. 5-255 — Reserved
6	2	**Start Time.** The Superframe Time Counter (STC) value at which to start scanning. This field is only valid if the Scan State is set to SCAN_ONLY_STARTTIME.

Table 3-15. Scan RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** SCAN
3	1	**Event Context ID Number.** This field must match the *Command Context ID Number* field value in the corresponding RCCB.
4	1	**Result Code.** This field contains the result of the SCAN command.

3.1.3.6 Set Beacon Filter Command

This command is used to set the Beacon filter.

The format of the RCCB and RCEB for this command is given in Table 3-16 and Table 3-17 respectively.

Table 3-16. Set Beacon Filter RCCB Format

Byte Offset	Size	Description
0	1	**Command Type.** GENERAL command
1	2	**Command.** SET_BEACON_FILTER
3	1	**Command Context ID Number.** URCD assigned ID for this command.
4	12	**Beacon Slot Filter Mask.** This field is a 96 bit mask (filter) that specifies the beacon slots for which the URCD does not want to receive "Beacon Received" notifications for. A bit value of one (1B) indicates that a notification should not be placed into the event buffer when a beacon is received during the associated beacon slot. A bit value of zero (0B) indicates that a notification should be placed into the event buffer when a beacon is received during the associated beacon slot.
16	1	**Bit** **Description** 7:1 These bits are reserved and must be set to zero 0 **Enable State**. This field specifies whether the filter is enabled or disabled **Value** **Meaning** 0 Filter DISABLED Filter is disabled. This is the default filter state on power up or after a RESET command completes successfully. 1 Filter ENABLED

Table 3-17. Set Beacon Filter RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** SET_BEACON_FILTER
3	1	**Event Context ID Number.** This field must match the *Command Context ID Number* field value in the corresponding RCCB.
4	1	**Result Code.** This field contains the result of the SET_BEACON_FILTER command.

3.1.3.7 Set DRP IE Command

This command is used to set one or more DRP IEs to be transmitted by the URC in its beacon.

The format of the RCCB and RCEB for this command is given in Table 3-18 and Table 3-19 respectively.

Table 3-18. Set DRP IE RCCB Format

Byte Offset	Size	Description
0	1	**Command Type.** GENERAL command
1	2	**Command.** SET_DRP_IE
3	1	**Command Context ID Number.** URCD assigned ID for this command.
4	2	**IE Length.** This field specifies the length of the IE data
6	Var	**IE Data.** This field contains the variable size array containing IE data.

Table 3-19. Set DRP IE RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** SET_DRP_IE
3	1	**Event Context ID Number.** This field must match the *Command Context ID Number* field value in the corresponding RCCB.
4	2	**Remaining Space.** This field indicates the number of bytes that are still available in the beacon slot for this UMC upon completion of this command.
6	1	**Result Code.** This field contains the result of the SET_DRP_IE command.

3.1.3.8 Set IE Command

This command is used to set one or more IEs in the beacon being transmitted by the URC.

The format of the RCCB and RCEB for this command is given in Table 3-20 and Table 3-21 respectively.

Table 3-20. Set IE RCCB Format

Byte Offset	Size	Description
0	1	**Command Type.** GENERAL command
1	2	**Command.** SET_IE
3	1	**Command Context ID Number.** URCD assigned ID for this command.
4	2	**IE Length.** This field specifies the length of the IE data
6	Var	**IE Data.** This field contains the variable size array containing IE data.

Table 3-21. Set IE RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** SET_IE
3	1	**Event Context ID Number.** This field must match the *Command Context ID Number* field value in the corresponding RCCB.

Table 3-21. Set IE RCEB Format (cont.)

Byte Offset	Size	Description
4	2	**Remaining Space.** This field indicates the number of bytes that are still available in the beacon slot for this UMC upon completion of this command.
6	1	**Result Code.** This field contains the result of the SET_IE command.

3.1.3.9 Set Notification Filter Command

This command instructs the URC to filter one or more notifications.

The format of the RCCB and RCEB for this command is given in Table 3-22 and Table 3-23 respectively.

Table 3-22. Set Notification Filter RCCB Format

Byte Offset	Size	Description
0	1	**Command Type.** GENERAL command
1	2	**Command.** SET_NOTIFICATION_FILTER
3	1	**Command Context ID Number.** URCD assigned ID for this command.
4	2	**Notification Mask.** This field specifies the notifications to be filtered. If a bit is set to a one (1B) then that notification must not be reported to the URCD via a new Event Block in the Event Buffer. **Bit** — **Notification** 0 — IE_RECEIVED 1 — BEACON_RECEIVED 2 — BEACON_SIZE_CHANGE 3 — BPOIE_CHANGE 4 — BP_SLOT_CHANGE 5 — BP_SWITCH_IE_RECEIVED 6 — DEV_ADDR_CONFLICT 7 — DRP_AVAILABILITY_CHANGE 8 — DRP 9 — BP_SWITCH_STATUS 10 — COMMAND_FRAME_RECEIVED 11 — CHANNEL_CHANGE_IE_RECEIVED 15:12 — Reserved
6	1	**Bit** — **Description** 7:1 — These bits are reserved and must be set to zero 0 — **Enable State**. This field specifies whether the filter is enabled or disabled **Value** — **Meaning** 0 — Filter DISABLED Filter is disabled. This is the default filter state on power up or after a RESET command completes successfully. 1 — Filter ENABLED

Table 3-23. Set Notification Filter RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** SET_NOTIFICATION_FILTER
3	1	**Event Context ID Number.** This field must match the *Command Context ID Number* field value in the corresponding RCCB.
4	1	**Result Code.** This field contains the result of the SET_NOTIFICATION_FILTER command.

3.1.3.10 Set TX Power Command

This command is used to set the default transmit power for transmissions by this URC.

The format of the RCCB and RCEB for this command is given in Table 3-24 and Table 3-25 respectively.

Table 3-24. Set TX Power RCCB Format

Byte Offset	Size	Description
0	1	**Command Type.** GENERAL command
1	2	**Command.** SET_TX_POWER
3	1	**Command Context ID Number.** URCD assigned ID for this command.
4	1	**Power Level.** This field indicates the number of steps below the highest power level that must be used.

Table 3-25. Set TX Power RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** SET_TX_POWER
3	1	**Event Context ID Number.** This field must match the *Command Context ID Number* field value in the corresponding RCCB.
4	1	**Result Code.** This field contains the result of the SET_TX_POWER command.

3.1.3.11 Sleep Command

On reception of this command the URC will initiate the hibernation sequence.

The format of the RCCB and RCEB for this command is given in Table 3-26 and Table 3-27 respectively.

Table 3-26. Sleep RCCB Format

Byte Offset	Size	Description
0	1	**Command Type.** GENERAL command
1	2	**Command.** SLEEP
3	1	**Command Context ID Number.** URCD assigned ID for this command.
4	1	**Hibernation Count.** This field specifies the number of superframes until the URC begins hibernation. This field is only valid when the *Enable/Disable* field is set to 1.
5	1	**Hibernation Duration.** This field specifies the number of superframes for which the URC intends to hibernate. This field is only valid when the *Enable/Disable* field is set to 1.
6	1	**Enable/Disable.** This is a control field for turning on or off the Sleep command feature in a URC. A value 0 will disable (turn off) Sleep mode operation. A value of 1 will enable (turn on) Sleep mode operation.

Table 3-27. Sleep RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** SLEEP
3	1	**Event Context ID Number.** This field must match the *Command Context ID Number* field value in the corresponding RCCB.
4	1	**Result Code.** This field contains the result of the SLEEP command.

3.1.3.12 Start Beaconing Command

This command instructs the URC to begin beaconing on the specified channel.

The format of the RCCB and RCEB for this command is given in Table 3-28 and Table 3-29 respectively.

Table 3-28. Start Beaconing RCCB Format

Byte Offset	Size	Description
0	1	**Command Type.** GENERAL command
1	2	**Command.** START_BEACONING
3	1	**Command Context ID Number.** URCD assigned ID for this command.
4	2	**BPST.** The STC value at which the URC must start beaconing.
6	1	**Channel Number.** The physical channel in which to beacon. The encoding of the channel number is specified in Table 4-9.

Table 3-29. Start Beaconing RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** START_BEACONING
3	1	**Event Context ID Number.** This field must match the *Command Context ID Number* field value in the corresponding RCCB.
4	1	**Result Code.** This field contains the result of the START_BEACONING command.

3.1.3.13 Stop Beaconing Command

This command instructs the URC to stop beaconing.

The format of the RCCB and RCEB for this command is given in Table 3-30 and Table 3-31 respectively.

Table 3-30. Stop Beaconing RCCB Format

Byte Offset	Size	Description
0	1	**Command Type.** GENERAL command
1	2	**Command.** STOP_BEACONING
3	1	**Command Context ID Number.** URCD assigned ID for this command.

Table 3-31. Stop Beaconing RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** STOP_BEACONING
3	1	**Event Context ID Number.** This field must match the *Command Context ID Number* field value in the corresponding RCCB.
4	1	**Result Code.** This field contains the result of the STOP_BEACONING command.

3.1.3.14 BP Merge Command

This command is used to instruct the URC to start or abort the beacon period merge operation.

The format of the RCCB and RCEB for this command is given in Table 3-32 and Table 3-33 respectively.

Table 3-32. BP Merge RCCB Format

Byte Offset	Size	Description
0	1	**Command Type.** GENERAL command
1	2	**Command.** BP_MERGE
3	1	**Command Context ID Number.** URCD assigned ID for this command.
4	2	**BPST Offset.** This field specifies the offset to the new beacon period start time in microseconds. If this field is set to 65535 (FFFFh), the URC will abort the currently processing BP merge operation.
6	1	**Beacon Slot Offset.** This field specifies the offset to the new beacon slot number after switching the beacon period. If this field is set to zero, the URC must select the new beacon slot number by using normal beacon period join rules.
7	1	**BP Move Countdown.** This field specifies the number of superframes before the URC moves its beacon period. If this field is set to zero, the URC must relocate its BPST immediately.

Table 3-33. BP Merge RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** BP_MERGE
3	1	**Event Context ID Number.** This field must match the *Command Context ID Number* field value in the corresponding RCCB.
4	1	**Result Code.** This field contains the result of the BP_MERGE command.

3.1.3.15 Send Command Frame Command

This command is used to instruct the URC to send a command frame.

The format of the RCCB and RCEB for this command is given in Table 3-34 and Table 3-35 respectively.

Table 3-34. Send Command Frame RCCB Format

Byte Offset	Size	Description
0	1	**Command Type.** GENERAL command
1	2	**Command.** SEND_COMMAND_FRAME
3	1	**Command Context ID Number.** URCD assigned ID for this command.
4	2	**Destination Address.** This field identifies the device address of the recipient.
6	2	**Bit** — **Description** 15:12 — **Frame Subtype.** This field gives the type of command to transmit. The mapping can be found in the MAC layer specification. 11 — **Ack Policy**. This bit indicates whether the command frame should be transmitted using Imm-ACK or No-ACK policy. No-ACK must be used for multicast destinations. **Value** — **Meaning** 0 — No-ACK 1 — Imm-ACK 10 — **Access Method.** This bit indicates whether the command frame should be transmitted using PCA or sent during a DRP reservation. **Value** — **Meaning** 0 — PCA 1 — DRP 9 — **RTS/CTS Policy.** If this bit equals 1B, the URC will use an RTS/CTS exchange before transmitting the command frame. This field is only valid when the Access Method is set to PCA. 8:7 — **Access Category.** This field indicates the access category to be used for PCA. This field is only valid when the *Access Method* is set to PCA. **Value** — **Meaning** 0 — AC_BK 1 — AC_BE 2 — AC_VI 3 — AC_VO 6:2 — **PHYRate.** This field gives the PHY Rate at which the command frame payload must be sent. 1 — **Secure Policy.** If this bit equals 1B, the URC will encrypt the command frame before transmission using the key identified by the provided TKID. 0 — This bit is reserved and must be set to zero
8	1	**MAS Number.** If non-zero, this field indicates the MAS during which the URC must send the Command Frame.
9	1	**Retry Count.** This field is the number of times the command frame transmission must be reattempted during the MAS.

Table 3-34. Send Command Frame RCCB Format (cont.)

Byte Offset	Size	Description
10	3	**TKID.** This field indicates the PTKID or GTKID of the key to use when the *Secure Policy* field indicates the command frame must be encrypted. If the Destination Address is a multicast address then this field contains a GTKID. Otherwise, this field contains a PTKID.
13	3	**Reserved.** This field is reserved and must be set to zero.
16	2	**Encryption Offset.** This field is only valid if the *Secure Policy* field indicates that the command frame must be encrypted. It specifies the offset into the Command Payload that must be encrypted.
18	2	**Command Payload Data Length.** This field specifies the length of the data to be sent in the payload portion of the command frame. The maximum size of Command Payload is 4076 bytes because of the size of the Command Buffer.
20	Var	**Command Payload.** This field specifies the raw data to be sent in the payload portion of the command frame.

Table 3-35. Send Command Frame RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** SEND_COMMAND_FRAME
3	1	**Event Context ID Number.** This field must match the *Command Context ID Number* field value in the corresponding RCCB.
4	1	**Result Code.** This field contains the result of the SEND_COMMAND_FRAME command.

3.1.3.16 Set Application Specific IE Notification

This command instructs the URC to notify the URCD when it detects an Application Specific IE in a beacon from a peer device.

The format of the RCCB and RCEB for this command is given in Table 3-36 and Table 3-37 respectively.

Table 3-36. Set Application Specific IE Notification RCCB Format

Byte Offset	Size	Description
0	1	**Command Type.** GENERAL command
1	2	**Command.** SET_ASIE_NOTIFICATION
3	1	**Command Context ID Number.** URCD assigned ID for this command.
4	2	**Device Address.** The device address of the peer device the URC must monitor.
6	2	**ASIE Identifier.** The Application Specific IE that the URC must scan for in peer devices' beacons.
8	1	**Bit** / **Description** 0 — **Enable/Disable**. If this bit is set to zero then the URC must stop scanning for the Application Specific IE (from a specific peer device if the Ignore Device Address bit is not set). 1 — **Ignore Device Address**. If this bit is set to one then the URC must send an IE Received notification if it detects the ASIE specified in the ASIE Identifier field from any peer device. If this bit is set to zero then the URC must only send an IE Received notification if it detects the ASIE in a peer device whose device addressed is equal to the value set in the *Device Address* field. 7:2 — Reserved

Table 3-37. Set Application Specific IE Notification RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** SET_ASIE_NOTIFICATION
3	1	**Event Context ID Number.** This field must match the *Command Context ID Number* field value in the corresponding RCCB.
4	1	**Result Code.** This field contains the result of the SET_ASIE_NOTIFICATION command.

3.1.4 Radio Control Notifications

The following sub-sections define the Event Block buffers returned for each notification defined in this specification. If the URC receives a beacon that would result in multiple notifications (e.g. Beacon Received and IE Received notifications) then the URC must send all the possible notifications for that beacon to the URCD.

Note that the IE Received and BP Switch IE Received notifications are sent to the URCD only when these IEs are detected in neighbor beacons.

The URC operational model for each of the events is given in Section 4.13.

3.1.4.1 IE Received Notification

This notification reports that the URC has received an IE targeted to this URC in a beacon from another device.

The RCEB for this notification is given in Table 3-38.

Table 3-38. IE Received Notification RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** IE_RECEIVED
3	1	**Event Context ID Number.** Zero (**Indication**)
4	2	**Source Address.** This field identifies the device which sent the IE.
6	2	**IE Length.** This field specifies the length of the IE data
8	Var	**IE Data.** This field contains the variable size array containing IE data.

3.1.4.2 Beacon Received Notification

This notification reports that a beacon was received.

The RCEB for this notification is given in Table 3-39.

Table 3-39. Beacon Received Notification RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** BEACON_RECEIVED
3	1	**Event Context ID Number.** Zero (**Indication**)
4	1	**Channel Number.** The physical channel on which the beacon was received. The encoding of the channel number is specified in Table 4-9.
5	1	**Bit** **Description** 7:4 These bits are reserved and must be set to zero 3:0 **Beacon Type**. This field indicates the type of the received beacon. **Value** **Meaning** 0 Scan 1 Neighbor 2 Overlapping Alien 3 Non-Overlapping Alien 4-15 Reserved
6	2	**Receive Time.** This field identifies the value of the STC at the time the beacon was received by the URC.
8	1	**LQI.** This field specifies the Link Quality Indication
9	1	**RSSI.** This field specifies the Receive Signal Strength Indication.
10	2	**Beacon Information length.** This field specifies the number of bytes in the beacon information.
12	Var	**Beacon Information.** Variable size array containing all the information in the received beacon.

3.1.4.3 Beacon Size Change Notification

This notification reports that the size of the beacon has changed due to a modification of one or more of the IEs handled by the URC.

The RCEB for this notification is given in Table 3-40.

Table 3-40. Beacon Size Change Notification RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** BEACON_SIZE_CHANGE
3	1	**Event Context ID Number.** Set to Zero (**Indication**)
4	2	**New Beacon Size.** This field indicates the new size of the beacon including the MAC Header, Beacon Parameters and all the IEs.

3.1.4.4 BPOIE Change Notification

This notification reports that the BPOIE that is being transmitted by the URC has changed.

The RCEB for this notification is given in Table 3-41.

Table 3-41. BPOIE Change Notification RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** BPOIE_ CHANGE
3	1	**Event Context ID Number.** Zero (**Indication**)
4	2	**BPOIE Length.** This field indicates the length of the BPOIE sent by the URC.
6	Var	**BPOIE.** Variable size array containing all the information in the BPOIE sent by the URC.

3.1.4.5 BP Slot Change Notification

This notification reports that the URC has changed the beacon slot number where it is transmitting its beacon.

The RCEB for this notification is given in Table 3-42.

Table 3-42. BP Slot Change Notification RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** BP_SLOT_ CHANGE
3	1	**Event Context ID Number.** Zero (**Indication**)
4	1	**Bit** — **Description** 7 — **No Slot.** This bit is set to one when the URC is unable to choose a new beacon slot. 6:0 — **Slot Number.** This field indicates the new beacon slot where the URC is transmitting its beacon. This field is only valid when the *No Slot* bit is set to zero.

3.1.4.6 BP Switch IE Received Notification

This notification reports that the URC has detected a BP Switch IE in a beacon from a neighbor device.

The RCEB for this notification is given in Table 3-43.

Table 3-43. BP Switch IE Received Notification RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** BP_SWITCH_IE_RECEIVED
3	1	**Event Context ID Number.** Zero (**Indication**)
4	2	**Source Address.** This field identifies the device which sent the BP Switch IE.
6	2	**IE Length.** This field indicates the length of the BP Switch IE received by the URC.
8	Var	**IE Data.** Variable size array containing the BP Switch IE received by the URC.

3.1.4.7 Device Address Conflict Notification

This notification reports that the URC has detected a device address conflict.

The RCEB for this notification is given in Table 3-44.

Table 3-44. Device Address Conflict Notification RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** DEV_ADDR_CONFLICT
3	1	**Event Context ID Number.** Set to Zero (**Indication**)

3.1.4.8 DRP Availability Changed Notification

This notification reports that the URC's DRP availability has changed.

The RCEB for this notification is given in Table 3-45.

Table 3-45. DRP Availability Changed Notification RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** DRP_AVAILABILITY_CHANGE
3	1	**Event Context ID Number.** Zero (**Indication**)
4	32	**DRP Availability Bitmap.** This field is a 256 bit bitmap that specifies the MASs that are available for a DRP reservation. The least significant bit corresponds to the first MAS in the superframe. If the bit is set to one, then the corresponding MAS is available for a DRP reservation, otherwise it is not available.

3.1.4.9 DRP Notification

This notification reports information related to DRP reservations for which the URCD is a participant.

The RCEB for this notification is given in Table 3-46.

Table 3-46. DRP Notification RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** DRP
3	1	**Event Context ID Number.** Zero (**Indication**)
4	2	**Source Address.** This field identifies the device which sent the DRP IE.
6	1	**Bit** **Description** 7:4 These bits are reserved and must be set to zero 3:0 **Reason Code**. This field determines the reason that this notification was generated. **Value** **Meaning** 0 DRP IE Received[1] 1 Conflict 2 Termination 3-15 Reserved
7	1	**Beacon Slot Number.** This field indicates the beacon slot number of the received beacon which contains the DRP IEs in the *IE Data* field.
8	2	**IE Length.** This field specifies the length of the IE
10	Var	**IE Data.** This field contains the variable size array containing the IE data.

[1]The URC received a beacon from a peer device with DRP IE(s) that had the URC's DevAddr as the target or owner.

3.1.4.10 BP Switch Status Notification

This notification reports that the URC has sent a BP Switch IE, relocated its beacon to an alien BP or halted the BP merge operation.

Table 3-47. BP Switch Status Notification RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** BP_SWITCH_STATUS
3	1	**Event Context ID Number.** Zero (**Indication**)
4	1	**Bit** **Description** 7:4 These bits are reserved and must be set to zero 3:0 **Status.** This field indicates the status of the BP merge operation. **Value** **Meaning** 0 Done. 1 Continue. 2 Halted (Neighbor halts the relocation process) 3 Halted (Alien relocates its beacon earlier) 4 Aborted 5-15 Reserved
5	1	**Beacon Slot Offset.** This field indicates the offset to the new beacon slot number after switching the beacon period or zero if the new beacon slot number is selected by using normal beacon period join rules.
6	2	**BPST Offset.** This field indicates the offset to the new beacon period start time in microseconds.
8	1	**BP Move Countdown.** This field indicates the remaining number of superframes before the URC moves its beacon period.

3.1.4.11 Command Frame Received Notification

This notification reports that the URC has received a command frame from a peer device.

Table 3-48. Command Frame Received Notification RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** COMMAND_FRAME_RECEIVED
3	1	**Event Context ID Number.** Zero (**Indication**)
4	2	**Receive Time.** This field reports the STC value when this Command Frame was received.
6	2	**Source Address.** This field identifies the device which sent the data.
8	2	**Destination Address.** This field identifies the destination address of the command frame
10	2	**Bit** — **Description** 15:12 — **Frame Subtype.** This field gives the type of command that was received. The mapping can be found in the MAC layer specification 11 — **Ack Policy**. This bit indicates whether the command frame received used the Imm-ACK or No-ACK policy. **Value** — **Meaning** 0 — No-ACK 1 — Imm-ACK 10:7 — These bits are reserved and must be set to zero 6:2 — **PHYRate.** This field gives the PHY Rate at which the command frame payload was sent. 1 — **Secure Frame Bit.** This bit is set if this command frame was received as a secure frame. 0 — This bit is reserved and must be set to zero
12	2	**Reserved.** This field is reserved and must be set to zero.
14	2	**Command Payload Data Length.** This field reports the length of the data received in the payload portion of a Command Frame. The maximum size of Command Payload is 4080 bytes because of the size of the Event Buffer. The URC must set the actual length of the command frame payload received in this field.
16	Var	**Command Payload.** This field is the raw data that was received in the payload portion of a Command Frame. If the command payload data length is greater than 4080 bytes, then this field will contain the first 4080 bytes only.

3.1.4.12 Channel Change IE Received Notification

This notification reports that the URC has detected a Channel Change IE in a beacon from a neighbor device.

The RCEB for this notification is given in Table 3-43.

Table 3-49. Channel Change IE Received Notification RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** CHANNEL_CHANGE_IE_RECEIVED
3	1	**Event Context ID Number.** Zero (**Indication**)
4	2	**Source Address.** This field identifies the device which sent the Channel Change IE.
6	2	**IE Length.** This field indicates the length of the Channel Change IE received by the URC.
8	Var	**IE Data.** Variable size array containing the Channel Change IE received by the URC.

3.1.4.13 Unknown Command or Command Type Received Notification

This notification reports that the URC received an RCCB with a Command and/or Command Type that it cannot decode.

Table 3-50. Unknown Command or Command Type Received Notification RCEB Format

Byte Offset	Size	Description
0	1	**Event Type.** GENERAL Event
1	2	**Event.** UNKNOWN_COMMAND_RECEIVED
3	1	**Event Context ID Number.** This field must match the *Command Context ID Number* field value in the RCCB that contained the unknown command or command type.

3.2 WHC Data Structures

This section defines the data structures used by the WHCD to interface with the WHC. It describes the two types of schedules (a Periodic Schedule and an Asynchronous Schedule), the Device Information Buffer format (see Section 3.2.8) and the structure of the Device Notification Buffer (see Section 3.2.9).

The periodic schedule is based on a Periodic Zone List (see Section 3.2.1). This (Periodic Zone List) is the root of all periodic (isochronous and interrupt transfer type) support for the WHC interface. Isochronous data streams are managed using Queue Heads (QHeads) with Isochronous Queue Element Transfer Descriptors. (iTDs described in Section 3.2.5).

The asynchronous schedule is based on an Asynchronous Transfer List (see Section 3.2.2). This (Asynchronous Transfer List) is the root for all the bulk and control transfer type support. Interrupt, Control and Bulk data streams are managed via QHeads (Section 3.2.6) and Queue Element Transfer Descriptors (qTDs described in Section 3.2.4). These data structures are optimized to reduce the total memory footprint of the schedule and to reduce (on average) the number of memory accesses needed to execute Wireless USB transactions.

3.2.1 Periodic Zone List

This schedule is for all periodic transfers (isochronous and interrupt). The periodic schedule is in main memory and referenced from the WUSBPERIODICLISTBASE address register and the WUSBMASINDEX register. The periodic schedule is an array of memory address pointers called the Periodic Zone List. The value in the WUSBPERIODICLISTBASE address register is combined with the value in WUSBMASINDEX register to produce a memory pointer into the Periodic Zone List. The Periodic Zone List implements a *sliding window* of work over time.

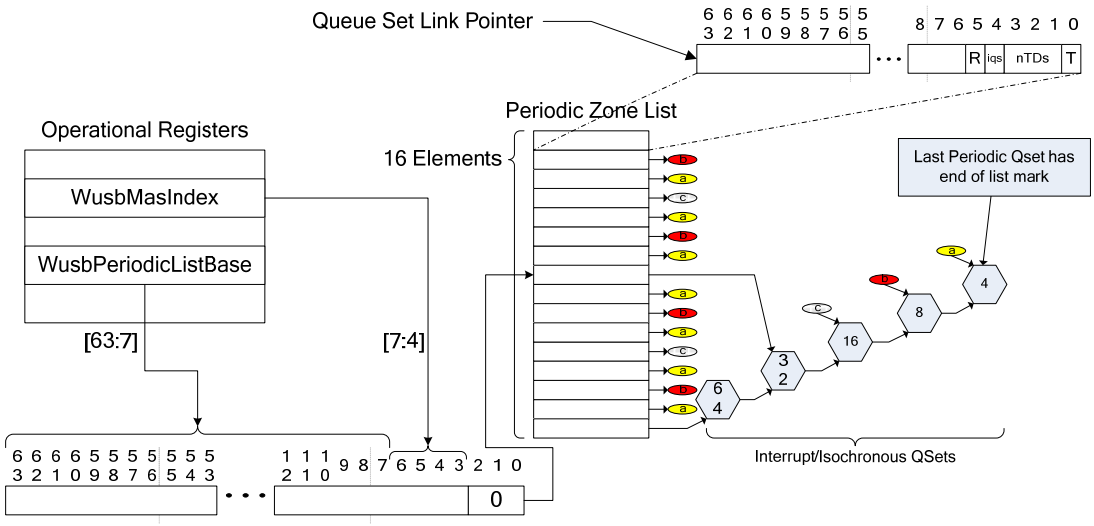


Figure 3-3. Example Periodic Schedule Organization

The Periodic Zone List is a 128 byte aligned array of Queue Set Link Pointers. The length of the zone list is 16 elements. Queue Set Link Pointers direct the WHC to the first Queue Set in the zone's periodic schedule for the current MAS. The link pointers are aligned on QWord boundaries within the Periodic Zone List. The format of the link pointers is the same as the Queue Set Link Pointers described in Section 3.2.6.1.

The Link pointers always reference memory objects that are 64-byte aligned. The referenced object may be an isochronous QHead or an interrupt QHead.

When the *T-Bit* (bit 0) in the Queue Set Link Pointer (see Section 3.2.6.1) is set to a one, the WHC must not use the value of the list pointer as a physical memory pointer.

3.2.2 Asynchronous List QHead Pointer

The Asynchronous Transfer List (based at the WUSBASYNCLISTADDR register), is where all the control and bulk transfers are managed.

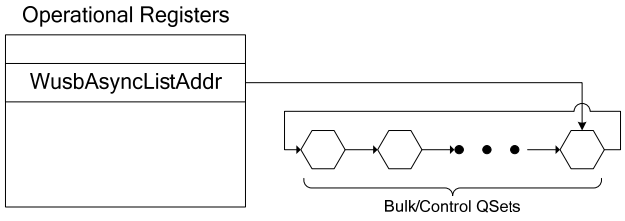


Figure 3-4. Asynchronous Schedule Organization

The Asynchronous list is a simple circular list of QHeads. The WUSBASYNCLISTADDR register is a pointer to the *next* QHead. This implements a pure round-robin service for all QHeads linked into the asynchronous list.

3.2.3 Interface Data Structure Model Overview

The organizational model of the interface data structures is illustrated in Figure 3-5. Individual transfers to function endpoints are supported using a group of data structures that are collectively referred to as a Queue Set. A Queue Set includes one QHead and an array of queue elements. The maximum number of queue elements in a Queue Set is eight.

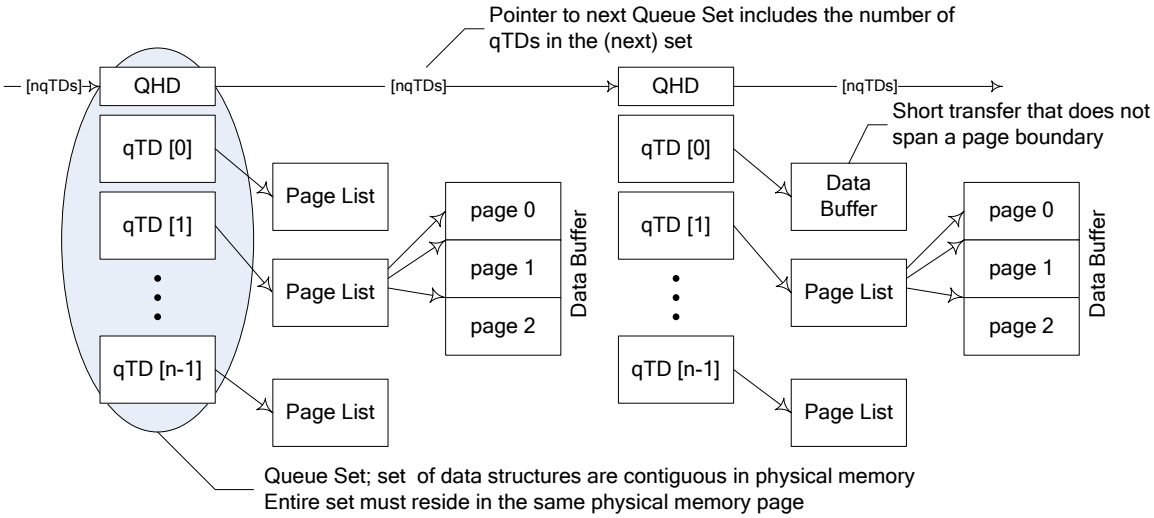


Figure 3-5. Interface Data Structure Organization

The qTDs in a Queue Set are intended to be used as a circular queue resource. The consumption of the queue elements is in strict ascending order. When the WHC completes a transfer associated with a qTD element, it will automatically attempt to advance to the next qTD in the array. For example, when it completes a transfer on qTD[1], it will (usually) automatically attempt to advance to qTD[2] (assuming the number of qTDs in this example is greater than 3). The exception to this rule is for handling a last packet flag for an IN endpoint. In the event that the WHCD needs more than one qTD to describe a transfer, encountering a last packet flag requires the host skip the rest of the qTDs associated with the transfer. Details are provided in

Section 4.7.1.2. When a WHC completes the last qTD element in the array, it automatically attempts to advance to qTD[0].

Each qTD contains a reference to a data buffer or a list of page pointers. If the buffer associated with the transfer does not span a page boundary, then the pointer is either the physical buffer address or a pointer to the page list. If the buffer does span a page boundary, then the pointer is to a page list. A page list is an array of physical page pointers.

3.2.4 Queue Element Transfer Descriptor (qTD)

This data structure is always associated with a single QHead (see Section 3.2.6) in the context of a Queue Set. This data structure is used to transfer data in one or more Wireless USB transactions. See Section 4.7 for a complete description of the behavioral model. This data structure is used to transfer up to 1048575 bytes. The structure contains a DWord of transfer state and a 64 bit pointer to the page pointer array (or data buffer). The last two DWords of the data structure are valid only if the qTD describes a control transfer.

The buffer associated with this transfer must be virtually contiguous. The buffer may start on any byte boundary. A separate buffer pointer list element must be used for each physical page in the buffer, regardless of whether the buffer is physically contiguous.

WHC updates (WHC writes) to individual qTDs only occur during transfer retirement (see Section 4.7.1.3). References in the following bit field definitions of updates to the 'qTD' are to the qTD portion of a Queue Set (see Figure 3-10).

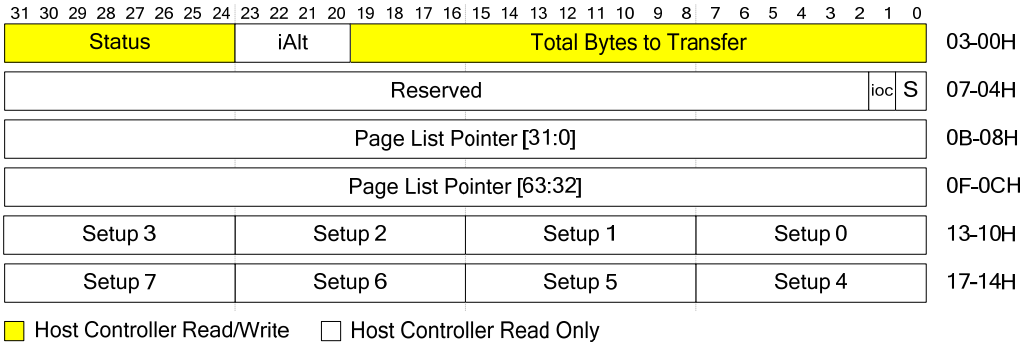


Figure 3-6. Queue Element Transfer Descriptor Block Diagram

3.2.4.1 qTD Transfer Length and Status

The first DWord of a qTD contains the number of bytes to transfer. The WHC will update this field along with the Transfer Status field when this transfer descriptor is retired. It also contains the next qTD (iAlt) to be processed in case a packet with the *Last Packet* flag set is received from the device.

Note: the field descriptions forward reference fields defined in the QHead (Section 3.2.6). Where necessary, these forward references are preceded with a *QH.* notation.

Table 3-51. qTD Transfer Length and Status (DWord 0)

Bit	Description
31:24	**Status.** The WHCD must set the *Active* bit in this field before the WHC starts execution of this qTD. This field is used by the WHC to communicate individual command execution states back to the WHCD. This field contains the status of the last transaction performed on this qTD. The bit encodings are: **Bit** — **Status Field Description** 7 — **Active.** Set to 1 by the WHCD to enable the execution of transactions by the WHC. Refer to Section 4.7.1.2 for operational details about when the WHC transitions this bit from a one to a zero. 6 — **Halted.** Set to a 1 by the WHC during status updates to indicate that a serious error has occurred at the device/endpoint addressed by this qTD. This can be caused by babble or reception of the STALL handshake from the device during a transaction. Any time that a transaction results in the *Halted* bit being set to a one, the *Active* bit is also set to 0. 5 — **Data Buffer Error.** Set to a 1 by the WHC during status update to indicate that the WHC is unable to keep up with the reception of incoming data (overrun) or is unable to supply data fast enough during transmission (underrun). Section 4.12.1.1.2 defines the requirements of the WHC when an underrun error occurs. If an overrun condition occurs, the WHC will force a timeout condition on the Wireless USB, invalidating the transaction at the source. If the WHC sets this bit to a one, then it remains a one for the duration of the transfer. 4 — **Babble Detected.** Set to a 1 by the WHC during status update when a "babble" (device transmits more than its maximum packet size) is detected during the transaction. In addition to setting this bit, the WHC also sets the *Halted* bit to a 1. Since "babble" is considered a fatal error for the transfer, setting the *Halted* bit to a 1 insures that no more transactions occur as a result of this descriptor. 3 — **Retry Count Exceeded.** Set to a 1 by the WHC during status update when the number of retries for a particular transaction exceeds the value in the *Maximum Retry* field of the QHead. In addition to setting this bit, the WHC also sets the *Halted* bit to a 1 to insure that no more transactions occur as a result of this descriptor. 2 — **Last Packet Flag.** The WHCD will set this bit to instruct the WHC to set the *Last Packet Flag* in the Wireless USB header when it transmits the last packet for an OUT transfer. This bit must be initialized to zero for IN transfers. The WHC will set this bit if it receives a packet with the *Last Packet Flag* set in the Wireless USB Header while performing this transfer. 1 — **Queue Set Inactivated**. Set to a 1 by the WHC during status updates to indicate that the Queue Set has been inactivated as per the *Inactivate on Next Transaction* bit (see Section 3.2.6.2). Please refer to Section 4.7.6 for a complete description of the WHC requirements how to inactivate the Queue Set. 1:0 — **Reserved.** This bit is reserved and must be set to zero.

Table 3-51. qTD Transfer Length and Status (DWord 0) (cont.)

Bit	Description
23:20	**Alternate qTD index (iAlt)**. This field is used by the WHC to advance the transfer to the next appropriate qTD when the WHC detects a last packet flag in a data packet and is ready to advance to the next qTD. This field is organized into two sub-fields as illustrated. **Bit** **Description** 3 **Valid.** When this bit is a 1B, then the value in the Alt qTD Number field is valid. 2:0 **Alt qTD Number.** This field is the index of the qTD the WHC must advance to if the transfer completes by encountering a last packet flag. Note: The WHCD must ensure the target qTD is properly initialized prior to setting the *Active* bit in the current qTD.
19:0	**Total Bytes to Transfer.** This field specifies the total number of bytes to be moved with this transfer descriptor. This field is decremented by the number of bytes actually moved during the transaction, only on the successful completion of the transaction. If the *Small Transfer* bit (see Section 3.2.4.2) is set then the maximum value that the WHC may store in this field is restricted to 4K. Please refer to Section 4.7.1.2 for a complete description of the WHC requirements when this field decrements to zero. If the value of this field is zero when the WHC fetches this transfer descriptor (and the *Active* bit is set), the WHC executes a zero-length transaction and retires the transfer descriptor. If the *QH.TRType* is set to Control then it executes a zero length control transfer. It is not a requirement for OUT transfers that *Total Bytes To Transfer* be an even multiple of *QH.Maximum Packet Length*. If the WHCD builds such a transfer descriptor for an OUT transfer, the last transaction will always be less than *QH.Maximum Packet Length*. The WHCD must also set the *Last Packet* bit in this scenario in order to properly mark the short packet in the data stream. However, if an IN transfer is requested by using multiple qTDs, *Total Bytes to Transfer* must be a multiple of *QH.Maximum Packet Length* except for the last qTD of the transfer.

3.2.4.2 qTD Token

The second DWord of a qTD determines whether the WHC must interrupt the WHCD when it retires this qTD and whether this qTD describes a small transfer.

Table 3-52. qTD Token (DWord 1)

Bit	Description
31:2	**Reserved.** These bits are reserved and must be set to zero.
1	**Interrupt On Complete (IOC).** If this bit is set to a one, it specifies that when this qTD is completed, the WHC must issue an interrupt.
0	**Small Transfer (S).** If this bit is set to a one then the *Page List Pointer* actually points to the buffer to be used for this qTD. The maximum number of bytes that can be transferred when this bit is set is less than or equal to 4K and cannot span a page boundary.

3.2.4.3 qTD Page List Pointer

The third and fourth DWords of a qTD comprise a 64-bit memory address of a data buffer or a Page List. A Page List is an array that contains the 64-bit physical memory address pointers to the individual pages of a data buffer. Refer to Section 4.7.2 for operational requirements. Each memory page referenced from the Page List array must be 4K bytes in length, except for the first and last, which may have length less that 4K.

Table 3-53. qTD Page List Pointer (DWord 4 and 5)

Bit	Description
63:0	**Page List Pointer.** This gives the physical address of the Page List. Each entry in the list is a 4K page aligned physical memory address except for the first one. This buffer must be physically contiguous and DWord aligned. The WHC is not required to update this field when it writes back the qTD after completing the transfer described by the qTD. If the *Small Transfer* bit is set to a one, then the *Page List Pointer* is actually the physical memory address pointer to the data buffer to be used with the qTD. In this case the pointer does not need to be DWord aligned.

3.2.4.4 qTD Setup Bytes

The fifth and sixth DWords of a qTD contain the eight bytes of Setup data if the *QH.TR Type* field is set to **Control** in the QHead. Otherwise, these DWords should be set to zeros by the WHCD.

3.2.4.5 Page List

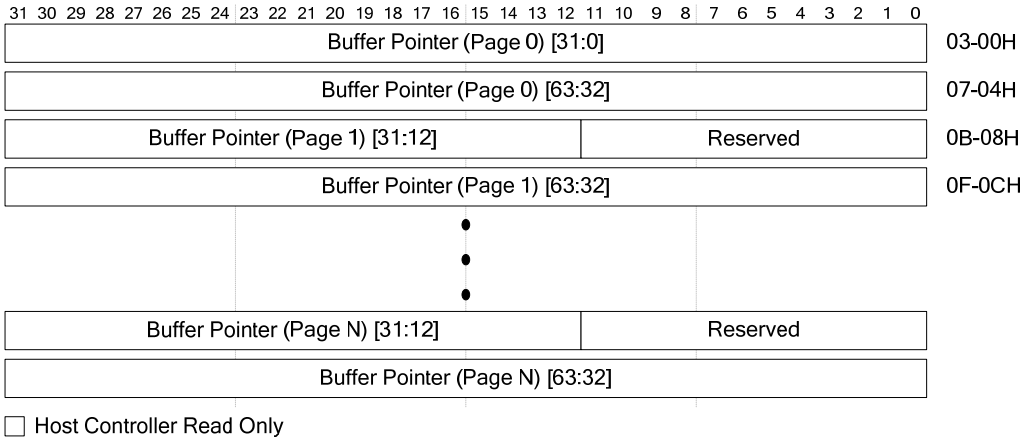


Figure 3-7. Page List Layout

As mentioned in Section 3.2.4.3, the Page List is an array of 64-bit physical memory address pointers which reference individual pages of a data buffer. The lower 12 bits in each pointer are reserved (except for the first one) as each memory pointer must reference the start of a 4K page and must be 4K in length. Only the first and last entries can be less than 4K in length. The WHCD must ensure that sufficient data pointers are present to satisfy the entire value of the *Total Bytes to Transfer* field in the qTD Transfer Length and Status DWord.

3.2.5 Isochronous Queue Element Transfer Descriptor (iTD)

This data structure is only used with a QHead with *TR Type* set to Isochronous (see Section 3.2.7). This data structure is used for one or more Wireless USB Isochronous transactions. See Section 4.7.11 for a complete description of the behavioral model. This data structure is used to transfer up to 1048576 (256*4096) bytes. The structure contains a DWord of transfer state, a 64 bit pointer to the page pointer array (or data buffer) and a 64 bit pointer to the Segment List arrays. This data structure must start on a 64-byte boundary and be physically contiguous.

The buffer associated with this transfer must be virtually contiguous. The buffer may start on any byte boundary. A separate buffer pointer list element must be used for each physical page in the buffer, regardless of whether the buffer is physically contiguous.

WHC updates (WHC writes) to individual iTDs only occur during transfer retirement (see Section 4.7.1.3). References in the following bit field definitions of updates to the 'iTD' are to the iTD portion of a QHead (see Figure 3-11).

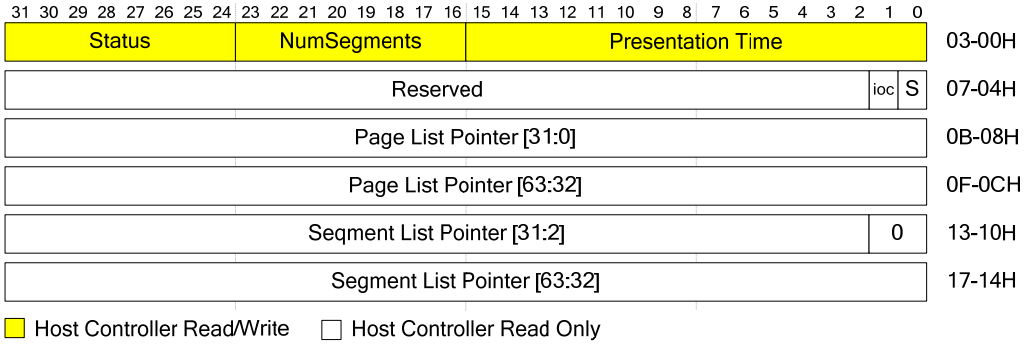


Figure 3-8. Isochronous Queue Element Transfer Descriptor Block Diagram

3.2.5.1 iTD Transfer Status

The first DWord of an iTD contains the Transfer Status field, the Presentation Time and the number of segments in the transfer described by the iTD. The WHC will update this field when this iTD is retired.

Table 3-54. iTD Status (DWord 0)

Bit	Description
31:24	**Status.** This field is set by the WHCD to indicate that an active data segment is present in the array pointed to by the *Segment List* pointer. The WHC will reset the *Active* bit to zero when all active segments have been transmitted or discarded. The bit encodings are: **Bit** — **Status Field Description** 7 — **Active.** Set to 1 by the WHCD to enable the execution of transactions by the WHC. Refer to Section 4.7.11 for operational details about when the WHC transitions this bit from a one to a zero. 6 — **Reserved.** This bit is reserved and must be set to zero. 5 — **Data Buffer Error.** Set to a 1 by the WHC during status update to indicate that the WHC is unable to keep up with the reception of incoming data (overrun) or is unable to supply data fast enough during transmission (underrun). Section 4.12.1.1.2 defines the requirements of the WHC when an underrun error occurs. If an overrun condition occurs, the WHC will force a timeout condition on the Wireless USB, invalidating the transaction at the source. If the WHC sets this bit to a one, then it remains a one for the duration of the transfer. 4 — **Babble Detected.** Set to a 1 by the WHC during status update when a "babble" (device transmits more than its maximum packet size) is detected during the transaction. The WHC must reset the *Active* bit and retire the iTD. 3:2 — **Reserved.** These bits are reserved and must be set to zero. 1 — **Queue Set Inactivated.** Set to a 1 by the WHC during status updates to indicate that the Queue Set has been inactivated as per the *Inactivate on Next Transaction* bit (see Section 3.2.6.2). Please refer to Section 4.7.6 for a complete description of the WHC requirements how to inactivate the Queue Set. 0 — **Reserved.** This bit is reserved and must be set to zero.
23:16	**Number of Segments (NumSegments).** This field indicates the number of data segments in the segment list pointed to by the *Segment List Pointer* field.
15:0	**Presentation Time.** For OUT transfers, this field indicates the presentation time associated with the first data segment in the segment list. Note that the WHC must ignore the upper most bit (Bit 16) of the Wireless USB Channel time 1/8th millisecond value to determining when to discard packets.

3.2.5.2 iTD Token

The second DWord of an iTD contains whether this iTD describes a small transfer and interrupt information.

Table 3-55. iTD Token (DWord 1)

Bit	Description
31: 2	**Reserved.** These bits are reserved and must be set to zero.
1	**Interrupt On Complete (IOC).** If this bit is set to a one, it specifies that when this iTD is completed, the WHC should issue an interrupt.
0	**Small Transfer (S).** If this bit is set to a one then the *Page List Pointer* actually points to the buffer to be used for this iTD. The maximum number of bytes that can be transferred when this bit is set is less than or equal to 4K and cannot span a page boundary.

3.2.5.3 iTD Page List Pointer

The third and fourth DWords of an iTD comprise a 64-bit physical memory address of a data buffer or a Page List. A Page List is an array that contains the 64-bit physical memory address pointers to the individual pages of a data buffer. Refer to Section 4.7.2 for operational requirements.

Table 3-56. iTD Page List Pointer (DWord 4 and 5)

Bit	Description
63:0	**Page List Pointer.** This gives the physical address of the Page List. Each entry in the list is a 4K page aligned physical memory address except for the first one. This buffer must be physically contiguous and DWord aligned. The WHC is not required to update this field when it writes back the iTD after completing the transfer described by the iTD. If the *Small Transfer* bit is set then the *Page List Pointer* is actually a pointer to the physical page of the buffer to be used with the iTD. In this case the pointer does not need to be DWord aligned.

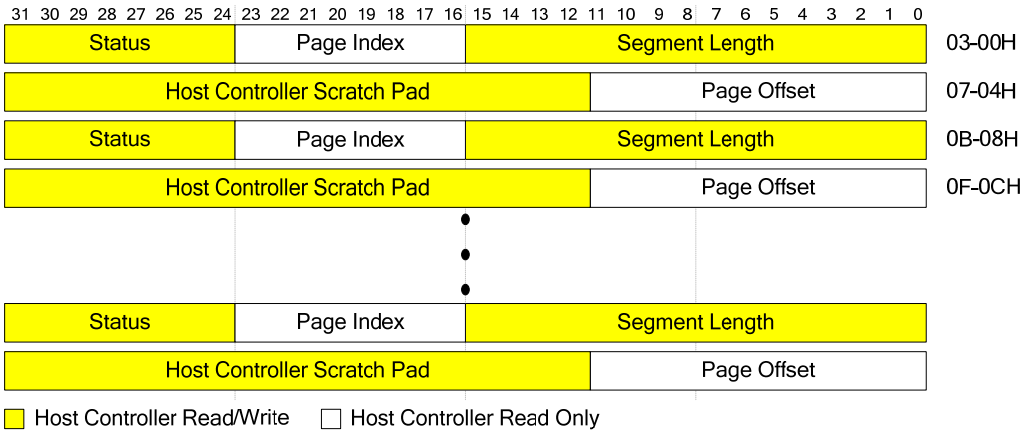
3.2.5.4 iTD Segment List Pointer

The fifth and sixth DWords of an iTD comprise the physical memory address of a Segment List. The Segment List is an array that contains the information on data segments to be transferred to this Isochronous endpoint. Each segment information element is 8 bytes in length and the format is defined in Section 3.2.5.5. Refer to Section 4.7.11 for operational requirements.

Table 3-57. Segment List Pointer (DWord 6 and 7)

Bit	Description
63:2	**Segment List Pointer.** This gives the physical address of the Isochronous Segment List buffer defined in Section 3.2.5.5.
1:0	**Reserved.** These bits are reserved and must be set to zero.

3.2.5.5 Segment List



3-9. Segment List Layout

Each entry in the Segment List consists of two DWords. The fields within each entry are described in Table 3-58.

Table 3-58. Segment List Entry

Bit	Description
63:44	**Host Controller Scratch Pad.** The WHCD will initialize these bits to zero and must not modify its contents until the transfer is completed. The WHC is free to use these bits to store intermediate transfer state.
43:32	**Page Offset.** This field is the offset within the page in the Page List referred to by *Page Index*.
31:24	**Status.** This field is used by the WHC to communicate individual Segment command execution states back to the WHCD. The bit encodings are: **Bit** — **Status Field Description** 7 — **Active.** Set to 1 by the WHCD to enable the execution of transactions by the WHC. Refer to Section 4.7.11 for operational details about when the WHC transitions this bit from a one to a zero. 6:5 — **Reserved.** These bits are reserved and must be set to zero. 4 — **Babble Detected.** Set to a 1 by the WHC during status update when a “babble” (device transmits more than the expected segment length specified by the WHCD) is detected during the transaction. The *Active* bit must be also set to a zero when this bit is set. 3:0 — **Reserved.** These bits are reserved and must be set to zero.
23:16	**Page Index.** This field is used in conjunction with the *Page Offset* field to determine the physical memory address of the data buffer for this segment.
15:0	**Segment Length.** This is the length of the Segment to be transferred. For OUT transactions, the WHC need not update this field. For IN transactions, the WHC must update this field with the actual number of bytes received.

3.2.6 Queue Head (QHead) with qTD overlay

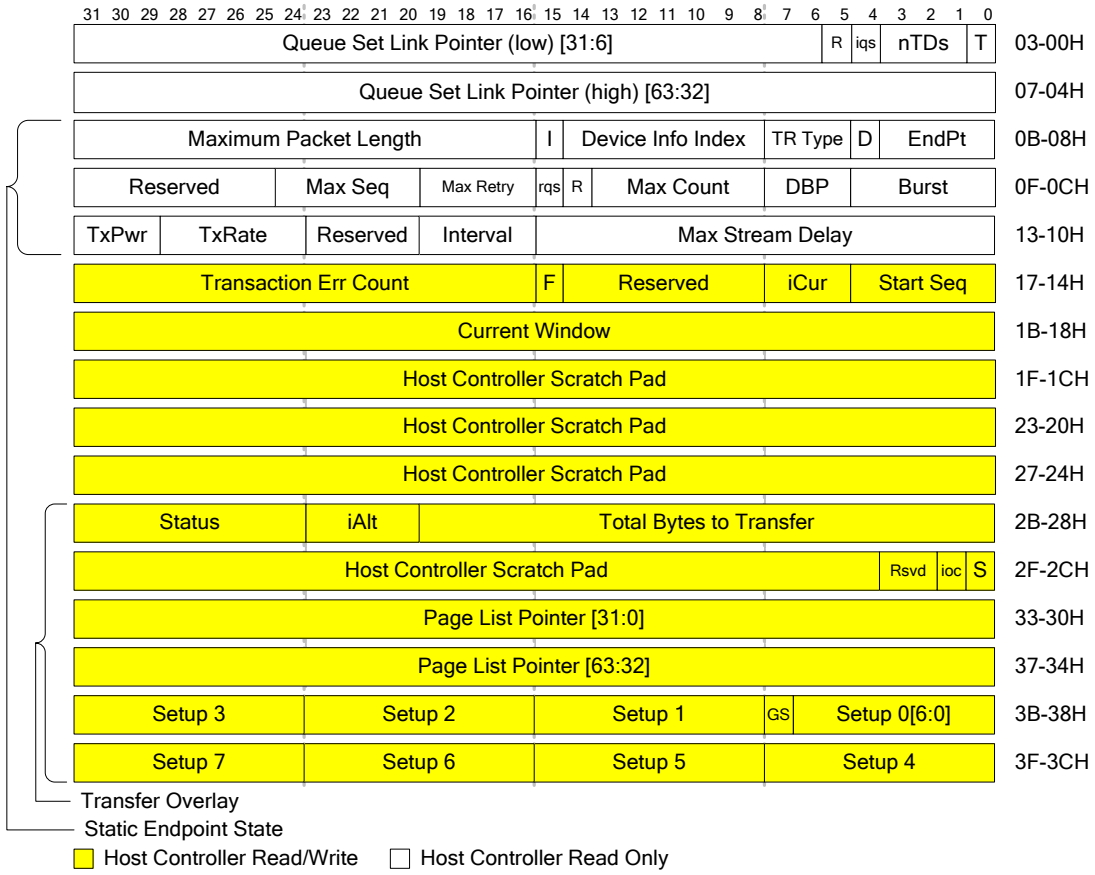


Figure 3-10. Queue Head with qTD overlay Structure Layout

3.2.6.1 Queue Set Link Pointer

The first two DWords of a QHead contain a 64-bit link pointer to the next Queue Set, as well as the control bits defined below.

This pointer may reference only another QHead (Section 3.2.6). It must not reference a qTD (Section 3.2.3).

Table 3-59. QHead (DWord 0 and 1)

Bit	Description
63:6	**Queue Set Link Pointer.** This field contains the physical memory address of the next Queue Set to be processed in the horizontal list and corresponds to memory address signals [63:6], respectively.
5	**Reserved.** This bit is reserved and must be set to zero.
4	**Isochronous Queue Set (iqs).** This field is set to a 1B if the target Queue Set is for an isochronous Queue Set. Otherwise, this field must be set to a 0B.
3:1	**Number of qTDs in Queue Set (nTDs).** This field indicates to the WHC the number of qTDs associated with the referenced QHead. The actual number of qTDs/iTDs is one more than the raw value in this field, which allows a maximum of eight.

Table 3-59. QHead (DWord 0 and 1) (cont.)

Bit	Description
0	**Terminate (T).** 1=Last QH (pointer is invalid). 0=Pointer is valid. If the QHead is in the context of the periodic list, a one bit in this field indicates to the WHC that this is the end of the periodic list. This bit is ignored by the WHC when the QHead is in the Asynchronous schedule. The WHCD must ensure that QHeads reachable by the WHC always have valid horizontal link pointers.

3.2.6.2 Endpoint Capabilities/Characteristics

The third, fourth and fifth DWords of a QHead specify static information about the endpoint. Most of the information in this region does not change over the lifetime of the endpoint. There are two types of information in this region:

- **Endpoint Characteristics.** These are the USB endpoint characteristics including addressing, transfer type and maximum sequence number.
- **Endpoint Capabilities.** These are adjustable parameters of the endpoint. They affect how the endpoint data stream is managed by the WHC. See Section 4.7.

The WHCD may modify the following fields in this region:

- *Maximum Packet Length*
- *Inactivate on Next Transaction* (I-bit)
- *Data Burst Preamble Policy*
- *Burst*
- *TxPwr*
- *TxRate*

Note that the WHCD must notify the WHC when it updates any of the fields in this region. See Section 4.6.

The WHC must not modify the bits in this region.

Table 3-60. Endpoint Capabilities/Characteristics: QHead (DWord 2)

Bit	Description
31:16	**Maximum Packet Length.** This directly corresponds to the maximum packet size of the associated endpoint (*wMaxPacketSize*). The WHCD can also specify an adjusted packet size less than the maximum packet size for Bulk and Interrupt endpoints. The maximum value this field may contain is 0xE00 (3584).
15	**Inactivate on Next Transaction (I).** This bit is used by the WHCD to request that the WHC set the *Active* bit to zero and consequently halt executing bus transactions to the associated endpoint. The WHC must set the *Active* bit to zero only on burst boundaries. See Section 4.7.6 for full operational details.
14:8	**Device Info Index.** The WHC computes the offset (*Device Info Index* << 6) into the page pointed to by the WUSBDEVICEINFOADDR register to get the physical memory address of the Device Information Buffer. Valid values are in the range of 0 through *N_DEVICES* – 1. See Section 3.2.8 for a detailed description of the Device Information Buffer.
7:5	**Transfer Type (TR Type).** These bits indicate the type of data transfer. Value encodings are: **Value** – **Meaning** 000b – Control 001b – Isochronous 010b – Bulk 011b – Interrupt 100b – Reserved 101b – Reserved 110b – Reserved 111b – Low Power Interrupt
4	**Direction.** This field indicates the data transfer direction. **Value** – **Meaning** 00b – OUT 01b – IN For Control transfers this field is ignored. The direction of the data phase of a control transfer is determined by Bit 7 (*GS* bit) of Setup packet 0. The WHCD must ensure that when TR Type field is set for Low Power Interrupt endpoints (TR_Type = 111b), this field must be set to IN (01b). A setting of OUT(00b) has undefined behavior.
3:0	**Endpoint Number (Endpt).** This 4-bit field selects the endpoint number on the device serving as the data source or sink.

Table 3-61. Endpoint Capabilities/Characteristics: QHead (DWord 3)

Bit	Description
31:25	**Reserved.** These bits are reserved and must be set to zero.
24:20	**Maximum Sequence Number (Max Seq).** This field indicates the maximum sequence number that the endpoint supports. Valid values are 1 through 31.
19:16	**Max Retry.** This field specifies the maximum number of retries for a transaction before the QHead is Halted. This field is only applicable to Control, Bulk and 'normal' Interrupt endpoints. The WHC does not use this field for Low power Interrupt endpoints. Please see section 4.7.1.2 for the WHC operational requirements.
15	**Reactivate Queue Set (rqs).** This bit is used by the WHCD to request that the WHC reactivates the Queue Set and resume the transfer. This bit is only effective when the *Active* bit is set to a zero and either the *Halted* bit or the *Queue Set Inactivated* bit is set to a one in the QHead overlay. When this bit is set to a one and the qTD/iTD pointed to by the *iCur* field is active, then the WHC will resume the transfer. See Section 4.7.8 for full operational details.
14	**Reserved.** This bit is reserved and must be set to zero.
13:8	**Maximum Packet/Transaction Count (Max Count).** This field is only applicable to Isochronous and Interrupt endpoints. For Isochronous endpoints it specifies the maximum number of packets the WHC can transfer to/from the endpoint during a Zone. For interrupt endpoints it specifies the number of attempts to perform a transaction to the endpoint during a Zone. Once a transaction is successful, the WHC must not perform any more transactions to that endpoint until the next service interval. See Section 4.4 for operation requirements.
7:5	**Data Burst Preamble Policy (DBP).**This field is used to specify the preamble to be used between data packets in a data burst. See Table 5-7 in the *Wireless USB Specification Revision 1.0* for the encodings of this field.
4:0	**Burst.** Maximum data burst size. Valid values are 1 through 16.

Table 3-62. Endpoint Capabilities/Characteristics: QHead (DWord 4)

Bit	Description
31:29	**Transmit Power (TxPwr).**This field is used to specify the power level to be used when transmitting data packets. See Section 5.2.1.2 of the *Wireless USB Specification Revision 1.0* for details on the usage of this field.
28:24	**TxRate.** This field describes the bit transfer rate to be used to perform transactions to this endpoint. This must be one of the rates supported by the PHY. It is specified as a five-bit field with the following encodings: **Value** **Meaning (Mb/s)** 00000B 53.3 00001B 80 00010B 106.7 00011B 160 00100B 200 00101B 320 00110B 400 00111B 480 01000B – 11111B Reserved
23:20	**Reserved.** These bits are reserved and must be set to zero.
19:16	**Interval.** This field specifies the interval between segments in the Segment List (See Section 3.2.5.5) expressed in 125 microsecond units. This field is only valid for Isochronous endpoints. The Interval is used as an exponent for a $2^{Interval}$ value, e.g. an Interval of 4 means a period of 2ms.
15:0	**Max Stream Delay.** This field specifies the maximum amount of stream delay in units of 125 microseconds. This field is valid only for Isochronous IN transfers. The WHC will retire this iTD according to the specified stream delay value for IN transactions. See Section 4.7.11.2 for operational requirements.

3.2.6.3 Transaction Working Space

The eleven DWords in this area represent a *transaction working space* for the WHC. The general operational model is that the WHC can detect whether the overlay area contains a description of an active transfer. If it does not contain an active transfer, then it follows the *Queue Set Link Pointer* to the next Queue Set. The WHC will never overlay the next qTD or alternate next qTD unless it is actively attempting to advance the queue (see Section 4.7.1.1.2). For the duration of the transfer, the WHC keeps the incremental status of the transfer in the overlay area. When the transfer is complete, the results are written back to the original qTD. The complete operational model of how this area is used by the WHC is described in Section 4.7.

DWord 5 of a QHead contains an index to the source qTD currently associated with the overlay. The WHC uses this index to write back the transfer status into the source qTD after the transfer is complete.

Table 3-63. Current qTD Index (DWord 5)

Bit	Description
31:16	**Transaction Err Count.** This field contains the count of the number of transaction errors encountered by the WHC. See Section 4.12.1.1 for operational requirements.
15	**Flow Control (F).** For Control and Bulk Transfer types, the WHC will not execute a data transfer to this endpoint when this bit is set to a 1. The WHCD will set this bit to 0 when the first qTD is added or to restart transactions to the device. On reception of a NAK handshake or an ACK handshake with *bvAckCode* of 0 from the device, the WHC sets this bit to 1.However if the WHC receives an ACK handshake with *bvAckCode* of 0 from the device during the Status stage of a control transfer then the WHC must not set this bit. The WHC will reset this bit to a 0 when it receives an endpoints ready notification from the device. For Interrupt Transfer type, the WHC must not execute a data transfer to this endpoint in the current service interval when this bit is set to a 1. The WHCD will set this bit to 0 when the qTD is added. On reception of a NAK handshake or an ACK handshake with *bvAckCode* of 0 from the device, the WHC sets this bit to 1. The WHC must reset this bit to a 0 when it encounters this QHead in the next service interval.
14:8	**Reserved.** These bits are reserved and must be set to zero.
7:5	**Current qTD Index (iCur).** This field contains the index of the current qTD being processed in this Queue Set.
4:0	**Start Sequence (Start Seq).** This field is used to maintain the first sequence number in the *Current Window*.

DWord 6 of a QHead contains the Current Window used for data sequence management and burst transfers. The WHC saves this value across qTDs linked to this QHead. This maintains a sliding transmit or receive window that is used to help track the data sequence number protocol.

DWords 7, 8 and 9 are intended to be used by the WHC as scratch pad work area for other sequence number tracking state. The WHCD will set the bits in this region to zero when it places a Queue Set on the schedule and must not modify any of its contents until the Queue Set is taken off the schedule.

The DWords 10-15 of a QHead are the Transfer Overlay area. This area has the same base structure as a qTD, defined in Section 3.2.4.

This area is characterized as an *overlay* because when the queue is advanced to the next qTD, the source queue element is *merged* onto this area. This area serves an execution cache for the transfer. Section 4.7.1.1.2 describes which fields in the QHead are over-written during queue advancement.

Note that the WHC can also use Bits 31:4 of DWord 11 as a scratch pad work area.

3.2.7 Queue Head (QHead) with iTD overlay

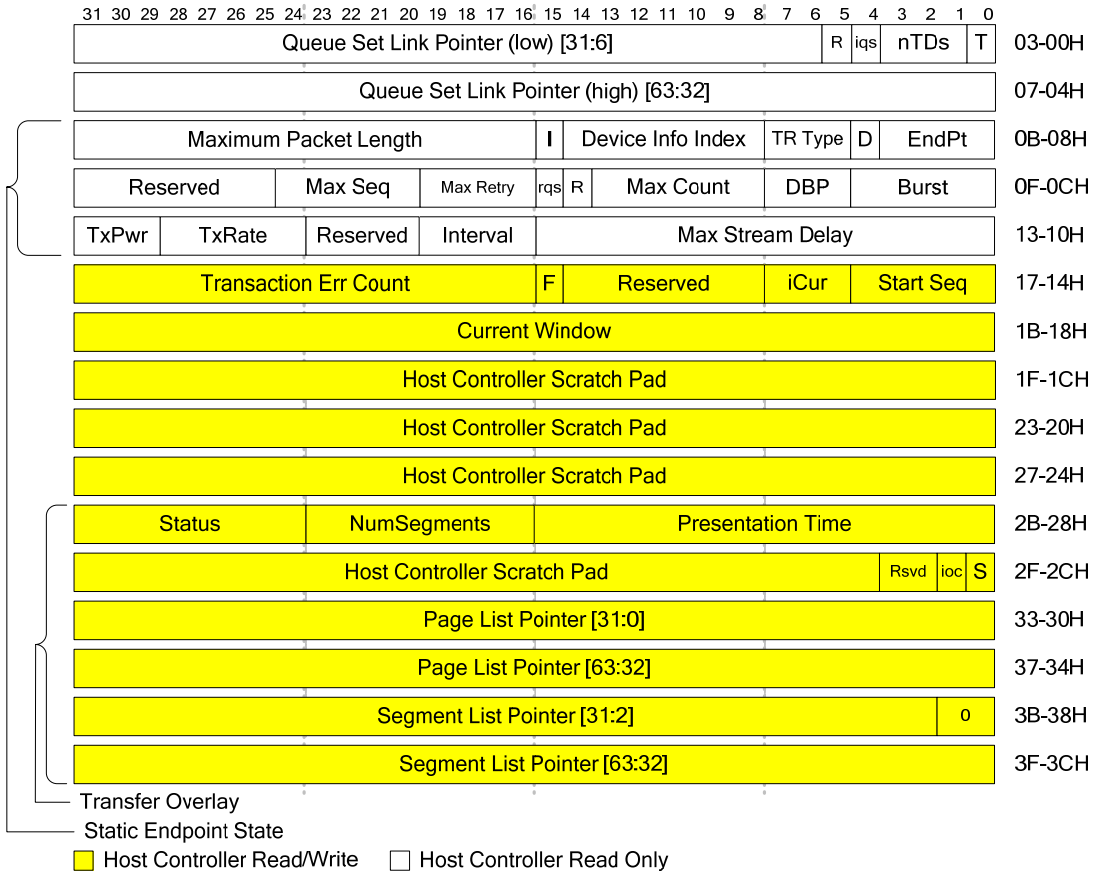


Figure 3-11. Queue Head with iTD Structure Layout

3.2.7.1 iTD Transaction Working Space

The eleven DWords in this area represent a *transaction working space* for the WHC for an Isochronous Queue Set. The general operational model is similar to the operational model for QHeads with qTDs. The complete operational model of how this area is used by the WHC is described in Section 4.7.11.

The differences in the usage of this area are documented in Table 3-64.

Table 3-64. Host-Controller Rules for Bits in iTD Overlay

DWord	Bit	Description
5	15	**Flow Control (F).** For Isochronous Transfer type, the WHC will not execute a data transfer to this endpoint in the current service interval when this bit is set to a 1. The WHCD will set this bit to 0 when the qTD is added. On reception of a NAK handshake or an ACK handshake with *bvAckCode* of 0 from the device, the WHC sets this bit to 1. The WHC will reset this bit to a 0 when it encounters this QHead in the next service interval.
10	15:0	**Presentation Time.** For OUT transactions this field is used as the presentation time in the Isochronous Packet header for the next Isochronous Data Packet that the WHC is going to send. This field is also used to determine the number of packets to discard when the value of the *Presentation Time* is less than the lower 16 bits of the current Wireless USB Channel Time 1/8th millisecond value. For IN transactions, the WHC updates this field based on the first Isochronous Data Packet it receives from the endpoint. It then uses this time along with the *Presentation Time* present in the Isochronous Packet header to determine the exact location that all subsequent packets received from this endpoint must be placed in. The WHC preserves this value across iTDs if the next iTD is Active when it retires the current iTD.
10	23:16	**NumSegments.** This field indicates the remaining number of segments in the segment list pointed to by the *Segment List Pointer* field.

3.2.8 Device Information Buffer Format

The Device Information Buffer stores all the static information about each device. The size of this buffer is determined by the number of devices that can be supported by the WHC at the same time, i.e. the *N_DEVICES* field in WHCSPARAMS register. Each device information element in this buffer is 64 bytes in length. This buffer must be at least (*N_DEVICES* X 64) bytes in size with a maximum size limited to 8K bytes. This buffer must be physically contiguous and must start on a 64 byte boundary.

This first 256 bits are the *Device Availability Info*, followed by *Device Address* (8 bits) and *Key Table Index* (8 bits) and *Secure Frame* Bit. See Section 4.7.3 for the WHC operational requirements. The structure of each element is as follows:

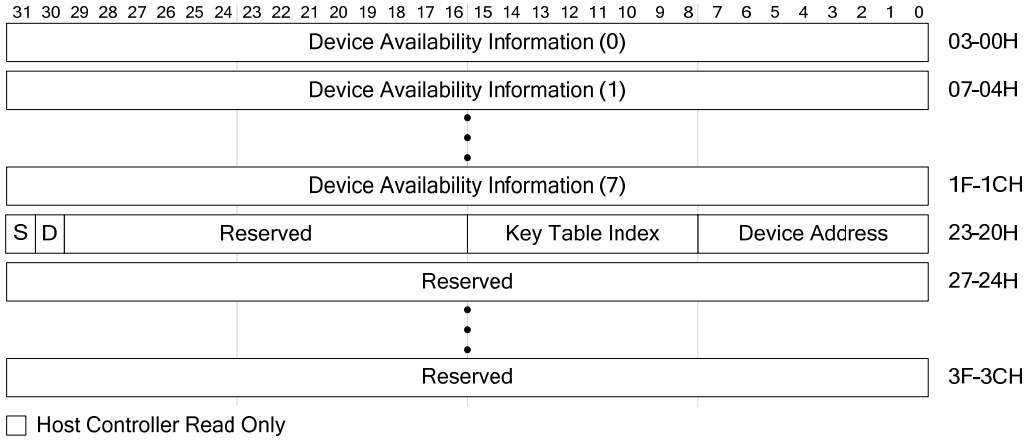


Figure 3-12. Device Information Buffer Format

3.2.8.1 Device Availability Information

The first eight DWords of the Device Information Buffer is a bit mask of the device MAS availability information. This is the time during which the WHC can communicate with this device. Each bit indicates whether the WHC can communicate with the device in that particular MAS slot. This information should be kept accurate by the WHCD.

3.2.8.2 Device Addressing and Security Information

The ninth DWord of the Device Information Buffer contains the device address. It also includes the security key table index to be used when performing secure transactions to this device.

Table 3-65. Device Addressing and Security Information (DWord 8)

Bit	Description
31	**Secure Frame Bit (S).** This bit is set if the WHC must perform secure transactions to the device. A value of zero in this field is used to indicate that the *Key Table Index* is invalid and the data must be sent unencrypted.
30	**Disable Bit (D).** If this bit is set then the WHC must not perform any transaction to this device.
29:16	**Reserved.** These bits are reserved and must be set to zero.
15:8	**Key Table Index.** This field specifies the index in the Key Table that is present within the WHC which contains the TKID and Security key to be used. This field is used to determine the encryption key to encrypt/decrypt data that is transmitted to/from this device. This is only valid if the *Secure Frame Bit* is set.
7:0	**Device Address.** This field specifies the device address.

3.2.9 Device Notification Buffer Format

Data packets received from devices during Device Notification Time Slots must be stored by the WHC into the Device Notification Buffer. The WUSBDNTSBUFADDR register contains the 64-bit physical memory address of this buffer. The only exception to this rule is the Device Endpoints Ready (DN_EPRdy) notification. The WHC must process by this notification itself and must not place it in this buffer.

The buffer is 4K bytes in length and must be page aligned. It contains a number of fixed size notification blocks and the WHC will write the Device Notification Messages received in the Device Notification Time Slots to consecutive notification blocks in this buffer. Each notification block is 64 bytes long and consists of 2 DWords of the Device Notification Message Header and 56 bytes Device Notification Data. Note that the Device Notification may be less than 56 bytes long. The structure of this buffer and the Device Notification Message is described in Figure 3-13.

Note that if there is no space available in the Device Notification Buffer, the WHC must discard the information sent by the device and stops DNTS scheduling by clearing the *Active* bit in the WUSBDNTSCTRL Register.

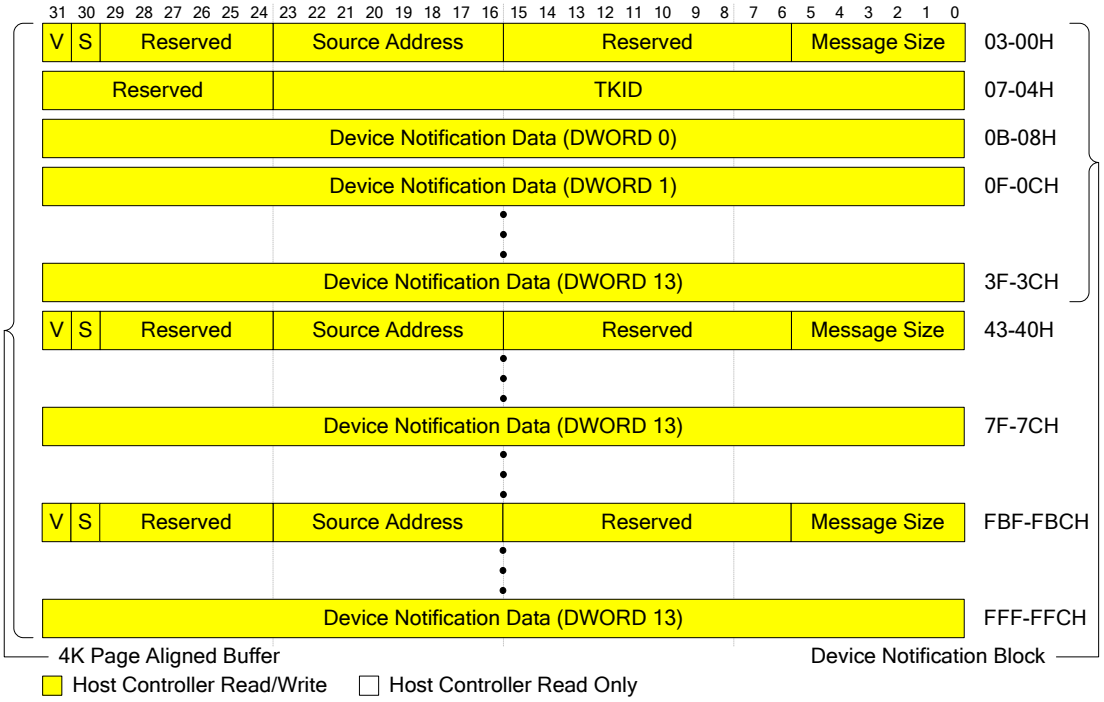


Figure 3-13. Device Notification Buffer Format

3.2.9.1 Device Notification Message Header

The WHC writes these DWords to the Device Notification Buffer when it receives a device notification message. The format of these DWords is described in Table 3-66 and Table 3-67.

Table 3-66. Device Notification Message Header (DWord 0)

Bit	Description
31	**Valid (V).** This bit indicates that this notification block contains a valid device notification message. The WHCD can determine the number of bytes by reading the *Message Size* field. The WHCD must reset this bit when it completes processing the received device notification message.
30	**Secure Frame (S).** This bit is set if this notification was received as a secure frame.
29:24	**Reserved.** These bits are reserved and must be set to zero.
23:16	**Source Address.** This field specifies the Wireless USB address of the device that sent the notification message.
15:6	**Reserved.** These bits are reserved and must be set to zero.
5:0	**Message Size.** This field indicates the number of bytes in the Device Notification Data that the device sent.

Table 3-67. Device Notification Message Header (DWord 1)

Bit	Description
31:24	**Reserved.** These bits are reserved and must be set to zero.
23:0	**TKID.** This field specifies the Temporal Key ID of the Wireless USB device that sent the notification message. This field is only valid when the *Secure Frame* bit is set.

3.2.9.2 Device Notification Data

The WHC places the payload of the received packet (*bType* field and the notification specific data) in this field of the notification block. The actual contents in this location are defined in Section 7.6 of the *Wireless USB Specification Revision 1.0*.

4. Operational Model

The general operational model details the operational requirements of the major architectural features of the UWB Multi-Interface Controller (UMC), which is comprised of (minimally) the UWB Radio Controller (URC), the Wireless USB Host Controller (WHC) and the system software modules which interact with the interfaces to these components. This section is organized with subsections devoted to architectural features of the WHC followed by those of the URC. Each section presents the operational model requirements for the hardware. Where appropriate, recommended system software operational models for features are also presented.

4.1 UMC Initialization

When the system boots, the URC is enumerated and assigned a base address for the register space. After initial power-on or after a *UWBReset* (hardware or via *UWBReset* bit in the URCCMD register), all of the URC registers and the WHC operational registers will be set to their default values as illustrated in Table 4-1 and Table 4-2. If the hardware supports other interfaces, e.g. a network interface, all the registers in those interfaces will also be reset to their default values. After hardware reset, only the operational registers not contained in the Auxiliary power well will be at their default values.

The WHC operational registers can also be reset to their default values by setting the *WHCReset* bit in the WUSBCMD register to a one. This operation does not affect registers in other interfaces including the URC registers.

Note that every interface that is exposed by the UMC must provide a ‘reset’ control for the interface.

Table 4-1. Default Values of URC Register Space

URC Register	Default Value (after Reset)
URCCMD	00000000h
URCSTS	00010000h
URCINTR	00000000h
URCCMDADDR	Undefined
URCEVTADDR	Undefined

Table 4-2. Default Values of WHC Operational Register Space

WHC Operational Register	Default Value (after Reset)
WUSBCMD	00000000h
WUSBSTS	00001000h
WUSBINTR	00000000h
WUSBGENCMDSTS	00000000h
WUSBGENCMDPARAMS	Undefined
WUSBGENADDR	Undefined
WUSBASYNCLISTADDR	Undefined
WUSBDNTSBUFADDR	Undefined
WUSBDEVICEINFOADDR	Undefined
WUSBSETSECKEYCMD	00000000h
WUSBTKID	Undefined
WUSBSECKEY	Undefined

Table 4-2. Default Values of WHC Operational Register Space (cont.)

WHC Operational Register	Default Value (after Reset)
WUSBPERIODICLISTBASE	Undefined
WUSBMASINDEX	Undefined
WUSBDNTSCTRL	00000000h
WUSBTIME	00000000h
WUSBBPST	00000000h
WUSBDIBUPDATED	00000000000000000000000000000000h

Section 4.1.1 describes in detail the steps required to initialize the URC and Section 4.1.2 describes the initialization sequence for the WHC.

4.1.1 URC Initialization

The steps required to correctly initialize the URC are illustrated in Figure 4-1.

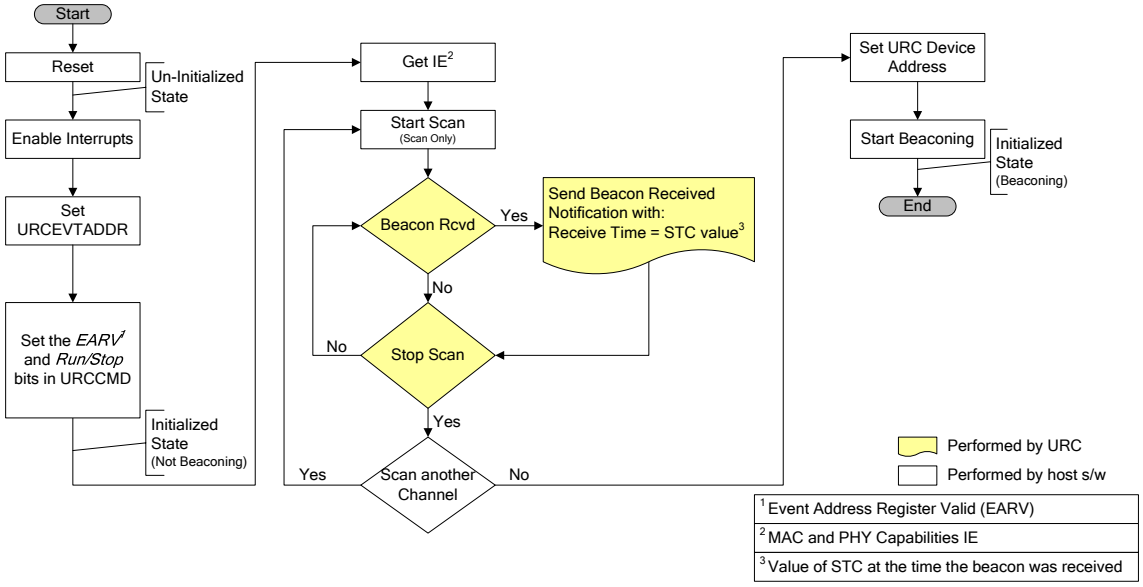


Figure 4-1. URC Initialization Sequence Flowchart

After resetting the URC, the URCD performs the following steps:

- Write the appropriate value to the URCINTR register to enable the appropriate interrupts.
- Write the base address of an Event Buffer to the URCEVTADDR register.
- Set the *Event Address Register Valid* and the *Run/Stop* bits to one in the URCCMD register. Setting the *Run/Stop* bit to one turns the URC *ON*.

At this point, the URC is ready to generate radio control events to the URCD. In order to issue a radio control command, the URCD must prepare a buffer containing a command block. The URCD must then set the size of the command buffer and enable the command (set the *Active* bit) by writing to the URCCMD register. The URCD must set the *Interrupt when Ready* bit in order to receive an interrupt when the URC is ready to accept the next command. To complete the initialization and to start sending beacons the URCD must do at least the following:

- Issue a Get IE command (see Section 4.13.6.1) to extract the MAC and PHY Capabilities IE. The URCD can then derive the UWB channels supported by the implementation.
- Issue a Scan command (see Section 4.13.4) with the scan state set to SCAN_ONLY on a UWB channel supported by the URC.
- If the URC receives one or more beacons (during the scan) from devices on the channel, a Beacon Received notification (see Section 4.13.11.2) is generated for every received beacon. The receive time in the Beacon Received notifications must be set to set to the value of the Superframe Time Counter (STC). See Section 4.13.2 for details on the STC.
- The scan will continue until the URCD issues a Scan command with the scan state set to SCAN_DISABLED which will stop the scan. After scanning on one of more channels, the URCD will determine which channel it would like to use for the UWB Radio communication.
- The URCD may modify any of the optional capabilities reported by the URC (reported in the MAC and PHY Capabilities IE) and set any other IEs that it wants the URC to send in the beacon using the Set IE command (see Section 4.13.6.2).
- The URCD must set the URC device address using the Set Device Address Management command (see Section 4.13.7).
- The URCD must instruct the URC to start sending beacons by using the Start Beaconing command (see Section 4.13.5.1).

After initialization of the URC, system software will read the UWB Interface Capability Registers and enumerate the other interfaces exposed by the hardware.

4.1.2 WHC Initialization

In order to initialize the WHC, the WHCD must perform the following steps:

- Write a value to the WUSBINTR register to enable the desired interrupts.
- Write the base address of the Periodic Zone List to the WUSBPERIODICLISTBASE register. All elements of the Periodic Zone List should have their T-Bits set to a one.
- Write the base address of the Device Notification Buffer to the WUSBDNTSBUFADDR register.
- Write the base address of the Device Information Buffer to the WUSBDEVICEINFOADDR register.
- Write the WUSBCMD register to set the Broadcast Cluster ID, Stream Index and turn the WHC ON via setting the *Run/Stop* bit to a one.
- Use the WUSB Generic Command (see Section 4.8.1) to set Information Elements to be broadcast within MMC packets.
- Use the Set Encryption Key Command (see Section 4.9.3) to set the group key.
- Establish a DRP for the Wireless USB Channel using the URC Commands, then use the WUSB Generic Command (see Section 4.8.2) to set the MAS slot information.

At this point, the WHC is up and running and will transmit MMC packets during Media Access Slots reserved for the Wireless USB cluster. In this specification, a set of MAS reserved for the Wireless USB cluster is referred to as a Wireless USB Reservation. The WHCD may then start to communicate with Wireless USB devices by doing the following:

- Use the WUSB Generic Command to set a Host IE with the appropriate attributes set to indicate that it is available for connections (either new connects or reconnects and connects only).
- Instruct the WHC to include a DNTS period in the MMCs it is transmitting. The frequency and number of slots in the DNTS period is set by the WHCD by setting the *Number of Slots* and *Interval* fields in the WUSBDNTSCTRL register.

- Use the WUSB Generic command to send a CONNECT_ACK IE to a DN_Connect notification if it receives one in the Device Notification Buffer.
- Use the asynchronous schedule (see Section 4.5) to communicate with the control endpoint on the device and perform a 4 way handshake. This schedule is also used to communicate with any other Bulk or Control endpoints the device might include in its Interface descriptors.
- Use the Set Encryption Key Command (see Section 4.9.2) to set the PTK to be used for all data transfers with the device.
- Use the Asynchronous schedule (see Section 4.5) to communicate with the control and/or bulk endpoints in the device.
- Finally use the periodic schedule (see Section 4.4) to communicate with any Isochronous or Interrupt endpoints the device exposes in its Interface descriptors.

4.2 WUSB Channel Time

The WHC must maintain a 24-bit free running timer in the WUSBTIME register. This timer is defined as the Wireless USB Channel Time in the *Wireless Universal Serial Bus Specification, Revision 1.0*. The Wireless USB Channel Time consists of a 17-bit 1/8th millisecond counter and a 7-bit microsecond counter. The default value of this register after reset (*UWBReset* or *WHCReset*) is 00000000h. The WHC must start advancing the timer when the *Run/Stop* bit in the WUSBCMD register transitions to a one (1B) and must increment the microsecond counter by one every microsecond. The microsecond counter counts from 0 to 124, and wraps back to zero every 125 microseconds. When the microsecond counter transitions from 124 to 0, the WHC increments the 1/8th millisecond counter by one. The 1/8th millisecond counter wraps back to 0 from the value of all ones (1FFFFh). When the WUSB Channel Time wraps around to 0, the WHC sets the *Channel Time Rollover* bit in the WUSBSTS register to a one and generates an interrupt if the *Channel Time Rollover Enable* bit is set to a one in the WUSBINTR register.

The WHC must use the value in this register to set the *WUSB Channel Time Stamp* field in MMC packets. The WHC also uses this register to determine the deadline of isochronous packets. See Section 4.7.11 for the detailed operations.

In addition to the WUSBTIME register, the WHC must update the *BPST Channel Time* field of the WUSBBPST register with the calculated value of the WUSB Channel Time at the BPST (Beacon Period Start Time) of the next superframe. The URC is responsible for maintaining the BPST in every superframe by monitoring the incoming beacons from its neighbor devices. It (URC) must adjust its BPST when it receives a beacon from a device which has a slower clock as per the WiMedia rules. At the end of the Beacon Period, the WHC must do the following:

- Update the WUSBBPST register with the calculated value of WUSB Channel Time at the BPST of the next superframe.
- Determine the difference between the final BPST after the Beacon Period and the original BPST before the Beacon Period and set this value in the *BPST Adjustment* field of the WUSBBPST register.
- If the adjustment value has changed from the previous superframe, the WHC must set the *BPST Adjustment Changed* bit in the WUSBSTS register to a one. In addition the WHC must generate an interrupt if the *BPST Adjustment Changed Enable* bit is set to a one in the WUSBINTR register.

4.3 Schedule Traversal Rules

The WHC executes transactions for devices using a simple, shared-memory schedule. The schedule is comprised of a few data structures, organized into two distinct lists. The data structures are designed to provide the maximum flexibility required by Wireless USB, minimize memory traffic and minimize hardware/software complexity.

The WHCD maintains two schedules for the WHC a periodic schedule and an asynchronous schedule. The root of the periodic schedule (see Section 4.4) is the WUSBPERIODICLISTBASE register (see Section

2.4.15) and the WUSBASYNCLISTADDR register (see Section 2.4.9) contains the address of the start of the asynchronous schedule (see Section 4.5).

Each schedule is composed of linked lists of data structures, nominally Queue Sets (one Queue Set per active endpoint). The WHC must traverse these schedules in order to acquire contexts to perform transactions to endpoints. The WHC can mix transactions from the periodic and asynchronous schedules in the same transaction group. However, transactions for Queue Sets on the periodic list have a higher priority than transactions for Queue Sets on the asynchronous schedule.

4.4 Periodic Schedule

The WUSBPERIODICLISTBASE register is the physical memory base address of the periodic zone list. This list is an array of physical memory pointers. The objects (also called Queue Sets) referenced from the zone list must be valid schedule data structures as defined in Chapter 3. In each zone, i.e. 16 consecutive MAS slots in a superframe, if the periodic schedule is enabled then the WHC must execute transactions for active Queue Sets on the periodic schedule. The WHC traverses the periodic schedule by constructing an array offset reference from the WUSBPERIODICLISTBASE and the WUSBMASINDEX registers (see Section 2.4.16). It fetches the element and begins traversing the graph of linked schedule data structures. The WUSBMASINDEX register is reset to a zero at the beginning of a superframe and incremented by one whenever it crosses a MAS boundary, i.e. every 256 microseconds.

The last Queue Set in a zone is identified by the *next* link pointer in the schedule data structure having its *T-bit* set to a one. When the WHC encounters a *T-Bit* set to a one during a horizontal traversal of the periodic list, it will start traversing the periodic zone list again until it determines there is no periodic transaction available in the current MAS slot. This causes the WHC to cease working on the periodic schedule and transitions immediately to traversing the asynchronous schedule. Note that the WHC may schedule transactions for Queue Sets that cross MAS boundaries as long as both of the MASs are owned by the WHC.

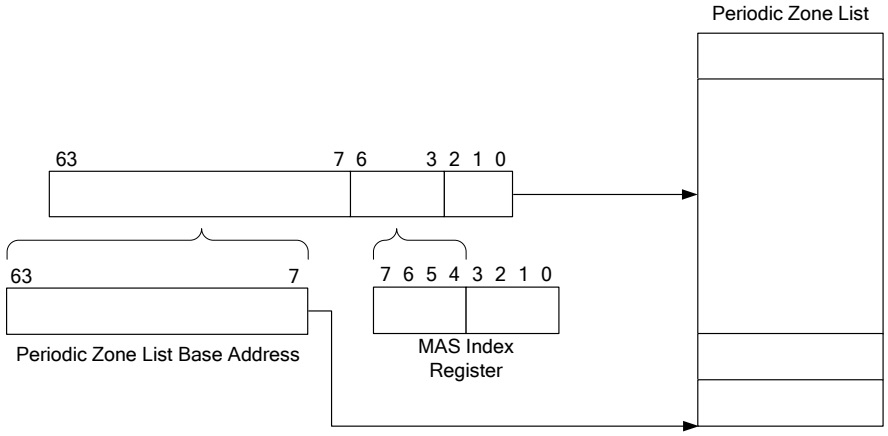


Figure 4-2. Derivation of Pointers into Periodic Zone List Array

The periodic schedule traversal is enabled or disabled via the *Periodic Schedule Enable* bit in the WUSBCMD register. If the *Periodic Schedule Enable* bit is set to a zero, then the WHC will not access the periodic zone list via the WUSBPERIODICLISTBASE register. Likewise, when the *Periodic Schedule Enable* bit is a one, then the WHC will use the WUSBPERIODICLISTBASE register to traverse the periodic schedule. Note that the WHC is not required to react to modifications to the *Periodic Schedule Enable* immediately.

The *Periodic Schedule Status* bit in the WUSBSTS register indicates the current status of the periodic schedule. The WHCD enables (or disables) the periodic schedule by writing a one (or zero) to the *Periodic Schedule Enable* bit in the WUSBCMD register. The WHCD can then poll the *Periodic Schedule Status* bit to determine when the periodic schedule has made the desired transition. The WHCD must not modify the *Periodic Schedule Enable* bit unless the value of the *Periodic Schedule Enable* bit equals that of the *Periodic Schedule Status* bit.

The periodic schedule is used to manage all isochronous and interrupt transfer streams. The base of the periodic schedule is the periodic zone list. The WHCD links schedule data structures to the periodic zone list to produce a graph of scheduled data structures. The graph represents an appropriate sequence of transactions on the Wireless USB channel. Figure 4-3 illustrates Queue Sets for isochronous and interrupt transfers linked to the periodic zone list. They are linked into the zone list ordered by poll rate. Longer poll rates are linked first (e.g. closest to the periodic zone list), followed by shorter poll rates, with Queue Sets with the shortest poll rate (polled in every zone) on the very end.

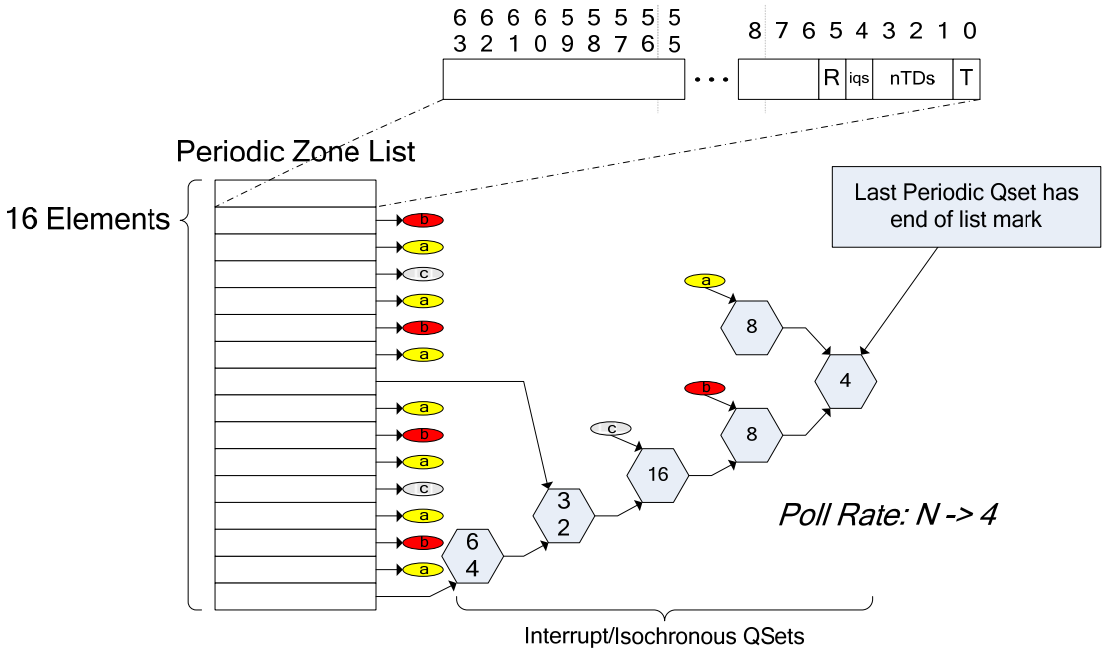


Figure 4-3. Example Periodic Schedule

The link path(s) from the periodic zone list to a Queue Set establishes in which zones a transaction will be executed for a Queue Set. A Queue Set is linked into the periodic schedule so it is polled at the appropriate rate. For isochronous endpoints, the WHCD sets the *Max Count* field in the QHead in order to limit the maximum number of packets that the WHC can transmit to or receive from the endpoint during the Wireless USB Reservation in a zone. However, if there is bandwidth available before the end of the Wireless USB Reservation in a zone, it may perform more transactions to an Isochronous endpoint. For interrupt endpoints, the WHCD sets the *Max Count* field to limit the maximum number of transactions that the WHC can perform to/from the endpoint during the Wireless USB Reservation in a zone. The WHC may perform up to *Max Count* – 1 transaction retries if the first transaction in that Wireless USB Reservation in the zone for the endpoint was unsuccessful. If any transaction to an Interrupt endpoint is successful, then the WHC must not perform any more transactions to that endpoint. The WHCD is responsible for managing bandwidth allocations for the Wireless USB channel for the required number of transactions.

Note that if the WHCD modifies the periodic schedule it must set the *Periodic Schedule Updated* bit in the WUSBCMD register. See Section 4.6 for detailed information.

4.5 Asynchronous Schedule

The WHC uses the operational register WUSBASYNCLISTADDR to access the asynchronous schedule.

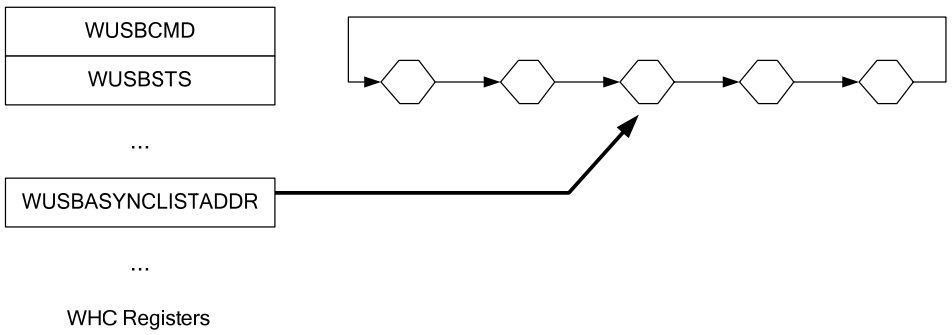


Figure 4-4. General Format of Asynchronous Schedule List

The WUSBASYNCLISTADDR register contains a physical memory pointer to the *next* Queue Set. When the WHC makes a transition to executing the asynchronous schedule, it begins by reading the Queue Set referenced by the WUSBASYNCLISTADDR register. The WHCD must set the Queue Set link pointer *T-bits* to a zero for Queue Sets in the asynchronous schedule.

The Asynchronous schedule traversal is enabled or disabled via the *Asynchronous Schedule Enable* bit in the WUSBCMD register. If the *Asynchronous Schedule Enable* bit is set to a zero, then the WHC will not access the asynchronous schedule via the WUSBASYNCLISTADDR register. Likewise, when the *Asynchronous Schedule Enable* bit is a one, then the WHC will use the WUSBASYNCLISTADDR register to traverse the asynchronous schedule. Modifications to the *Asynchronous Schedule Enable* bit are not necessarily immediate. Rather the new value of the bit will only be taken into consideration the next time the WHC needs to use the value of the WUSBASYNCLISTADDR register to get the next QHead.

The *Asynchronous Schedule Status* bit in the WUSBSTS register indicates the current status of the asynchronous schedule. The WHCD enables (or disables) the asynchronous schedule by writing a one (or zero) to the *Asynchronous Schedule Enable* bit in the WUSBCMD register. The WHCD can then poll the *Asynchronous Schedule Status* bit to determine when the asynchronous schedule has made the desired transition. The WHCD must not modify the *Asynchronous Schedule Enable* bit unless the value of the *Asynchronous Schedule Enable* bit equals that of the *Asynchronous Schedule Status* bit.

The asynchronous schedule is used to manage all Control and Bulk transfers. Control and Bulk transfers are managed using Queue Sets. The asynchronous schedule is based at the WUSBASYNCLISTADDR register. The default value of the WUSBASYNCLISTADDR register after reset is undefined and the schedule is disabled when the *Asynchronous Schedule Enable* bit is a zero.

The WHCD may only write to the WUSBASYNCLISTADDR register with defined results when the schedule is disabled .e.g. *Asynchronous Schedule Enable* bit in the WUSBCMD register and the *Asynchronous Schedule Status* bit in the WUSBSTS register are zero. The WHCD enables execution from the asynchronous schedule by writing a valid memory address (of a Queue Set) into this register. Then the WHCD enables the asynchronous schedule by setting the *Asynchronous Schedule Enable* bit to one. The asynchronous schedule is actually enabled when the *Asynchronous Schedule Status* bit is a one.

When the WHC begins traversing the asynchronous schedule, it begins by using the value of the WUSBASYNCLISTADDR register. It reads the first referenced data structure and begins executing transactions and traversing the linked list as appropriate. When the WHC "completes" processing the asynchronous schedule, it retains the value of the last accessed QHead's *Queue Set Link Pointer* in the WUSBASYNCLISTADDR register. Next time the asynchronous schedule is accessed; this is the first data structure that will be read. This provides round-robin fairness for traversing the asynchronous schedule.

The Queue Sets in the asynchronous list are linked into a simple circular list as shown in Figure 4-4. Queue Sets are the only valid data structures that may be linked into the asynchronous schedule. A Queue Set Link

Pointer with the *iqs* bit set to a one (i.e. a Queue Set with one or more isochronous transfer descriptors) in the asynchronous schedule yields undefined results.

The *Maximum Packet Length* field in a QHead is sized to accommodate the use of this data structure for all transfer types. The *Wireless USB Specification, Revision 1.0* specifies the maximum packet sizes for all transfer types. The WHCD should always parameterize the QHead data structures according to the core specification requirements.

Note that if the WHCD modifies the asynchronous schedule it must set the *Asynchronous Schedule Updated* bit in the WUSBCMD register. See Section 4.6 for detailed information.

4.6 Updating Asynchronous and Periodic Schedules

WHC implementations can reduce the number of main memory accesses they perform by caching some or all of the schedule data structures. To enable this, the WHCD must inform the WHC whenever it modifies the schedule. There are two control bits in the WUSBCMD register used to notify the WHC of changes to the schedules:

- *Asynchronous Schedule Updated* bit for modifications to the asynchronous schedule and
- *Periodic Schedule Updated* bit for modifications to the periodic schedule.

The WHCD must set one of the above bits to a one depending on the schedule that is being updated if:

- it modifies any part of the static (Endpoint Capabilities/Characteristics) portion of a QHead or
- it adds or removes a Queue Set to or from a schedule or
- it activates a previously inactive qTD (or iTD) which is at the location pointed to by *iCur* field in the QHead.

Note that if one or more Queue Sets were removed from the Asynchronous schedule then the WHCD must also set the *Asynchronous Schedule QSet Removed* bit to a 1B. Similarly if one or more Queue Sets were removed from the Periodic schedule then the WHCD must also set the *Periodic Schedule QSet Removed* bit to a 1B. These bits must be set in addition to the above mentioned control bits in the WUSBCMD register.

The WHCD must not modify the asynchronous (periodic) schedule while the *Asynchronous (Periodic) Schedule Updated* bit is set to a one, i.e. the WHCD must not perform any operation that requires setting the *Asynchronous (Periodic) Schedule Updated* bit to a one, however the WHCD can activate a qTD/iTD any time regardless of the value of this bit as long as it is not pointed to by the *iCur* field in the containing Queue Set.

When the WHC detects the *Asynchronous (Periodic) Schedule Updated* bit is set to a one, it will synchronize its internal cache of the asynchronous (periodic) schedule to that in main memory. When it completes the synchronization of the schedule(s) it must do the following:

- Set the *Asynchronous (Periodic) Schedule Updated* bit to a 0B.
- If the Set *Asynchronous (Periodic) Schedule QSet Removed* bit was set to one then the WHC must also reset this bit to a zero.
- Determine whether to generate an interrupt based on the following rules:
    - The *Interrupt on Asynch (Periodic) Schedule Synched Enable* bit is set to a one in the WUSBINTR register.
    - The *Interrupt on Asynch (Periodic) Schedule Synched Doorbell* bit to a one in the WUSBCMD register.
- Set the *Interrupt on Asynch (Periodic) Schedule Synched Doorbell* bit to zero in the WUSBCMD register
- If the WHC must generate an interrupt then it must do the following:

- Sets the *Interrupt on Asynch (Periodic) Schedule Synched* bit to a one in the WUSBSTS register
- Generate the interrupt.

4.7 Managing Transfers via Queue Sets

This section presents an overview of how the WHC must service Queue Sets.

A Queue Set is used to describe transactions to or from an endpoint. It consists of a QHead (see Section 3.2.6) and one or more qTD structures defined in Section 3.2.4. For isochronous endpoints, a Queue Set contains one or more iTD structures defined in Section 3.2.5. A QHead and its associated transfer descriptor data structures are concatenated in a physically contiguous memory space. The total number of qTDs or iTDs in the Queue Set is indicated by the *nTDs* field in the referencing *Queue Set Link Pointer*. The QHead structure contains static endpoint characteristics and capabilities and a working area from where individual bus transactions for an endpoint are executed (see Overlay area defined in Figure 3.2.8 and Figure 3.2.9). Each qTD/iTD represents one or more bus transactions, which is defined in the context of this specification as a *transfer*. The WHC always executes a transaction for the qTD/iTD pointed to by the *Current qTD Index (iCur)* field in the QHead if it is in the active state (i.e. *Active* bit in the qTD/iTD set to a one).

The WHC uses one or more Queue Sets to create a transaction group. The general processing model for the WHC's use of Queue Sets is:

- Read one or more Queue Sets and create a transaction group.
- Execute the transaction group (moving data into/out of the system as required).
- Write back the transaction results of the transactions to the overlay area of the individual QHeads.
- Repeat the above steps, until the end condition (as per the schedule type) is reached. See Section 4.4 and Section 4.5.

If the WHC encounters errors during a transaction, the WHC will set one (or more) of the error reporting bits in the QHead's *Status* field. The *Status* field accumulates all errors encountered during the execution of a qTD/iTD (i.e. the error bits in the QHead *Status* field are 'sticky' until the transfer (qTD/iTD) has completed). This status must be written back to the source qTD/iTD when the transfer is complete.

On successful transfer completions, the WHC must auto-advance (without intervention by the WHCD) to the next qTD/iTD by incrementing the *iCur* field by one. If the value of *iCur* is greater than the value of *nTDs* in the Queue Set Link Pointer to this Queue Set, then *iCur* must be reset to zero. If the transfer completed unsuccessfully, then the WHC must halt the queue so no additional bus transactions will occur for the endpoint and the WHC will not advance the queue.

4.7.1 Example Queue Set Traversal State Machine

An example WHC operational state machine of Queue Sets traversal is illustrated in Figure 4-5. This state machine is a model for how a WHC might traverse one or more Queue Sets. The *Fetch QHs* state (Section 4.7.1.1) consists of a number of internal states.

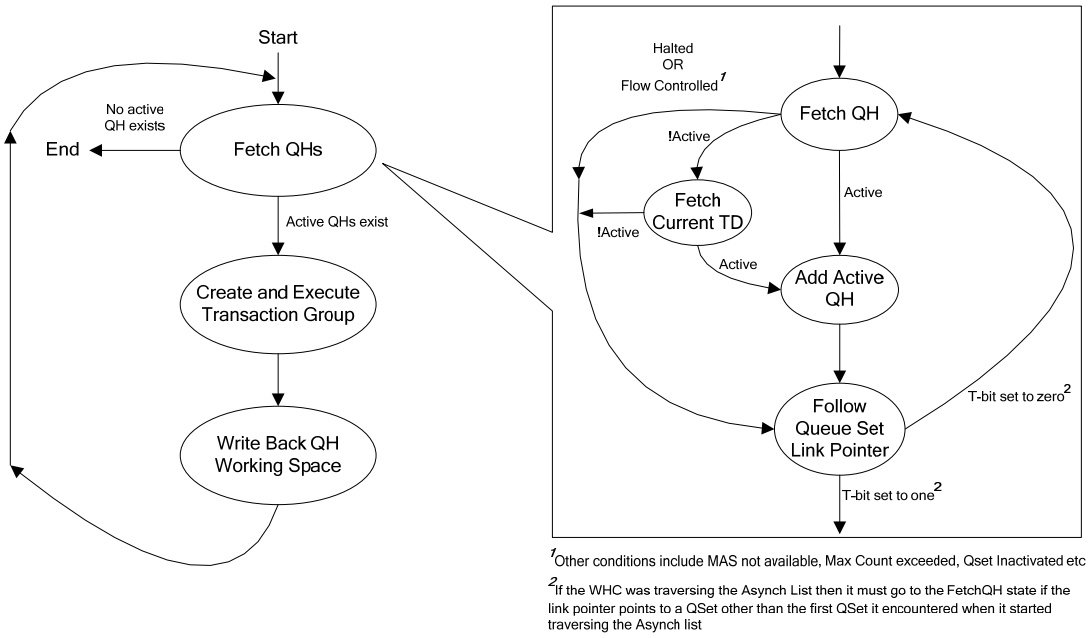


Figure 4-5. WHC Queue Set Traversal State Machine

This traversal state machine applies to all QHeads, regardless of transfer type. The following sections describe each state. Each state description describes the entry criteria. The *Execute Transaction Group* state (Section 4.7.1.2) describes the basic requirement for all endpoints.

Note: Prior to the WHCD placing a QHead into either the periodic or asynchronous list, the WHCD must ensure that the QHead is properly initialized. Minimally, the QHead should be initialized to the following (see Section 3.2.6 or Section 3.2.7 for layout of a QHead):

- Valid static endpoint state.
- For the very first use of a QHead, the WHCD must zero-out the QHead transfer overlay, i.e. the *iCur* field is initialized to a zero, then prepare a valid qTD/iTD in the first qTD/iTD location in the Queue Set.
- The *Current Window* must be initialized according to the maximum burst size in the *Burst* field, i.e. the least significant *Burst* number of bits must be set to one. For example, when the maximum burst size is 4, the *Current Window* must be initialized to 0000000Fh.
- The *Host Controller Scratch Pad* fields must be initialized to zero.

4.7.1.1 Fetch QHeads

In this state, the WHC fetches one or more QHeads that are used to construct a transaction group. This state is divided into the following sub-states:

- Fetch QH.
- Fetch Current TD.
- Add Active QH.

- Follow Queue Set Link Pointer.

This state is complete when there is no more active QHead in the list, either the periodic zone list or the asynchronous list, or the WHC has as many active QHeads on-chip as needs for the transaction group it is creating. Following sections describe the operations in each sub-state.

4.7.1.1.1 Fetch QH

A QHead can be referenced from the physical address stored in the WUSBASYNCLISTADDR register (Section 2.4.9), periodic zone list element or the *Queue Set Link Pointer* field of another QHead. A referenced data structure is a QHead with qTD overlay defined in Figure 3-10 if the *iqs* bit is set to a zero, or a QHead with iTD overlay defined in Figure 3-11 if the *iqs* bit is set to a one. If the QHead is referenced by the WUSBASYNCLISTADDR register, it is always the QHead with qTD overlay.

For periodic endpoints, if it is the first fetch of this QHead in a Zone, the WHC must reset the *Flow Control* bit in the QHead. The WHC must ensure that it performs the number of transactions that are required for this QHead as per the value in the *Max Count* field. For an IN transfer it must also perform a blank transaction to an endpoint after it completes the last transaction to the endpoint in this service interval.

After the QHead has been fetched, the WHC conducts the following queries to see if it can schedule a transaction to/from this endpoint. The QHead is a candidate for execution if none of the following criteria are met:

- If the Medium Access Slot is not available for this device as per the *Device Availability Information* field in the Device Information Buffer.
- If the *Active* bit is set to a zero, the *Queue Set Inactivated* bit is set to a one and the *Reactivate Queue Set (rqs)* bit is set to a zero, this endpoint has been inactivated as requested by the WHCD.
- For bulk, control or interrupt endpoints, if the *Halted* bit is set to a one and the *Reactivate Queue Set (rqs)* bit is set to a zero.
- If the *Flow Control (FC)* bit is set to a one.
- For periodic endpoints (isochronous and interrupt), if the WHC has performed *Max Count* number of transactions to this endpoint in this zone unless it only needs to perform a blank transaction.
- The device is disabled as per the *Disabled* bit in the Device Information Buffer.

4.7.1.1.2 Fetch Current TD

If the QHead fetched is a candidate for execution, but is not active, the WHC will fetch the qTD/iTD pointed to by the *iCur* field and determines whether or not to perform an overlay.

If the fetched qTD/iTD has its *Active* bit set to a one and the *Reactivate Queue Set* bit is set to a zero, the WHC performs the overlay. The WHC performs the overlay based on the following rules:

- For control transfers, the *Current Window* field is initialized with the least significant *Burst* number of bits set to a one and the *Start Seq* field is set to a zero. The WHC must send the Setup bytes present in the qTD to the device as well as perform a Status phase transaction on completion of the data phase if there is one.
- For isochronous IN transfers the *Presentation Time* field for the next iTD is calculated based on the *Presentation Time* field in the current overlay area. See Section 4.7.11.2 for details.
- All the other areas of the overlay are set by the incoming qTD/iTD (except the Host Controller Scratch Pad area).

If the fetched qTD/iTD has its *Active* bit set to a one and the *Reactivate Queue Set* bit is set to a one, the WHC only overlays the first DWord of the qTD/iTD and all the other fields must be retained.

If the fetched qTD/iTD has its *Active* bit set to a zero and it is an IN transfer and the WHC needs to perform a blank transaction (see Section 4.7.5) to the endpoint then the QHead must be added to the active list.

If the fetched qTD/iTD has its *Active* bit set to a zero, the WHC aborts the QHead overlay and follows the Queue Set Link Pointer to the next schedule data structure.

4.7.1.1.3 Add Active QH

The WHC enters this state from the *Fetch QH or Fetch Current TD* state. It adds this QHead to the list of QHeads that will be used to create the transaction group.

The WHC exits this state when the QHead's information has been accumulated.

4.7.1.1.4 Follow Queue Set Link Pointer

The WHC enters this state when any of the following conditions exist:

- When the WHC exits the *Add Active QH* state.
- If the current qTD/iTD is not active in the *Fetch Current TD* state.

The WHC uses the *Queue Set Link Pointer* in the QHead to determine the next schedule data structure to fetch. It must use this pointer only if:

- the QHead is on the periodic zone list and the T-bit is set to zero or
- the QHead is on the asynchronous list and if the *Queue Set Link Pointer* points to a Queue Set other than the first Queue Set it encountered when it started traversing this list

The WHC must also follow the Queue Set Link Pointer if any of the criteria mentioned in Section 4.7.1.1.1 is met.

4.7.1.2 Create and Execute Transaction Group

The WHC enters this state from the *Fetch Queue Heads* state only if at least one Queue Set is available to perform a transaction in the current MAS slot.

On entry to this state, the WHC will create a transaction group. A transaction will be added to this group if the WHC is able to perform the transaction in the time available for the transaction group. The WHC uses the following parameters in the QHead to determine whether a transaction will fit in the current transaction group:

- For bulk, control and interrupt endpoints, the WHC uses the *Maximum Packet Length*, the *TxRate*, the *Burst*, the *Current Window* and the *Total Bytes to Transfer*.
- For isochronous endpoints, the WHC uses the *Maximum Packet Length*, the *TxRate*, the *Burst* and the *Current Window* fields. It may also use the *Segment Length* fields in the active segment list elements.

Note that the WHC must schedule all the OUT transactions first followed by IN transactions in a transaction group to minimize the direction switching.

Once it has created the transaction group, the WHC begins executing the transaction group with one or more transactions using the endpoint information in the QHeads. It generates a WCTA IE including a number of W_XCTAs for the QHeads:

- For an active QHead for an OUT transfer, the WHC generates a W_{DR}CTA for data and a W_{DT}CTA for handshake if the WHC received a handshake packet in the previous transaction to this endpoint. If the handshake packet was not received, the WHC only generates a W_{DT}CTA to receive a handshake packet and update the *Current Window*.
- For an active QHead for an IN transfer, the WHC generates a W_{DT}CTA.
- For an inactive QHead for an IN transfer, with a pending blank transaction, the WHC generates a Blank W_{DT}CTA (see Section 4.7.5).
- The WHC may insert a W_{DNTS}CTA when the DNTS scheduling is enabled (see Section 4.10).

If one or more data packets were successfully transferred during the transaction, the transfer state in the overlay area is advanced. To advance the QHead's transfer state, the *Total Bytes to Transfer* field is decremented by the number of bytes moved in the transaction, the *Current Window* and *Start Seq* fields are advanced according to the packet's sequence numbers.

Note that the *Total Bytes to Transfer* field may be zero when all the other criteria for executing a transaction are met. When this occurs, the WHC will execute a zero-length transaction to the endpoint. If the *Direction* bit indicates an IN transaction and the device delivers data, the WHC will detect a packet babble condition, set the *Babble Detected* and *Halted* bits in the *Status* field, set the *Active* bit to a zero and write back the results to the source qTD. The WHC will not perform any more transactions to this endpoint without intervention by the WHCD.

The number of bytes moved during an IN transaction depends on how much data the device endpoint delivers. The maximum number of bytes in a packet a device can send is *Maximum Packet Length*. The number of bytes moved in a packet during an OUT transaction is either *Maximum Packet Length* or *Total Bytes to Transfer*, whichever is less. For isochronous transfers, the number of bytes moved during an OUT transaction is determined by the *Segment Length* in the segment list elements associated with the packets.

If there was a transaction error during the transaction, the transfer state (as defined above) is not advanced by the WHC. The WHC considers a transaction to be in error only when none of the data packets in a burst of packets were transferred successfully. For bulk, control and interrupt endpoints (except low power interrupt IN endpoints), the *Max Retry* field determines the number of retries to be performed. Transaction errors are summarized in Section 4.12.1.1. The WHC also increments the *Transaction Error Count* field by one when it detects a transaction error.

The following events will cause the WHC to clear the *Active* bit in the QHead's qTD/iTD overlay status field. When the *Active* bit transitions from a one to a zero, the transfer in the overlay is considered complete. The reason for the transfer completion (clearing the *Active* bit) determines the next state of the Queue Set.

- The number of transaction retries exceeds the value in the *Max Retry* field. When this occurs, the *Halted* bit is set to a one and the *Active* is set to a zero. The *Retry Count Exceeded* bit is also set. This results in the hardware not advancing the queue and the pipe halts. The WHCD must intercede to recover.
- The device responds to the transaction with a STALL handshake. When this occurs, the *Halted* bit is set to a one and the *Active* bit is set to a zero. This results in the hardware not advancing the queue and the pipe halts. The WHCD must intercede to recover.
- The *Total Bytes to Transfer* field is zero after the transaction completes. Note that for a zero length transaction, it was zero before the transaction was started. When this condition occurs, the *Active* bit is set to a zero. The WHC increments the *iCur* field by one or resets it to a zero if the current qTD is the last one in the Queue Set.
- The *Direction* bit is an IN and the device sent a packet with its *LastPacketFlag* bit set. When this occurs, the *Active* bit is set to zero. The WHC refers to the *iAlt* field to determine the next *iCur* value. If the *Valid* bit in the *iAlt* field is set to a one, the WHC updates the *iCur* field with the *Alt qTD Number* in the *iAlt* field. If the *Valid* bit in the *iAlt* field is set to a zero, the WHC does not update the *iCur* field. The WHCD must intercede to recover.

  Note that if both this condition and the previous one (*Total Bytes to Transfer* field is zero) are met, then the WHC must follow this rule.
- The *TR Type* is Isochronous and the *Num Segments* field is zero after the transaction completes. When this occurs, the *Active* bit is set to a zero. The WHC increments the *iCur* field by one or resets it to a zero if the current iTD is the last one in the Queue Set.
- The *Direction* bit is an IN and the device sends more than the expected number of bytes (i.e. *Maximum Packet Length* or *Total Bytes to Transfer* bytes, whichever is less) (i.e. a packet babble). This results in the WHC setting the *Babble Detected* bit to a one and the *Active* is set to a zero. If the *TR Type* is Bulk, Interrupt or Control, the WHC also sets the *Halted* bit to a one and the WHCD must intercede to recover. If the *TR Type* is Isochronous, the WHC increments the *iCur* field by one or resets it to a zero if the current iTD is the last one in the Queue Set.

The duration of this state depends on the time it takes to complete the transaction group.

4.7.1.2.1 Flow Control

An endpoint enters the flow control state when the device responds with one of the following handshakes:

- An ACK handshake in an OUT transaction with its *bvAckCode* field set to a zero.
- A NAK handshake in an OUT or IN transaction (including a status stage transaction of a control transfer)

Upon reception of a flow control response from any type of endpoint, the WHC sets the *Flow Control* bit to a one.

The WHC does not execute any transaction for this endpoint while the *Flow Control* bit is set to a one (see Section 4.7.1.1.1).

For periodic (isochronous or interrupt) endpoints, the WHC clears the flow control condition at the beginning of the next service interval (zone). The WHC resets the *Flow Control* bit when it first fetches a QHead in a zone. For periodic OUT endpoints, the WHC must perform a transaction with the burst size one in the first transaction after clearing the flow control condition or perform a transaction with a W_{DT}CTA requesting a handshake packet.

For asynchronous (control or bulk) endpoints, the WHC clears the flow control condition when it receives an Endpoints Ready Notification (*DN_EPRdy*) from the device.

4.7.1.2.2 Halting a QHead

A halted endpoint is defined only for transfer types of control, bulk and interrupt. The following events indicate that the endpoint has reached a condition where no more activity can occur without intervention by the WHCD:

- An endpoint returns a STALL handshake during a transaction, or
- A transaction had *Max Retry* consecutive error conditions, or
- A Packet Babble error occurs on the endpoint.

When any of these events occur (for a QHead), the WHC halts the QHead. To halt the QHead, the *Active* bit is set to a zero and the *Halted* bit is set to a one. There may be other error status bits that are set when a QHead is halted. The WHC always writes back the overlay area to the source qTD when the transfer is complete, regardless of the reason (normal completion, last packet or halt). The WHC will not advance the transfer state if a transaction results in a Halt condition (i.e. no updates necessary for *Total Bytes to Transfer*, *Page List Pointer*, *Current Window* and *Start Seq*).

When a QHead is halted, the *WUSB Error Interrupt* bit in the WUSBSTS register is set to a one. If the *WUSB Error Interrupt Enable* bit in the WUSBINTR register is set to a one, a hardware interrupt is generated.

If a QHead has been halted due to *Max Retry* consecutive transaction errors, the WHCD may request the WHC to retry the transfer from the current state (with some modification of static parameters) by simply reactivating the Queue Set (see Section 4.7.8).

Note that if a QHead that has been halted due to an error condition other than the *Max Retry* count being exceeded, then the WHCD must remove the Queue Set from the schedule, make any modifications it wants (e.g. zeroing out the QHead working space, changing any static parameters etc) and then add the Queue Set back to the schedule. See Section 4.6 for details on updating either the Periodic or Asynchronous schedules.

4.7.1.3 Write Back QHead Working Space

This state is entered from the *Execute Transactions* state. The WHC may write back the intermediate transfer status to the QHead working space in main memory after executing one or more transactions for the

QHead. A WHC implementation may cache all the information required as long as it wants to and does not necessarily have to write back this information. However, if the qTD/iTD overlay in a QHead is completed (i.e. when the *Active* bit is reset to zero) then the WHC must write back either:

- the *Total Bytes to Transfer* and *Status* fields to the source qTD or
- the *NumSegments*, *Presentation Time* and *Status* fields to the source iTD.

The WHC uses the *iCur* field to determine the target address of the qTD/iTD. The source data for the write-back is the first DWord of the transfer overlay in the QHead (see Figure 3-10 or Figure 3-11). This DWord is written back to the first DWord of the target qTD/iTD. This state is also referred to as qTD/iTD retirement.

If this is an IN transfer, then it is the responsibility of the WHC to perform a blank transaction, if necessary, to the endpoint targeted by this QHead. However, if the Queue Set is removed from the schedule before the WHC can perform the blank transaction, then it need not perform a blank transaction to the endpoint.

Note that the WHC must update DWords 5 and 6 in the QHead Working Space when it retires the qTD/iTD.

4.7.2 Page List Usage for Data Transfer

A qTD or an iTD uses a separate data structure, *Page List*, which contains an array of buffer pointers and is used to reference the data buffer for a transfer. This specification requires that the buffer associated with the transfer be *virtually contiguous*. This means: if the buffer spans more than one physical page, it must obey the following rules (Figure 4-6 illustrates an example):

- The first portion of the buffer must begin at some offset in a page and extend through the end of the page.
- The remaining buffer cannot be allocated in small chunks scattered around memory. For each 4K chunk beyond the first page, each buffer portion matches to a full 4K page. The final portion, which may only be large enough to occupy a portion of a page, must start at the beginning of the page and be contiguous within that page.

A Page List is a physically contiguous buffer containing an array of 64-bit buffer pointers and long enough to support a maximum transfer size of 1048575 bytes for qTD and 1048576 bytes for iTD.

For control, bulk and interrupt transfers, the WHC uses the *Page List Pointer* field to reference the buffer pointer which should be used to start the current transaction. For isochronous transfers, the WHC uses the *Page Index* field in the Segment List as an index value to determine which buffer pointer in the Page List should be used to transfer the segment. The WHC uses a different buffer pointer for each physical page of the buffer. This is always true, even if the buffer is physically contiguous.

The WHC must detect when the current transaction will span a page boundary and automatically move to the next buffer pointer in the Page List. The next pointer is reached by incrementing the *Page List Pointer* or *Page Index* and pulling the next page pointer from the list. The WHCD must ensure there are sufficient buffer pointers to move the amount of data specified in the *Total Bytes to Transfer* or the *Page Index* field.

In addition to using the Page List, a qTD or an iTD can reference the data buffer directly by setting the *Page List Pointer* field to the physical address of the buffer. This can be identified by setting the *Small Transfer (S)* bit to a one and only applicable when the transfer can be executed using a single physical buffer page, i.e. maximum transfer size is 4K. If small transfer is used for iTD, the *Page Index* in the Segment List will not be used.

Figure 4-6 illustrates a nominal example of how the WHCD would initialize the buffer page pointers for a transfer. For a transfer using multiple buffer pages, the WHCD would initialize the Page List and set the *Page List Pointer* field. The *Page List Pointer* points to the beginning of the Page List. The upper 52-bits of Page 0 references the start of the physical page and the lower 12-bits holds the offset in the page. The remaining page pointers are set to reference the beginning of each subsequent 4K pages. The *Small Transfer (S)* bit is set to a zero.

For a transfer using a single buffer page, the WHCD would set the *Page List Pointer* field to the physical address of the buffer. The *Small Transfer (S)* bit is set to a one.

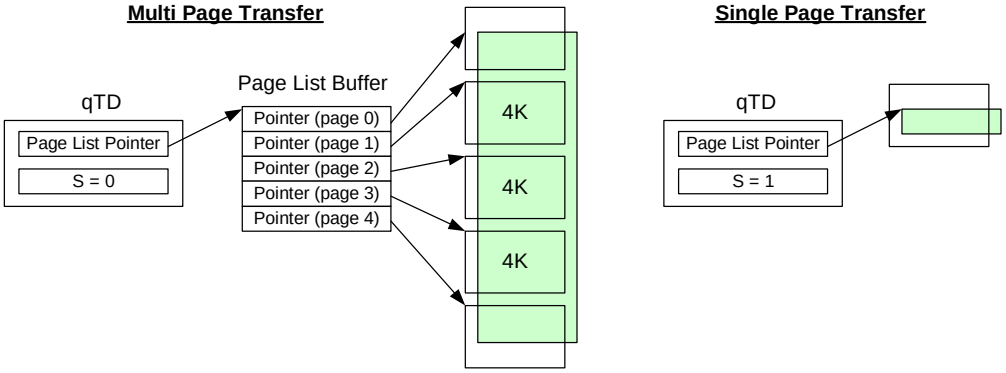


Figure 4-6. Example Mapping of Page List Pointer to Buffer Pages

4.7.3 Device Information

The WHCD uses a Device Information Buffer (see Figure 3-12) to indicate device specific information to the WHC. The Device Information Buffer is a 64-byte aligned physically contiguous buffer in system memory which contains an array of device information elements. Each device information element is 64 bytes in length. The total size of the Device Information Buffer is determined by the number of devices supported by the WHC which is declared by the *N_DEVICES* field in the WHCSPARAMS register. When the WHC performs a transaction using a QHead, it must refer to the device information element associated with this QHead, which is pointed to by the *Device Info Index* field in the QHead. Figure 4-7 presents how the WHC determines the physical address of the associated device information element from the *Device Info Index*.

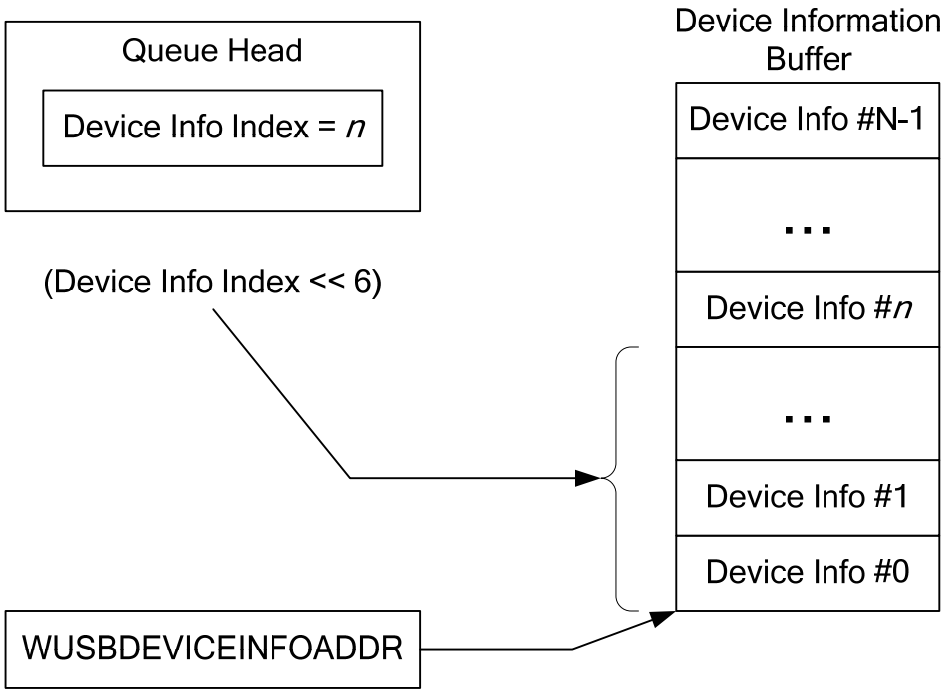


Figure 4-7. Association of QHead to Device Information

A device information element contains the following information

- *Device Address*. This field contains an 8-bit address assigned to this device. The WHC uses this value as the lower byte of the 16-bit *DestAddr* field in the MAC header of any data frame sent to this device.
- *Secure Frame Bit*. If this bit is set to a one, all transactions to/from this device must be executed using secure frames. The encryption key can be referred to by the *Key Table Index* field.
- *Key Table Index*. This field indicates the location containing a 128-bit AES encryption key and a Temporal Key Identifier (TKID) used to transmit/receive data to/from this device. This field is valid only when the *Secure Frame Bit* is set to a one.
- *Disabled*. If this bit is set to a one, the WHC does not execute any transaction to/from any endpoint in this device.
- *Device Availability*. This field is a 256-bit bitmap indicating whether each MAS in a superframe is available for this device or not. Each bit represents the availability of each MAS respectively. The least significant bit is associated with the first MAS in the superframe. If the bit is set to a one, the corresponding slot is available for this device and the WHC can execute transactions to/from this device. If this bit is a zero, the slot is not available for this device and the WHC must not execute transactions.

The WHC may cache a number of device information elements internally and reuse them without additional fetches for future transactions. For this reason, the WHCD must set the associated bit in the WUSBDIBUPDATED register when it has changed the content in the Device Information Buffer. When any bit in the WUSBDIBUPDATED register is set to a one, the WHC must update the internal cache of the corresponding device information and then reset the associated bit in the WUSBDIBUPDATED register to a zero. The WHCD must not modify the content in the Device Information Buffer while the associated bit in the WUSBDIBUPDATED register is a one. Note that writing of a zero to each bit in this register has no effect to its value and the WHCD can only set the bits associated with the modified device information elements to a one.

4.7.4 Managing Transfer Complete Interrupts from QHeads

The WHC will set an interrupt to be signaled when the completed transfer (qTD or iTD) has an *Interrupt on Complete (IOC)* bit set to a one, or whenever an IN transfer (qTD) completes by receiving a packet with its *Last Packet Flag (LP)* bit set. If the WHCD needs multiple qTDs or iTDs to complete a client request, the intermediate qTDs or iTDs do not require interrupts. The WHCD may only need a single interrupt to notify it that the complete buffer has been transferred, however the WHCD may set IOCs to occur more frequently. A motivation for this may be that it wants early notification so that interface data structures can be re-used in a timely manner.

4.7.5 Blank Transaction for IN Transfers

For transfers from bulk, interrupt and isochronous IN endpoints, the WHC must perform a blank transaction to convey the receive window to the endpoint. A blank transaction is executed when one of the following conditions is met:

- For bulk IN endpoints, the WHC must perform a blank transaction after completing a transfer and there is no active transfer following.
- For isochronous or interrupt IN endpoints, the WHC must perform a blank transaction as the last transaction in a service interval if it has received data packets from the endpoint in the previous transaction and advanced the receive window.

A blank transaction is performed by inserting a *Blank $W_{DT}CTA$* in an MMC packet. The *wStart* field in this $W_{DT}CTA$ is set to that of the next $W_X CTA$. Note that a WHC implementation may choose to write back the QHead working space (see Section 4.7.3) only after performing a blank transaction to the endpoint specified in the QHead.

4.7.6 Inactivate on Next Transaction

When the WHC detects the *Inactivate on Next Transaction (I)* bit set in an active QHead, it must inactivate the Queue Set by resetting the *Active* bit to a zero and setting the *Queue Set Inactivated* bit to a one. When the WHC inactivates the Queue Set, it must also write the status back to the source qTD/iTD and set the *WUSBINT* bit in the WUSBSTS register (regardless of the *IOC* bit). The WHC must not advance the *iCur* field when the Queue Set is inactivated.

The WHCD may reactivate the Queue Set by setting the *I-Bit* to a zero, setting the *Active* bit to a one in the qTD/iTD and setting the *Reactivate Queue Set* bit to a one (see Section 4.7.8). The WHCD may change some parameters in the QHead before reactivation, e.g. *Maximum Packet Length*. When the WHCD sets or resets the *I-Bit*, it must inform the WHC that the static field in a QHead has been changed via setting either the *Periodic Schedule Updated* or the *Asynchronous Schedule Updated* in the WUSBCMD register (see Section 4.6).

4.7.7 Modifying QHead Static Endpoint State

The WHCD is allowed to change certain fields in the QHead static endpoint state. If the QHead is active then the WHCD must de-activate the QHead before modifying any of the static fields. The WHCD can force the WHC to temporarily de-activate a QHead using the *I-Bit* (see Section 4.7.6). Once the WHCD has made the modifications, it can re-start the transfer with the new parameters as mentioned in the Section 4.7.8.

If the change is being made to the *Maximum Packet Length* field in the QHead for an IN transfer then it is necessary that the WHCD ensures that the remaining transfer length is an integral multiple of the new *Maximum Packet Length* except for the last qTD of the transfer. This allows the WHC to correctly handle packets that are received at the end of the current qTD.

NOTE: The WHCD can change the *Maximum Packet Length* field value in the QHead for Bulk and Interrupt transfer types only.

4.7.8 Reactivating a Queue Set

The WHCD can reactivate a previously inactivated/halted Queue Set by setting the *Active* bit in the qTD/iTD to a one and setting the *Reactivate Queue Set (rqs)* bit to a one in the QHead. The *Reactivate Queue Set* bit is only effective when the *Active* bit is a zero and either the *Halted* bit or the *Queue Set Inactivated* bit is set to a one in the QHead overlay. When this bit is set to a one in the above condition, the WHC will fetch the qTD/iTD pointed to by the *iCur* field. If the fetched qTD/iTD has its *Active* bit set to a one, then the WHC will overlay the first DWord of the qTD/iTD, but does not modify all the other fields in the QHead. The WHC will just resume the transfer from the current state, i.e. *Current Window*, *Start Seq*, *Page List Pointer* and all the intermediate information maintained in the Host Controller Scratch pad fields.

When the WHCD reactivates a Queue Set which has been inactivated as per the *I-Bit*, it will perform the following steps:

- Sets the *I-Bit* to a zero.
- Set the *Active* bit to a one and set the *Queue Set Inactivated* bit to a zero in the qTD *Status* field.
- Set the *Reactivate Queue Set* bit to a one.
- Set the *Asynchronous/Periodic Schedule Updated* bit to a one in the WUSBCMD register

On the other hand, the WHCD will perform the following steps when it reactivates a halted Queue Set due to the *Retry Count Exceeded*:

- Set the *Active* bit to a one, set the *Halted* bit to a zero and set the *Retry Count Exceeded* bit to a zero in the qTD *Status* field.
- Set the *Reactivate Queue Set* bit to a one.
- Set the *Asynchronous/Periodic Schedule Updated* bit to a one in the WUSBCMD register

Note that the WHC is responsible for reloading the retry count when a Queue Set which was halted due to the retry count being exceeded (i.e. *Retry Count Exceeded* bit set to a one in the qTD/iTD) is reactivated.

4.7.9 Control Transfers

A control transfer is always performed by using a single qTD. When the WHC fetches a new active qTD for the QHead with the transfer type (*TRType*) of Control, it performs the following operations:

- Initializes the *Current Window* field with the least significant *Burst* bits set to a one. The *Burst* is usually one and the *Current Window* would be set to 00000001h.
- Initializes the *Start Seq* field to a zero.

The WHC must first perform a setup stage transaction along with the data or status stage by setting the Setup flag in the WxCTA and including the setup data specified in the qTD. It should keep including it until it sees a valid response (data or handshake) from the endpoint.

If the *Total Bytes to Transfer* is greater than zero, the WHC must first perform data stage transactions. The direction of a control transfer is determined by the most significant bit (*GS* bit) of the first byte in the setup data. If the bit is set to a one, it is a control read (IN) transfer otherwise it is a control write (OUT) transfer. The WHC performs a status stage transaction after completing data stage transactions by transferring all the data and the *Total Bytes to Transfer* has reached zero or the transfer has been completed by receiving a data packet with its *LastPacketFlag* set.

If the *Total Bytes to Transfer* is zero then the WHC must perform a control transfer without data stage and only performs a status stage transaction. The *GS* bit in the first setup byte is ignored.

Upon successful completion of a status stage transaction, the WHC completes the transfer by resetting the *Active* bit to a zero and writes the status back to the source qTD.

4.7.10 Interrupt Transfers

A QHead for an interrupt endpoint is linked to the periodic zone list according to the poll rate (service interval).

The WHC can perform transactions to/from this endpoint up to the number specified by the *Max Count* field during each service interval (zone) only when it requires a retry of a transaction. For interrupt IN endpoints, the following rules for performing transactions apply:

- Once the WHC completes a transaction successfully, it must stop performing any more transactions for this endpoint after executing a blank transaction (See Section 4.7.5).
- If the WHC detects some data but does not receive it successfully due to some error, e.g. bad FCS, it must perform transaction retries up to the number specified in the *Max Count* field in this zone or the number of retries of this transaction equals the value in the *Max Retry* field.
- If the WHC detects no data from an interrupt IN endpoint during the transaction, it must not perform any more transactions for this endpoint during the current zone and increment its internal count of retries of this transaction.

If the WHC has performed the *Max Retry* number of consecutive retries (i.e. without a successful transaction to this endpoint in between), it must halt the Queue Set and complete the transfer with *Retry Count Exceeded*.

4.7.10.1 Low Power Interrupt

When the *TR Type* field is set to 111b (Low Power Interrupt) in the QHead, the WHC must perform transactions for a low power interrupt IN endpoint. The WHC's operation is different from that for normal interrupt IN endpoints (*TR Type* is 011b) in the following points:

- The WHC does not use the *Max Retry* field and never completes a transfer with *Retry Count Exceeded*.

- When the WHC receives a NAK handshake from the endpoint, it must generate a Device Alive Notification (*DN_Alive*), place it to the Device Notification Buffer pointed to by the WUSBDNTSBUFADDR register, increment the *Current Offset* field in the WUSBDNTSBUFADDR register by one, and set the *WUSBDNTSINT* bit in the WUSBSTS register to a one. If the *WUSB DNTS Interrupt Enable* bit in the WUSBINTR register is set to a one then the WHC must generate an interrupt.

4.7.11 Isochronous Transfers

The WHC uses the following data structures for isochronous transfers:

- QHead (*TRType* is Isochronous)
- Isochronous Queue Element Transfer Descriptor (iTD)
- Device Information
- Page List
- Segment List

Queue Sets are linked to the Periodic Zone List. If an isochronous Queue Set is linked, the Queue Set Link Pointer DWord must have its *iqs* bit set to a one in the previous Queue Set or in the periodic zone list. The device information associated with the Queue Set is referenced by the *Device Info Index* field in the QHead. One or more iTDs are associated with the QHead. The WHC fetches an active iTD pointed to by the *iCur* field and overlays it in the Queue Set's overlay portion.

An iTD has two buffer pointers: *Page List Pointer* and *Segment List Pointer*. These pointers point to the physical addresses of the buffers containing a *Page List* and a *Segment List* respectively. A Page List is an array of buffer pointers to each data buffer page (See Section 4.7.2). A Segment List is an array of segment information elements for the data segments transferred to/from the isochronous endpoint.

Figure 4-8 presents the relationship between these data structures.

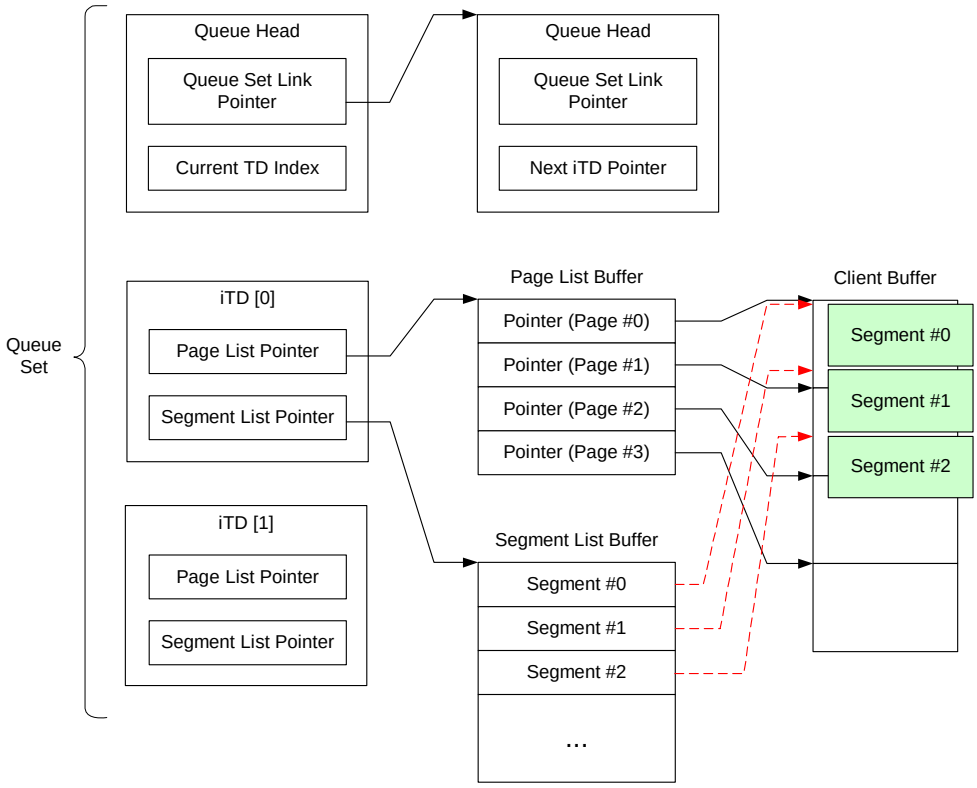


Figure 4-8. Example Data Structures for an Isochronous Transfer

A segment list element contains the *Page List Index* and the *Page Offset* used to determine the start address of the buffer location for the segment. The *Page List Index* is an index value to the Page List which contains the start address of the segment buffer. The *Page Offset* is the offset value to the page pointed to by the *Page List Index*. The segment list element also contains the *Segment Length*. This indicates the length of the segment to be transmitted for OUT transfers. For IN transfers, the WHCD specifies the expected segment length and the WHC writes back the actual length of the received segment.

If an Isochronous transfer uses a data buffer of a single page, the WHCD may not use a Page List, but sets the *Page List Pointer* field directly to the data buffer pointer. The *Small Transfer (S)* bit is set to a one. The *Page Index* field in the Segment List is not used in this case.

The maximum number of packets to be transferred during a zone is specified by the *Max Count* field in the QHead. After the WHC has attempted to transfer the *Max Count* number of packets to/from the endpoint it must not perform any transaction for this endpoint in the current zone.

4.7.11.1 WHC Operational Model for Isochronous OUT Transfers

In Isochronous OUT Transfers, the WHC fetches one or more segment data from the system memory to construct a Wireless USB Isochronous Packet. The WHC sets the *Presentation Time* field in the first packet of a transfer to the *Presentation Time* value specified by the WHCD.

The WHC determines the number of segments in a packet by accumulating the *Segment Length* (with 2-byte *wLength* field in the Wireless USB Isochronous Packet) along with the segment list. If the total length (including the 3-byte Isochronous Header) will exceed the *Maximum Packet Length* specified in the QHead by adding the next segment length, the WHC must not include that segment in the current packet.

The *Presentation Time* field in each packet can be determined by using the following parameters:

- Current *Presentation Time* value in the QHead overlay
- The offset of the first segment in the packet from the segment pointed to by the *Segment List Pointer* field in the QHead overlay
- The *Interval* value in the QHead.

The presentation time of the packet will be determined by adding the offset multiplied by the interval to the current *Presentation Time* value in the QHead overlay. Note that the actual interval value in 125 microsecond unit is $2^{Interval}$.

When the WHC receives a handshake packet from the endpoint indicating the successful arrival of previous data packets, the WHC must clear the *Active* bit in the segment list for all the data segments in the packets.

When the WHC detects an error in the transaction, it must increment the *Transaction Error Count* in the QHead (see Section 4.12.1.1).

Figure 4-9 illustrates an example sequence of isochronous OUT transactions. In this example, the endpoint has an interval value of 2. The *Presentation Time* field in the iTD is set to 100. The iTD has a pointer to the segment list which has information of 8 segments.

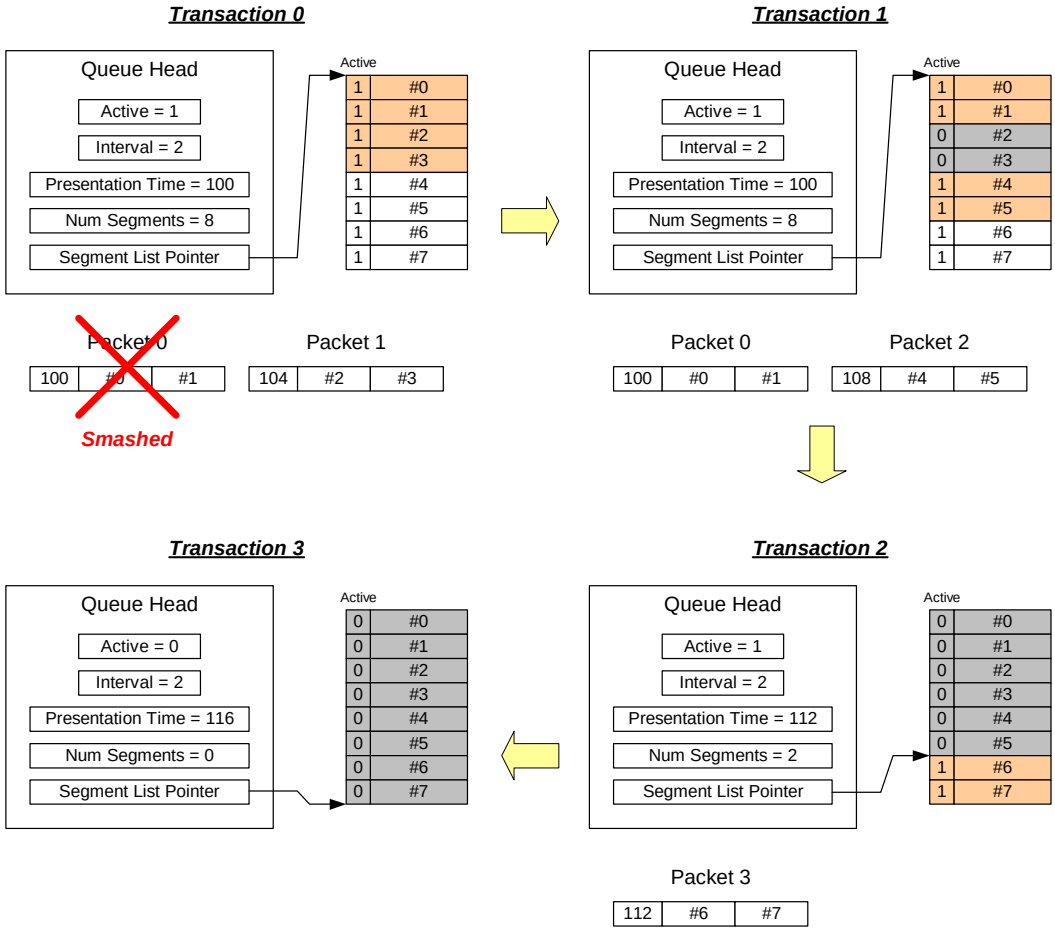


Figure 4-9. Example Isochronous OUT Transactions

In the first transaction (*Transaction 0*), the WHC performs a transaction with 2 packets. Both packets have 2 segments. The first packet has its presentation time set to 100, i.e. the *Presentation Time* field value in the QHead. The presentation time of the second packet is determined by adding the *Interval* (2) multiplied by the distance between the first segment in this packet (#2) and the first segment in the first packet (#0) to the *Presentation Time* in the QHead (100). In this example, the presentation time of packet (#1) would be 104 (2*2 +100). Suppose the first packet gets smashed and only the second packet was transferred in this transaction, the segment list will be updated by resetting the *Active* bits for the segment #2 and #3 to a zero. Since the first segment was not transferred successfully, all the other fields in the QHead are left unchanged such as *Segment List Pointer*, *Num Segments* and *Presentation Time*.

In the second transaction (*Transaction 1*), the WHC retransmits the packet 0 and also sends a new packet (Packet 2). Both of these packets are transferred successfully and the WHC sets the *Active* bit for the segments in these packets to a zero. The WHC updates the QHead by advancing the *Segment List Pointer* so that it points to the first segment with its *Active* bit set, i.e. segment #6. The *Num Segments* and *Presentation Time* are also updated accordingly.

In the third transaction (*Transaction 2*), the WHC just sends a remaining packet. After completion of this transfer, the WHC updates the segment list and the QHead. Since *Num Segments* has transitioned to a zero, the WHC completes the transfer by setting the *Active* bit in the QHead to a zero and writing the status back to the source iTD.

When the WHC detects that the *Presentation Time* value in the iTD is nearing expiration (by comparing it with the current Wireless USB Channel Time in the WUSBTIME register), it must discard one or more

packets whose presentation time will be outdated. The WHC must discard the whole packet(s), not just some segments in the packet. When the WHC decides to discard a packet, it will update the QHead as follows:

- Advance the *Current Window* by clearing the bit pointed to by the current *Start Seq* field and setting another bit next to the last sequence number in the current window.
- Advance the *Start Seq* value to the first bit set.

When the WHC schedules a new transaction to this endpoint, it must generate an Isochronous Packet Discard IE and include it in the MMC. When it generates an Isochronous Packet Discard IE, it determines the fields in the IE as follows:

- Set the *bFirstReceiveWindowPosition* field in the IE to the *Start Seq* value in the QHead.
- Set the *bmDeviceReceiveWindow* field in the IE to the *Current Window* value in the QHead.

This IE must also contain the number of discarded packets, the number of discarded segments and the identifier for this IE. It is the WHC's responsibility to maintain these parameters internally.

After successful reception of a handshake packet from the endpoint in the transaction, the WHC stops sending an Isochronous Packet Discard IE. Figure 4-10 illustrates an example of isochronous OUT transactions with a packet discard.

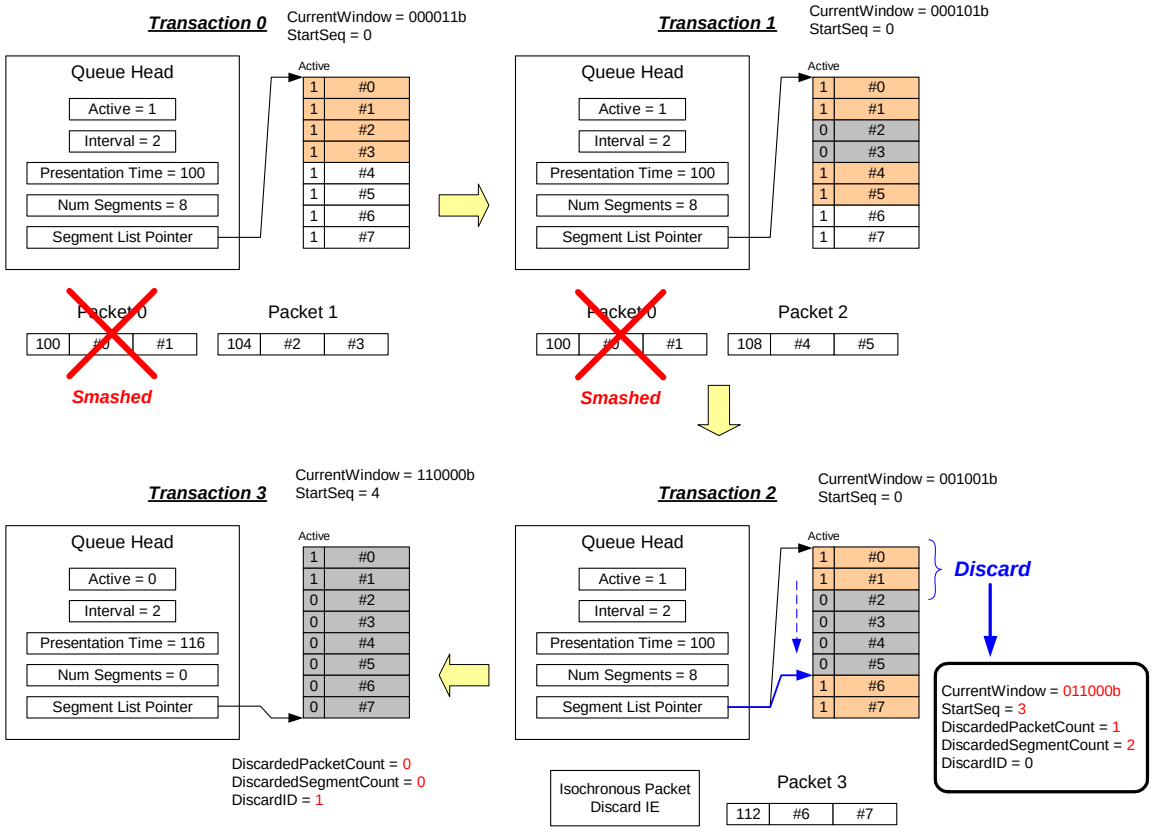


Figure 4-10. Example Isochronous OUT Packet Discard Scenario

When the WHC receives an ACK handshake with its *bvAckCode* set to a zero or a NAK handshake from the endpoint, it sets the *Flow Control* bit to a one and does not perform any more transactions to this endpoint during the current Zone. The WHC will restore from the flow control state at the next service interval (see Section 4.7.1.2.1).

4.7.11.2 WHC Operational Model for Isochronous IN Transfers

In Isochronous IN transfers, the WHC must place the data segments in a received packet to the appropriate location in the buffer associated with the presentation time of each segment.

When the WHCD sets an active iTD for the first time in a Queue Set, the WHC must determine the presentation time of the first segment in the iTD from the *Presentation Time* of the first packet received from the endpoint. Once the WHC receives a packet from the endpoint, it sets the *Presentation Time* value in the QHead's overlay area. The WHC must update the Segment List for all the segments in the received packet by resetting the *Active* bit to a zero and setting the *Segment Length* field to the actual length of the received segment.

Figure 4-11 illustrates an example of isochronous IN transactions. The presentation time in the QHead is invalid before executing the first transactions.

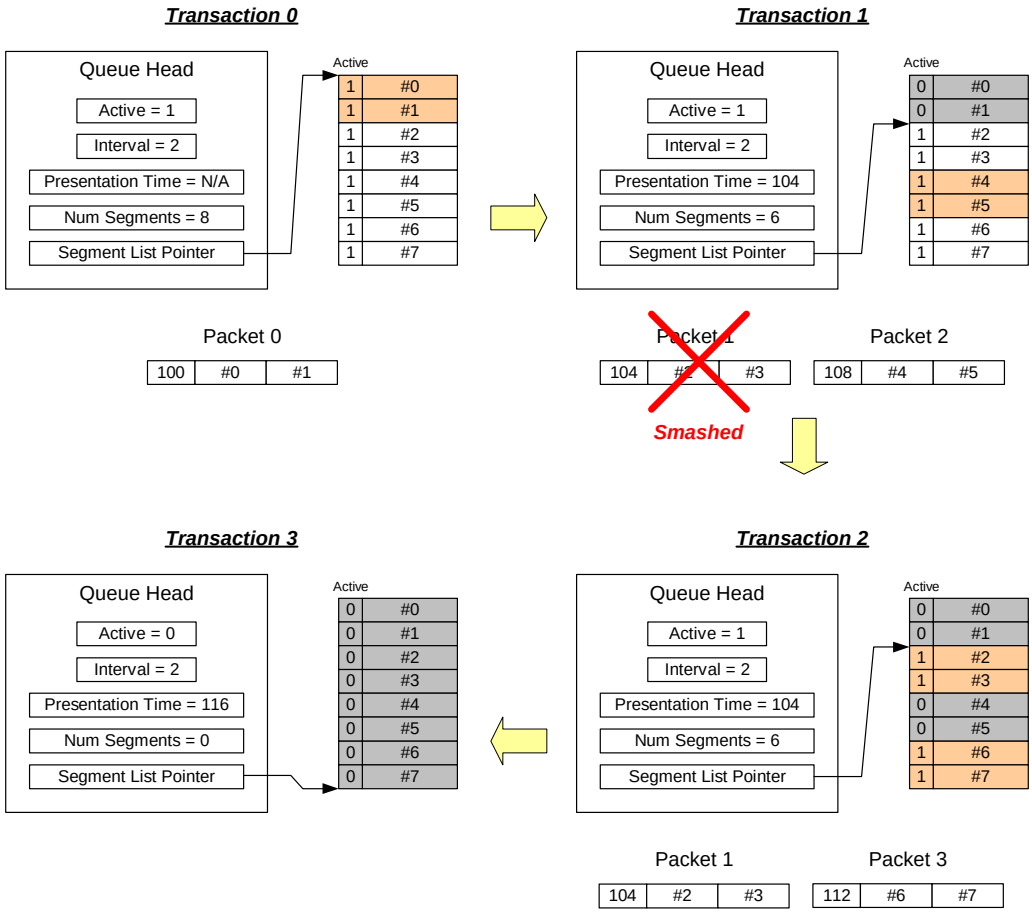


Figure 4-11. Example Isochronous IN Transactions

The WHC first performs a transaction with burst size one. After receiving the first packet from the endpoint, the WHC sets the *Presentation Time* field to the presentation time of the packet. Then, it updates the *Segment List Pointer* so that it points to the first segment list element with its *Active* bit set, i.e. segment #2. The WHC also updates the *Presentation Time* and the *Num Segments* field.

In the second transaction (*Transaction 1*), the WHC performs a transaction with burst size two and the endpoint sends two packets, however the first packet gets smashed. Upon reception of the second packet, the WHC determines the pointer to the segment list element associated with the first segment in the received packet using the following procedure:

- Get the distance between the *Presentation Time* in the QHead and the presentation time of the received packet. In this example, the *Presentation Time* in the QHead is 104 and the presentation time of the received packet is 108, hence the distance is 4.
- Divide the distance by the *Interval*. In this example, the *Interval* is 2 and the result would be 2.
- Add the above result multiplied by 8 (64-bit) to the current *Segment List Pointer*. This will be the pointer to the segment list element for the first segment in the received packet. Segment list elements for the subsequent segments are just following.

If the current Wireless USB Channel Time exceeds the current *Presentation Time* plus the *MaxStreamDelay*, the WHC must consider the current segment list element outdated and advance the *Segment List Pointer* to the segment list element whose presentation time is still valid. If all the remaining segment list elements are outdated, the WHC must retire the iTD by setting the *Active* bit to a zero even though it has not received all the packets. The *Active* bit in the segment list elements for those segments not received are left unchanged.

When all the segment list elements have become either inactive or outdated, the WHC will retire the current iTD and write the transfer status back to the source iTD. It then updates the *iCur* field in the QHead. If the WHC finds the next iTD is active, the presentation time for that iTD will be calculated using the presentation time in the overlay area. The presentation time of the first segment in the next transfer will be the presentation time of the last segment in the last transfer plus interval value.

If the WHC does not receive any packet for more than the *MaxStreamDelay* period after fetching the iTD first time, it must retire the iTD by resetting the *Active* bit. All the segment list elements are left unchanged.

If the WHC receives a data packet greater than the *Maximum Packet Length* in the QHead, it must discard the packet, set the *Babble Detected* bit and reset the *Active* bit in the QHead to a zero. It must also write the transfer status back to the source iTD and update the *iCur* field.

If the actual length of the received segment is greater than the expected segment length, i.e. the *Segment Length* value specified by the WHCD, the WHC must not place this segment in the buffer, but it must discard the segment, reset the *Active* bit to a zero and set the *Babble Detected* bit to a one in the Segment List. This error only affects the current segment, but the WHC will handle subsequent segments or packets as normal.

4.7.11.3 WHCD Operational Model for Isochronous Transfers

A client buffer request to an isochronous endpoint may contain 1 to N segments. When N is larger than one, the WHCD may have to use multiple iTDs to read or write data with the buffer.

The WHCD is responsible for setting the appropriate parameters in the QHead or the iTDs.

- The WHCD must set the *Interval* field in the QHead according to the *bInterval* value of the endpoint.
- For OUT transfers, the *Presentation Time* field must be set to the presentation time value associated with the first segment in the transfer. The presentation time is the Wireless USB channel time in 125 microseconds when the associated data will be consumed by the endpoint. The WHCD must determine an appropriate presentation time by using the Wireless USB channel time at some time in the future.
- For IN transfers, the *Max Stream Delay* field must be specified in the iTD based on the *wMaxStreamDelay* value of the endpoint.

4.8 Wireless USB Generic Commands

The WHC provides a generic command mechanism used by the WHCD to set different kinds of information to the WHC. The generic command is executed by using the WUSBGENCMDSTS, the WUSBGENCMDPARAMS register and optionally the WUSBGENADDR register. The WHCD writes type specific parameters in the WUSBGENCMDPARAMS register. It then writes to the WUSBGENCMDSTS register to set the *Command Type* and enables the command by setting the *Active* bit to a one. It may also use the WUSBGENADDR register to specify the buffer location containing any additional type specific information that the command may require.

When the *Active* bit in the WUSBGENCMDSTS register is set, the WHC reads the *Command Type* from this register and uses that to determine the type specific parameters in the WUSBGENCMDPARAMS register. If the specified type of command requires additional parameters in system memory, it fetches those parameters from the buffer pointed to by the WUSBGENADDR register. After completing the command, the WHC resets the *Active* bit in the WUSBGENCMDSTS register to a zero. The WHCD must not try to issue another command while the *Active* bit is a one. In addition, it must not clear the *Active* bit to a zero when it is set to a one. Unless otherwise noted in the description for each of the commands in the following sections, the WHC must reset the *Active* bit (and the command is deemed "complete") once it has parsed the command register, locally stored any parameters for that command as well as other any other information required to execute that command in the location pointed to by the WUSBGENADDR register.

The WHCD can require an interrupt from the WHC when it completes executing the command specified by setting the *Interrupt on Complete* bit in the WUSBGENCMDSTS register and setting the *Generic Command Completion Enable* bit in the WUSBINTR register.

4.8.1 Add/Remove MMC IE

The WHCD uses the *Add MMC IE* command to add or modify a group of WUSB Channel IEs, i.e. an IE block, included in MMC packets. The maximum size of each IE block is 255 bytes. A WHC must have internal storage for at least 3 IE blocks. The maximum number of IE blocks that the WHC can maintain at the same time is specified with the *N_MMC_IES* field in the WHCSPARAMS register. The IE block is identified by using an *IE_HANDLE*. The valid *IE_HANDLE* values are in the range of zero through *N_MMC_IES* minus one.

Each IE block has the following parameters as well as the content of the IEs

- Interval
- Repeat Count

The *Interval* determines how frequently these IEs are sent within MMC packets, expressed in milliseconds. The WHC keeps counting the time after sending a set of IEs before sending the same IE block again. The *Repeat Count* indicates the number of consecutive MMCs that must include these IEs. The WHC includes the same set of IEs in the consecutive MMCs in every opportunity determined by the specified *Interval* value. If the *Interval* is set to 0, the WHC must include the IEs in all the MMC packets until this IE block is removed. The *Repeat Count* filed is ignored if the *Interval* is set to 0.

If the *Interval* is set to 255, the WHC sends the IEs in *Repeat Count* number of consecutive MMCs and does not send them in any more MMCs after that. In this case, the WHC must reset the *Active* bit in the WUSBGENCMDSTS register only after sending the IEs the number of times specified in the *Repeat Count* field.

When the WHCD issues an *Add MMC IE* command, it first prepares a buffer that contains a set of IEs. Each IE data (including *bLength* and *IE Identifier*) must be located contiguously in the buffer. The WHCD writes the physical address of the buffer in the WUSBGENADDR register. It then writes to the WUSBGENCMDPARAMS register to set the *IE_HANDLE*, *Interval*, *Repeat Count* and *Size of IE Block*. Finally it sets the *Command Type* (set to 1) and enables the command by setting the *Active* bit to a one in the WUSBGENCMDSTS register.

A WHC must add the IE blocks that are to be sent in the MMC (based on the *Repeat Count* and *Interval* values specified when the IE block was added) after all the WxCTA IEs in ascending IE_HANDLE order. The WHCD must ensure that the IEs it adds using this command are ordered as per the *Wireless USB Specification Revision 1.0.*

If an *Add MMC IE* command is issued with the *IE_HANDLE* for an IE block currently sent in MMCs, the WHC must replace the current IE block with the newly specified IEs.

When the WHCD wants to remove an existing IE block, it uses the *Remove MMC IE* command. The WHCD specifies the IE block to be removed by setting the *IE_HANDLE* field in the WUSBGENCMDPARAMS register. The WHCD then writes the *Command* Type (set to 2) and enables the command (set the *Active* bit to 1) in the WUSBGENCMDSTS register. The WUSBGENADDR register is not used for this command.

If a *Remove MMC IE* command is issued with the *IE_HANDLE* for an invalid (non existant) IE block, the WHC will set the *Error* bit in the WUSBGENCMDSTS register and complete this command. Note that the WHC must store the IEs that are passed in via this command internally. The WHCD may reuse the memory where these IEs where initially placed for other purposes once the command is completed.

4.8.2 Set WUSB MAS

The WHCD uses the *Set WUSB MAS* command to update the MAS slot information reserved for the WUSB cluster. The WUSBGENADDR register points to the buffer that contains the 32 byte array of 256 entries. The WHCD issues this command by writing the WUSBGENCMDSTS register to set the *Command Type* (set to 3) and enables the command by setting the *Active* bit.

The WHC uses the specified information to determine the MAS slots that can be used to schedule WUSB transaction groups. If a bit is a one then the corresponding MAS slot is reserved for WUSB and the WHC can execute transactions during the slot. If the bit is a zero then the slot may be reserved by other devices and the WHC must not execute WUSB transactions during the slot.

4.8.3 Channel Stop

The *Channel Stop* command is used by the WHCD to stop the Wireless USB channel at the specified time. The WHCD issues this command by specifying the *Stop Time* in the WUSBGENCMDPARAMS register. It then sets the *Command Type* (set to 4) and enables the command by setting the *Active* bit in the WUSBGENCMDSTS register. The WHC must include a Channel Stop IE in every MMC it sends until the specified time. The *StopTime* field in the Channel Stop IE must be set to the *Stop Time* value specified in the command.

If the WHCD intends that the WHC poll for remote wake notifications, then it (the WHCD) must ensure that the Channel Stop information element's *bmAttribute.RemoteWake* field is set to one (1B) by setting the *Remote Wake* bit in the *Attributes* field. See Section 4.11 for details.

If the WHCD cancels a pending channel stop request, it issues a *Channel Stop* command with its *Cancel* bit set in the WUSBGENCMDPARAMS register. When the WHC receives this command, it stops including a Channel Stop IE in its MMCs and returns back to the normal operation.

If the command to cancel *Channel Stop* is received by the WHC after the channel has been stopped or the WHC did not receive a prior *Channel Stop* command, then the WHC must set the *Error* bit in the WUSBGENCMDSTS register and complete this command.

4.8.4 Remote Wake Poll Enable

The *Remote Wake Poll Enable* command is used by the WHCD to enable or disable the remote wake poll operation. The WHCD issues this command by writing the WUSBGENCMDSTS register to set the *Command Type* (set to 5) and set the *Active* bit to a one.

If the *Enabled* bit is set to a one in the WUSBGENCMDPARAMS register, the WHC will enable the remote wake poll. The *Remote Wake Poll Interval* field indicates the interval (number of WiMedia MAC superframes) at which the WHC will restart the Wireless USB channel to poll for remote wakeup

notifications. The *DNTS Window Width* field indicates the number of slots in the device notification time slots during the poll operation. See Section 4.11.2 for the detailed operational requirements.

If the *Enabled* bit is set to a zero, then the remote wake poll operation will be disabled.

4.9 Security

All WHC implementations must be able to encrypt outgoing data and decrypt incoming data by using the AES-128 Counter with CBC-MAC (CCM) cryptography. The WHC must maintain session keys, i.e.a pair-wise temporal key (PTK) for each device and a group temporal key (GTK), to be used for secure frame encapsulation of Wireless USB packets and MMCs respectively. A PTK is assigned for each device and used to transmit or receive packets to or from a device. The GTK is assigned for the entire Wireless USB cluster and used to transmit MMCs.

The maximum number of keys that can be maintained by the WHC at the same time is declared by the *N_KEYS* field in the WHCSPARAMS register. The WHC implementation must support at least one PTK per device and one GTK hence the number of keys that the WHC must support is determined by the number of devices supported by the WHC at the same time plus one for GTK. The installed key is referred to by using an index number of the key, i.e. *Key Table Index*. The valid *Key Table Index* value is in the range of 0 through *N_KEYS* – 1.

The WHC does not manage connection keys or any other keys.

4.9.1 Security Command

The WHCD performs security commands to install or remove 128-bit keys by using the WUSBSETSECKEYCMD register, the WUSBTKID register and the WUSBSECKEY register.

When the WHCD installs a new key, it executes a *Set Encryption Key* command by performing the following steps:

- Write the 24-bit Temporal Key Identifier (TKID) of the new encryption key to the WUSBTKID register.
- Write the 128-bit key to the WUSBSECKEY register.
- Write to the WUSBSETSECKEYCMD register to set the *Key Table Index*, indicate whether the installed key is a GTK or not via setting the *Group Key* bit and activate the command via setting the *Set Encryption Key Bit* to a one.

The valid *Key Table Index* value is in the range of zero through *N_KEYS* minus one. If the WHCD specifies the *Key Table Index* value greater than or equal to the *N_KEYS* value, the WHC's operation is undefined. The *Erase Encryption Key Bit* must be set to a zero when this command is performed. The WHC will acknowledge the completion of a command via resetting the *Set Encryption Key Bit* in the WUSBSETSECKEYCMD register to a zero.

If a *Set Encryption Key* command is requested with the *Key Table Index* for an existing key table entry, the WHC must replace the existing key with the new key.

When the WHCD removes an existing encryption key, it executes an *Erase Encryption Key* command by writing to the WUSBSETSECKEYCMD register to set the *Key Table Index* associated with the removed key and activate the command via setting the *Erase Encryption Key Bit* to a one. The *Set Encryption Key Bit* must be set to a zero when this command is performed. The WHC will acknowledge the completion of a command via resetting the *Erase Encryption Key Bit* in the WUSBSETSECKEYCMD register to a zero.

If an *Erase Encryption Key* command is requested with the *Key Table Index* for an invalid key table entry, then the WHC's operation is undefined.

4.9.2 PTK Management

The WHCD is responsible for all the connection and association process such as connection context management and 4-way handshakes. A 4-way handshake is used for mutual authentication with the device and to generate a PTK to be used for all the data transfer with the device.

When the WHCD performs a 4-way handshake, it must disable secure control transfers to/from the default endpoint of the device by setting the *Secure Frame Bit* in the Device Information to a zero. The WHCD must inactivate all the QHeads targeted to this device other than the default control endpoint while it performs a 4-way handshake.

After successful completion of a 4-way handshake, the WHCD must install the new PTK by using a *Set Encryption Key* command. The *Group Key* bit in the WUSBSETSECKEYCMD register must be set to a zero when it performs this command. It must also update the Device Information for this device to associate it with the created PTK via setting the *Key Table Index* field to the index number of the installed PTK and enable the secure communication with the device via setting the *Secure Frame Bit* to a one.

4.9.3 GTK Management

The WHCD must install a GTK before setting the MASs that the WHC may use to communicate with Wireless USB devices that are part of its cluster. The *Group Key* bit in the WUSBSETSECKEYCMD register must be set to a one when the WHCD performs a *Set Encryption Key* command. The WHC must transmit all the MMC packets by using a secure frame encapsulation with the installed GTK.

In addition, the WHCD must not erase the group key while the WHC is sending MMCs. Doing so will result in undefined behavior.

4.10 Device Notification

The WHCD enables device notification processing by setting the *Active* bit in the WUSBDNTSCTRL register. When enabling the DNTS (Device Notification Time Slot) schedule, the WHCD specifies following parameters to the WHC:

- Interval
- Number of Slots

The *Interval* field in the WUSBDNTSCTRL register indicates how frequently the WHC should schedule a DNTS. This field is expressed in milliseconds.

The *Number of Slots* field in the WUSBDNTSCTRL register is used as the *bNumslots* field in a W_{DNTS}CTA and is used to determine the total duration of the DNTS.

The WHC must set the *DNTS Schedule Status* field in the WUSBSTS register to a one when the DNTS schedule is enabled.

When a device notification arrives from a device during a DNTS, the WHC places it in the device notification buffer pointed to by the WUSBDNTSBUFADDR register. The device notification buffer must be page aligned and 4K in length. This buffer is divided into 64 notification blocks. Each device notification block is 64 bytes in length. Before enabling the DNTS schedule, the WHCD must set the WUSBDNTSBUFADDR register to the base address of the device notification buffer, i.e. the *Current Offset* field must be set to a zero. The WHC treats this buffer as a circular buffer. When it reaches the end of the 4K buffer it will start writing new notifications at the top of the device notification buffer if the WHCD has already completed processing of those notification blocks.

When the WHC places a received device notification in the buffer, it must perform the following steps:

- Set the *Source Address* field in the notification block to the lower 8 bits of the *SrcAddr* field in the MAC header of the received notification packet.
- Set the *Message Size* field in the notification block to the size of the device notification data (including the *bType*).

- If the device notification is received as a secure frame, set the *Secure Frame* bit to a one and writes the *TKID* value to the notification block. Otherwise, set the *Secure Frame* bit to a zero.
- Write the received device notification data (including the *bType*) to the notification block.
- Validate the notification block via setting the *Valid* bit to a one.

After successful transfer of the notification block, the WHC increments the *Current Offset* field by one so that the WUSBDNTSBUFADDR register will point to the buffer location where the next device notification will be placed.

After the DNTS period ends, the WHC sets the *WUSBDNTSINT* bit in the WUSBSTS register if at least one device notification was placed in the DNTS buffer during the DNTS. If the *WUSB DNTS Interrupt Enable* bit is set in the WUSBINTR register, the WHC will generate a hardware interrupt. When the *WUSBDNTSINT* interrupt is signaled, the WHCD will process one or more valid device notification messages and clear the *Valid* bit to a zero.

If the *Valid* bit is set to a one in the device notification block pointed to by the WUSBDNTSBUFADDR register when a device notification is received, the WHC will discard it, disable the DNTS schedule by clearing the *Active* bit in the WUSBDNTSCTRL register, reset the *WUSB DNTS Schedule Status* bit to a zero in the WUSBSTS register and then sets the *Device Notification Buffer Overflow* bit in the WUSBSTS register. If the *Device Notification Buffer Overflow Enable* bit is set in the WUSBINTR register, the WHC must generate a hardware interrupt.

Note that the WHC is responsible for processing all DN_EPRdy notifications it receives from devices connected to it (see Section 4.7.1.2.1).

4.11 Power Management

PC platforms may use multiple power supplies in support of some system power states. The power supplies are not redundant (i.e. not used concurrently) but are activated based on the system power state. Typically, S0 (full on) is powered by an inexpensive, inefficient power supply. S3, S4 may result in a switch to an Auxiliary power supply, which is a much lower power capable, but efficient power supply. The Auxiliary power supply cannot supply power to the entire platform, so many of the components provide support for multiple power ‘wells’, usually a ‘core’ and ‘aux’ well. When the main power supply is on, it powers everything in the core and aux wells and the Auxiliary power supply is off. If the Auxiliary power supply is on, then the main power supply is off, but only logic in the ‘aux’ well has power.

Implementations that conform to this specification must implement an ‘aux’ well in order to support remote wake events/communications from either WiMedia or Wireless USB devices when the system power state has been set to S3 or S4. Details on the logic required to be included in the ‘aux’ well is described below. Implementations are required to support the PCI Power Management interface, see Appendix A.

System software utilizes the PCI Power Management Interface (described in Appendix A) for managing the power states of a PCI/PCIe UMC implementation. This section covers the operational requirements of a UMC implementation to deliver the expected behavior when placed in device state other than D0. It is organized two sub-sections. The first details operational requirements of system software for transitioning the device out of the D0 state. The second details operational requirements for supporting Wireless USB Power Management and WiMedia Hibernation mode (and how these modes must interact).

4.11.1 Transitioning To a Dx State

System software must idle (stop) the PCI function before it transitions the function out of the D0 state. Idling is required in order to inhibit the UMC from attempting to perform bus master DMA requests to system memory. There may be several implications for system software (depending on software implementation) that must be taken into consideration in order to correctly idle the PCI function. The reference diagram in Figure 4-12 illustrates an example software implementation that uses separate device drivers for each sub-function interface the PCI function. In this implementation, all sub-function drivers are required to idle associated sub-function before the driver that owns the device function (URC/USB Bus Driver in this example) is allowed to change the D-state of the PCI device function. In this implementation,

the sub-function drivers must be aware of the device function driver in order to communicate synchronization and/or state information, so that the function driver (UWB Bus Driver/URC Driver) has the information about when all sub-functions have been idled and it can safely transition the device function out of the D0 state. Note that the process for gracefully stopping the WHC sub-function is described in Section 4.8.3.

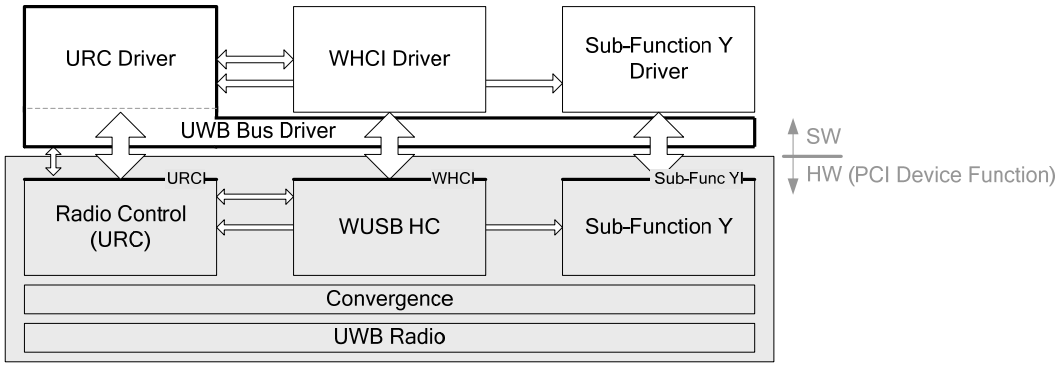


Figure 4-12. UMC Reference Block Diagram

4.11.2 WHC Power Management

A host system can manage host power in two general ways. The first is during normal operation by reducing power consumption by unused resources/components/subsystems during idle periods of the TDMA protocol. The power consumption reduction mechanisms and opportunities in this area are implementation-specific and beyond the scope of this specification.

The second way is for system software to idle the device function and then explicitly place the PCI device function into a higher D-state (i.e. D1 to D3). As mentioned in section 4.11.1 all interfaces (sub-functions) must be idled before the device function D-state should be modified. The WHC sub-function is stopped by simply setting the Run/Stop bit in the WUSBCMD register to a zero. A run/stop control bit value of zero has several implications:

- The WHC sub-function will not perform any bus master accesses (read or writes) to any of its memory based interface data structures.
- The WHC will not continuously maintain the associated Wireless USB channel.
- If the WHC has been enabled for remote wake polling, it will periodically restart the Wireless USB channel and provide device notification time slots within the channel for devices to transmit remote wake notifications. See below for additional operational requirements.

It is recommended that the WHCD always gracefully shut down the Wireless USB channel whenever it intends to idle the WHC sub-function. A graceful shutdown of the channel is described in Section 4.8.3 and involves the use of the Channel Stop IE. The graceful shutdown provides devices in the Wireless USB Cluster the opportunity to observe the intent of the host. The Channel Stop IE allows them to avoid performing error recovery procedures (to uselessly attempt to re-acquire the Wireless USB channel). If the WHCD intends that the WHC poll for remote wake notifications, then it (the WHCD) must ensures that the Channel Stop information element's *bmAttribute.RemoteWake* field is set to one (1B).

The commands 'Channel Stop' and 'Remote Wake Poll Enable' intentionally do not have any required or implied side effects. For example, stopping the channel with the remote wake attribute set to a one does not enable the WHC function for remote wake. It only ensures that the Channel Stop IE transmitted during the channel stop process has a 1B value in the *bmAttribute.RemoteWake* field. The WHCD must use the separate Remote Wake Poll Enable command to enable the WHC for remote wake polling. Therefore, the WHCD must use the appropriate form of commands to ensure devices in the Wireless USB cluster have proper notification that the channel is being shut down and whether (or not) the host will subsequently be periodically polling for remote wake. There is no order implied or required between the Channel Stop and

the Remote Wake Poll Enable commands. The only rule is that to properly prepare the WHC and the associated Wireless USB Cluster for remote wake polling, both of these commands must be completed before the WHCD shuts off the WHC by writing a zero (0B) to the run/stop bit in the WUSBCMD register.

The WHC sub-function must be running and maintaining a valid Wireless USB channel (run/stop bit in the WUSBCMD register has a value of 1B) before the WHCD can use the WUSBGENCMDSTS and WUSBGENCMDPARAMS registers to enable the WHC for remote wake polling.

A WHC that has been enabled for remote wake polling must not attempt to poll for remote wake until the WHC interface has been shut off (run/stop bit in the WUSBCMD register has a value of zero (0B)). Each MMC must contain (at least):

- A WCTA_IE with a W_{DNTS}CTA. The number of notification slots provided in each transaction group during the polling sequence must be at least the value specified in the Remote Wake Poll Enable command.
- Channel Stop IE. When the WHC is simply polling for remote wake notifications (via the WCTA_IE) it will simultaneously be announcing another shut down of the Wireless USB channel with this information element. The shutdown time must coincide with the completion of the last transaction group time segment.
- A Host Information IE. This information element must contain the same Host information (host name, or CHID) as was used to identify the channel before shutting down active operation. The WHCD must provide this information element. The Host Information IE block to be used with remote wake polling must be associated by the WHCD with MMC IE block zero (0). This is the ONLY IE that is valid for the WHCD to set prior to idling the WHC with remote wake polling enabled.

The WHC must transmit at least three MMCs at the poll interval specified in the Remote Wake Poll Enable command. The WHC must maintain the remote wake polling interval without intervention from the WHCD. This allows the WHC to conduct remote wake polling even if the device function has been transitioned out of the D0 state. Examples of remote wake polls by a Wireless USB host are illustrated in Figure 4-13. In these examples, the poll rate interval is 10 superframes. As illustrated every 10 superframes, the WHC restarts the Wireless USB channel for a period of time to complete at least three transaction groups, which are annotated with the appropriate Host Information IE and provide DNTS windows for devices to transmit their notifications.

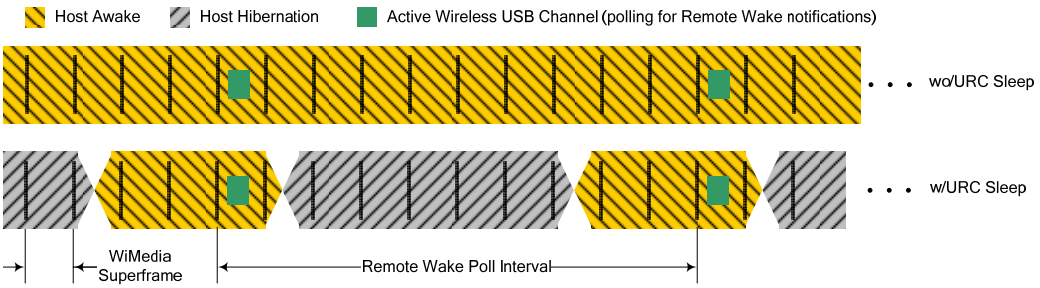


Figure 4-13. Examples of Remote Wake Polling Interval

The WHC sub-function works in conjunction with the URC sub-function to ensure that the WiMedia MAC channel time used for the Wireless USB channel is reasonably accounted for in the WiMedia distributed reservation framework. This requirement applies to whether the sub-function interfaces are active or idled. As such, Figure 4-13 illustrates two general scenarios on how the remote wake polling works within the WiMedia distributed reservation framework. Note that these scenarios are for reference only. Other operating options are valid and allowed. A summary of the valid combinations of interface state and the required behavior are listed in Table 4-3. The URC and WHC sub-function interfaces have valid combined states of operation, as listed below. The ON/OFF indicators correlate to the value of the run/stop command bit in the associated sub-function interface command register.

URC Interface States			WHC Interface States		
ON		Normal operation	ON		Normal operation
OFF	Sleep enabled	URC will cycle in/out of hibernation mode until external event (or system software cancellation) causes transition out of Sleep mode. Other sub-functions can have functionality within the hibernation cycle.	OFF	Remote wake poll enabled	No system bus master DMA operations allowed. WHC sub-function restarts the Wireless USB channel at a specified interval and provides DNTS time during which cluster members may transmit remote-wake candidate notification messages. [1]
	Sleep not enabled	No radio operation is permitted on URC or from any other sub-function in this mode.		Remote wake poll not enabled	No system bus master DMA operations allowed. WHC sub-function must not initiate any radio activity.

1. A valid implementation option is for the URC to ignore the hibernation parameters and associated functionality when the WHC has been enabled for remote wake polling. This means the URC will continue to maintain the beacon protocol with an appropriate DRP to allow the WHC to poll for remote wake notifications.

Table 4-3. Combinations of Sub-function Active vs Remote Wake Poll Enabled

URC run/stop [sleep state]	WHC run/stop [remote wake poll enabled]	Explanation
ON	ON	Normal active operational state
ON OFF [Sleep enabled] [3] OFF [Sleep not enabled]	OFF [remote wake poll not enabled]	WHC will not poll for remote wake. WHC logic can be completely powered down.
ON [2] or OFF [Sleep enabled]	OFF [remote wake poll enabled]	WHC will poll for remote wake. With the URC 'on' system software sets establishes the DRP reservation for the Wireless USB channel and the remote wake poll operation. [1]
OFF [Sleep not enabled] OFF [Sleep not enabled] OFF [Sleep enabled]	ON OFF [remote wake enabled] ON	ERROR : System software should not allow these combinations of state. Failure to do so will result in undefined behavior.

1. The software algorithm or policy for establishing and maintaining the reservation for the Wireless USB channel in this operational mode is beyond the scope of this specification.

2. The WHC will not conduct the remote wake poll unless system software has established a DRP reservation for the Wireless USB channel prior to the remote-wake poll deadline.

3. Sleep is the nomenclature in the URC for putting the host into cyclic hibernation mode.

System software must ensure the URC sleep parameters are consistent with the WHC *remote wake poll interval* parameters. The intent of this requirement is that the implementation should not have to 'check' these parameters for consistency and possibly modify and/or signal an error to the host in the case of conflicts. Inconsistent parameters will result in undefined behavior.

The URC and WHC sub-functions are required to share and modify some cross-sub-function internal state. First of all, the URC needs to know whether the WHC has been enabled for remote wake polling. This information is required by the URC because it dictates the operations the URC must do as it cycles out of hibernation near a remote wake poll interval deadline. Specifically, once the URC sub-function has re-joined the beacon group, it must establish a small DRP reservation for the Wireless USB channel. The URC must then be able to write the WUSB MAS bit vector with a value reflecting the DRP it has allocated for the purpose. All this needs to be completed prior to the WHC remote wake poll interval deadline. The second line of remote wake polling in Figure 4-13 illustrates the intended operation.

A WHC must only retain and react to valid remote wake notifications. Table 4-4 summarizes. A valid notification message must be encrypted using the PTK of the device. A WHC must not initiate a remote wake on the platform (assert an interrupt from D0 or assert PME# if in D3) unless it receives a valid notification during a remote wake poll DNTS. For example, the host will not assert and interrupt (or PME#) unless the notification received is encrypted and a notification type defined in Table 4-4.

Table 4-4. Valid Device Notifications for Remote Wake

Remote Wake Device Notification	Explanation
DN_RemoteWakeup	This is the only device notification that can be used by a Wireless USB device during a remote wake poll.

The WHC is not required to keep the WUSB channel alive after receiving a valid notification. Rather, it is simply required to retain the notification (at least one) and assert PME# or an interrupt (which ever is appropriate for the device state of the device function). As the system returns to operational mode, the WHC device driver will reprogram the WHC's operational registers and restart the WHC by setting the run/stop bit to a one (1B). At this point, bus master DMAs are re-enabled and the WHC can then write the saved notification packet to the memory-based notification buffer and signal an interrupt (if enabled). From this, the WHCD can determine which device initiated the interrupt and why.

4.11.3 WHC Resource Requirements for Remote Wake Polling

In order to support remote wake polling operations, the implementation must retain information and functional units in the auxiliary power well (Aux Well). Table 4-5 summarizes the items which need to be retained in the Aux power well. Note that these items can be reset on a D3hot to D0 reset, unless specifically noted to be exempt.

Table 4-5. WHC Resources for the Auxiliary Power Well

Item	Description
Host Information IE (MMC IE Block zero)	Must be set up by the WHCD prior to idling the WHC interface when remote wake polling has been enabled.
Radio Subsystem	The PHY and Baseband must be in the Aux power well.
Security Key Store	This information is used to decrypt and authenticate notification packets. This information must not be reset on D3hot to D0 reset.

Table 4-5. WHC Resources for the Auxiliary Power Well (cont.)

Item	Description
WUSB Microscheduler	A sufficient amount of the WHC sub-function must be in the Aux power well to be able to correctly construct and transmit the remote wake polling MMCs.
Notification Buffer	The WHC must have storage for at least one notification packet. This buffer must not be reset on D3hot to D0 reset.
Remote Wake Polling Parameters	These parameters must be retained internally to drive correct and timely polling for remote wake notifications.
WUSBCMD Register	The Wireless USB channel's stream index and broadcast cluster ID must be in the Aux power well so that they are available for the MMC packet transmissions that poll for remote wake.
WUSB Channel MAS Information	This is WHC internal data structure manipulated by the WHCD via the Generic command *Set WUSB MAS*. This structure identified the explicit media access slots available for use by the WHC for the Wireless USB channel.
Wireless USB Channel Time Counter	This logic must be available for correct operation during remote wake polling. There is no requirement that the value be consistent between remote wake poll periods.

4.11.4 URC Resource Requirements for Sleep Mode

In order to support Sleep mode operations, the implementation must retain information and functional units in the auxiliary power well (Aux Well). Table 4-6 summarizes the items which need to be retained in the Aux power well. Note that these items can be reset on a D3hot to D0 reset, unless specifically noted to be exempt.

Table 4-6. URC Resources for the Auxiliary Power Well to support Sleep Mode

Item	Description
Superframe Time Counter	Note that this is allowed to be a derivation of the Wireless USB Channel Time Counter (see Table 4-5).
Device Address	This is the device address used by the URC in beacon transmissions between hibernation periods.
MAC Address	This is the MAC address used by the URC in beacon transmissions between hibernation periods.
Notification Filter	This filter mask is used by the URC to decide which events detected by the URC should result in waking the system (asserting PME# or an interrupt) while in Sleep mode.
Event Buffer	The URC must have storage for at least one event (notification). This buffer must not be reset on D3hot to D0 reset.
Beacon Frame Buffer	This is storage, etc. used for maintaining the Beacon information elements set by the URCD. The URCD must ensure that a minimalist set of appropriate (and required) IEs have been set up prior to entry to sleep mode.
Transmit Power	The URC must use the same transmit power setting established by the URCD prior to entry into sleep mode. The URC must retain the last transmit power setting value and use it while in sleep mode.
Sleep Mode Operating Parameters	These parameters must be retained internally in order to correctly manage the Sleep mode hibernation cycles as per the parameters set by the URCD.

4.12 WHC Interrupts

The WHC hardware provides interrupt capabilities based on a number of sources. There are several general groups of interrupt sources:

- Interrupt as a result of executing transactions from the schedule (success or error conditions),
- WHC events and
- WHC error events

All transaction-based sources are maskable through the WUSBINTR register (see Section 2.4.5). Additionally, individual transfer descriptors can be marked to generate an interrupt on completion. This section describes each interrupt source and the processing that occurs in response to the interrupt.

Initial interrupt processing is the same, regardless of the reason for the interrupt. When an interrupt is signaled by the hardware, CPU control is transferred to the WHCD's Wireless USB interrupt handler. The precise mechanism to accomplish the transfer is OS specific. For this discussion it is just assumed that control is received. When the interrupt handler receives a control, its first action is to read the WUSBSTS (WUSB Status) register (see Section 2.4.4). It then acknowledges the interrupt by clearing all the interrupt status bits by writing ones to these bits positions. The handler then determines whether the interrupt is due to schedule processing or some other event. After acknowledging the interrupt, the handler (via an OS-specific mechanism) schedules a deferred procedure call (DPC) which will execute later. The DPC routine processes the results of the schedule execution. The precise mechanisms used are beyond the scope of this document.

Note that the WHC is not required to de-assert a currently active interrupt condition when the WHCD sets the interrupt enables (in the WUSBINTR register) to a zero. The only reliable method the WHCD should use for acknowledging an interrupt is by transitioning the appropriate status bits in the WUSBSTS register from a one to a zero.

4.12.1 Transfer/Transaction Based Interrupt

These interrupt sources are associated with transfer and transaction progress.

4.12.1.1 Transaction Error

A transaction error is any error that caused the WHC to think that the transfer did not complete successfully. Table 4-1 lists the events/responses that the host can observe as a result of a transaction. The effects of the error counter and interrupt status are summarized in the following paragraphs.

Table 4-7. Summary of Transaction Errors

Event	QHead Side-effects			Status Register (WUSBSTS)
	Num Retries >= Max Retry[1]	XactErrCount	Status Field	WUSBERRINT
FCS	TRUE	+1	RetryCountExceeded is set to 1[1]	1
Timeout				
Bad PID				
Error ACK[2]				
FCS	FALSE	+1	N/A	N/A
Timeout				
Bad PIC				
Error ACK[2]				
Babble	N/A	N/A	See Section 4.12.1.1.1	1

Table 4-7. Summary of Transaction Errors (cont.)

Event	QHead Side-effects			Status Register (WUSBSTS)
	Num Retries >= Max Retry[1]	XactErrCount	Status Field	WUSBERRINT
Buffer Error	TRUE	+1	See Section 4.12.1.1.2	1
	FALSE	+1		N/A

[1]This number of consecutive retries of a transaction is not applicable to Isochronous and low power interrupt endpoints. A transaction is counted as a retry if all the packets in the previous transaction to this endpoint were not transferred successfully.

[2]An ACK handshake for an OUT transaction with a *bvAckCode* containing all the bits corresponding with the packets sent in this transaction set to one, i.e. no packet was received by the endpoint successfully.

The WHC is responsible for counting the number of consecutive retries per transaction for a particular endpoint. The number of retries must be reset to zero whenever a successful transaction is performed to that endpoint. The transaction error count (XactErrCount) on the other hand is a running total of all the transaction errors that occurred while performing transactions to this endpoint. It is set to Zero by the WHCD. The WHCD can reset this field only when the Queue Set is inactive.

4.12.1.1.1 Babble

When a device transmits more data on the Wireless USB Channel than the WHC is expecting for this transaction, it is defined to be babbling. In general, this is called a *Packet Babble*. When a device sends more data than the *Maximum Length* number of bytes, the WHC sets the *Babble Detected* bit to a one and resets the *Active* bit in the QHead. The WHC also halts the endpoint in the case of a Control, Bulk or Interrupt endpoint by setting the *Halted* bit in the QHead. The WHC must write the status back to the source qTD/iTD. The *Maximum Length* is defined as the minimum of the *Total Bytes to Transfer* and the *Maximum Packet Length* for a Control, Bulk or Interrupt transfer. For Isochronous endpoints, the *Maximum Length* is always equal to the *Maximum Packet Length* in the QHead.

When this error occurs the *WUSBERRINT* bit in the WUSBSTS register is set to a one and if the *WUSB Error Interrupt Enable* bit in the WUSBINTR register is set to a one, then a hardware interrupt is signaled to the system.

For Isochronous endpoints, a babble condition also occurs when a device transmits a packet including a data segment greater than the WHC is expecting. The expected segment length is determined by the *Segment Length* field specified by the WHCD in the segment list element associated with the received segment. If this babble condition occurs, the WHC only discards this segment, sets the *Babble Detected* bit in the segment list to a one and resets the *Active* bit in the segment list to a zero. The WHC does not discard other segments. The *Active* bit in the QHead is left unchanged.

Note that a Babble error can only occur on an IN transaction. A WHC never sends more data than specified (*Maximum Length* and/or *Segment Length*) and it never starts an OUT transaction that will exceed the allocated channel time.

4.12.1.1.2 Data Buffer Error

This event indicates that an overrun of incoming data or an underrun of outgoing data has occurred for this transaction. This would generally be caused by the WHC not being able to access required data buffers in memory within necessary latency requirements. These conditions are considered transaction errors. When these errors do occur, the WHC records the fact the error occurred by setting the *Data Buffer Error* bit in the QHead.

If the data buffer error occurs on an IN transaction, the WHC will not acknowledge the packets that were not transferred to main memory due to the data buffer error. This will force the endpoint to resend the packets in response to the next IN to the endpoint.

If the data buffer error occurs on an OUT transaction, the WHC must corrupt the end of the packet so that it cannot be interpreted by the device as a good data packet. Simply truncating the packet is not considered acceptable.

4.12.1.2 WUSB Interrupt (Interrupt on Completion (IOC))

Transfer Descriptors (qTDs or iTDs) contain a bit that can be set to cause an interrupt on their completion. The completion of the transfer associated with the schedule item causes the *WUSB Interrupt (WUSBINT)* bit in the WUSBSTS register to be set to a one. If the *WUSB Interrupt Enable* bit in the WUSBINTR register is set to a one, a hardware interrupt is signaled to the system. If the completion is because of errors, the *WUSBERRINT* bit in the WUSBSTS register is also set to a one.

4.12.1.3 Last Packet

Reception of a data packet that has its *LastPacketFlag* set in the WUSB header during Control, Bulk or Interrupt transfers signals the completion of the transfer. The WHC completes the qTD when it has received all the packets before the packet with its *LastPacketFlag* set. Whenever a last packet completion occurs during a QHead execution, the *WUSBINT* bit in the WUSBSTS register is set to a one regardless of the *IOC* bit value. If the *WUSB Interrupt Enable* bit is set in the WUSBINTR register, a hardware interrupt is signaled to the system.

4.12.1.4 WUSB DNTS Interrupt

The WHC must set the *WUSBDNTSINT* bit in the WUSBSTS register to a one if it receives one or more Device Notifications from devices during a scheduled Device Notification Time Slot. It places these notifications in the Device Notification Buffer pointed to by the WUSBDNTSBUFADDR register. If the *WUSB DNTS Interrupt Enable* bit is set in the WUSBINTR register, a hardware interrupt is signaled to the system.

4.12.1.5 Device Notification Buffer Overflow

If there is no space available in the device notification buffer, the WHC must set the *Device Notification Buffer Overflow* bit in the WUSBSTS register. If the *Device Notification Buffer Overflow Enable* bit is set in the WUSBINTR register, a hardware interrupt is signaled to the system.

See Section 4.10 for other requirements of a WHC when this condition occurs.

4.12.2 WHC Event Interrupts

4.12.2.1 Interrupt on Periodic Schedule Synched

This event is used by the WHCD to determine that the WHC has synchronized its internal cache of the periodic schedule with the schedule on the system memory. If the *Interrupt on Periodic Schedule Synched Enable* bit in the WUSBINTR register is a one, the WHC must issue a hardware interrupt. A detailed explanation of this feature is described in Section 4.6.

4.12.2.2 Interrupt on Asynchronous Schedule Synched

This event is used by the WHCD to determine that the WHC has synchronized its internal cache of the asynchronous schedule with the schedule on the system memory. If the *Interrupt on Asynchronous Schedule Synched Enable* bit in the WUSBINTR register is a one, the WHC must issue a hardware interrupt. A detailed explanation of this feature is described in Section 4.6.

4.12.2.3 BPST Adjustment Changed

The URC must synchronize its local clock with the superframe by adjusting its Beacon Period Start Time (BPST) when it receives a beacon from a device with a slower clock. The WHC must maintain the BPST adjustment value in every superframe. The BPST adjustment value is the difference between the final BPST after the beacon period and the original BPST before the beacon period. The BPST adjustment value is maintained in the *BPST Adjustment* field in the WUSBBPST register. If this adjustment value changed from the previous superframe, the WHC sets the *BPST Adjustment Changed* bit in the WUSBSTS register. If the *BPST Adjustment Changed Enable* bit is set in the WUSBINTR register, a hardware interrupt is signaled.

4.12.2.4 Channel Time Rollover

When the Wireless USB Channel Time value in the WUSBTIME register wraps back to a zero, the WHC sets the *Channel Time Rollover* bit in the WUSBSTS register to a one. If the *Channel Time Rollover Enable* bit is set in the WUSBINTR register, a hardware interrupt is signaled.

4.12.2.5 Generic Command Completion

When the WHC completes the command specified in the WUSBGENCMDSTS register and the *Interrupt on Complete* bit is set in that register, the WHC must set the *Generic Command Completion* bit in the WUSBSTS register to a one. If the *Generic Command Completion Enable* bit is set in the WUSBINTR register, a hardware interrupt is signaled.

4.12.2.6 Host System Error

The WHC is a bus master and any interaction between the WHC and the system may experience errors. The type of host error may be catastrophic to the WHC (such as a Master Abort) making it impossible for the WHC to continue in a coherent fashion. In the presence of non-catastrophic host errors, such as parity errors, the WHC could potentially continue operation. The recommended behavior for these types of errors is to escalate it to a catastrophic error and halt the WHC. Host-based error must result in the following actions:

- The *Run/Stop* bit in the WUSBCMD register is set to a zero.
- The following bits in the WUSBSTS register are set:
  - *Host System Error* bit is to a one.
  - *HCHalted* bit is set to a one.
- If the *Host System Error Enable* bit in the WUSBINTR register is a one, then the WHC will issue a hardware interrupt.

Note: After a *Host System Error*, the WHCD must reset the WHC by setting the *WHCReset* bit to a one in the WUSBCMD register before re-initializing and restarting the WHC.

4.13 URC Command/Event Processing

The URC exposes a message based command interface to control the UWB radio, e.g. scan, start beaconing, etc. Each command consists of a command block sent from the URCD to the URC, i.e. Radio Control Command Block (RCCB) and an event block returned from the URC to the URCD, i.e. Radio Control Event Block (RCEB) to return the result of the command. The URC also generates event blocks to notify the URCD of asynchronous events on the URC, e.g. a Beacon Received notification.

In order to issue a radio control command, the URCD must first initialize the URC as described in Section 4.1.1. It (the URCD) can then start sending commands to the URC as described in the following steps:

- Create a command block (command buffer) in system memory.
- Write the physical address of the command buffer to the URCCMDADDR register (see Section 2.3.4).
- Set the *Command Size* field in the URCCMD register (see Section 2.3.1) to the number of bytes in the command buffer and enable the execution of the command via setting the *Active* bit to one.

The URC must set the *Active* bit in the URCCMD register to a zero when it completes reading the command block from system memory and is ready to accept a new command. If the *Interrupt when Ready* bit is set in the URCCMD register, then the URC also sets the *Ready for Command Interrupt (URCRCI)* bit in the URCSTS register (see Section 2.3.2). If the *Ready for Command Interrupt Enable* bit is set in the URCINTR register (see Section 2.3.3), then a hardware interrupt is generated.

To enable event processing so that the URCD can receive event blocks from the URC, the URCD must perform the following steps:

- Allocate a physically contiguous event buffer which is page-aligned and 4K in length.
- Write the physical address of the event buffer to the URCEVTADDR register (see Section 2.3.5). Since the event buffer is page-aligned, the least 12 bits of the address must be zero.
- Set the *Event Address Register Valid* bit to a one in the URCCMD register.

The URCD must not write to the URCEVTADDR register if either the *Event Processing Status* in the URCSTS register or the *Event Address Register Valid* bit in the URCCMD register is set to one.

The URC will indicate that the event processing is enabled by setting the *Event Processing Status* bit in the URCSTS register to a one. It must also set the *Event Address Register Valid* bit in the URCCMD register to a zero if it was set to one as part of the event processing enabling procedure. The URC will place any event to be returned to the URCD in the buffer location pointed to by the URCEVTADDR register. Whenever the URC has written an event block, it increments the *Current Offset* field in the URCEVTADDR register by the size of the event block that has just been written. It is not necessary to place the event block at a DWord-alignment boundary. All the event blocks must be placed consecutively. Figure 4-14 presents an example usage of the event buffer.

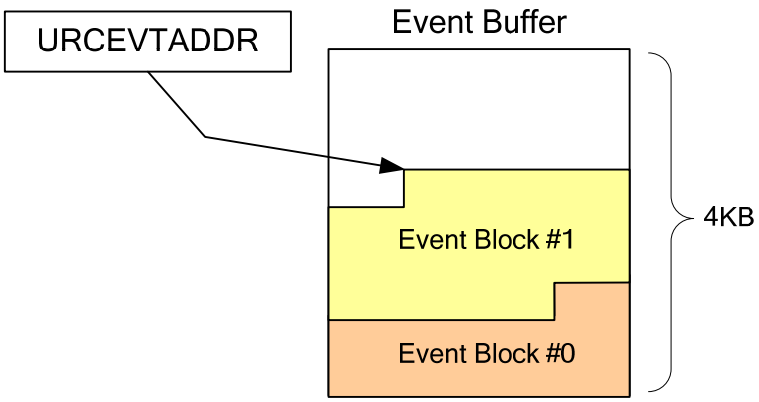


Figure 4-14. Example Event Buffer Usage

If there is not enough space remaining in the event buffer to place the next event block, the URC will retire the current event buffer via setting the *Event Ready (URCER)* bit in the URCSTS register and disable further event processing by setting the *Event Processing Status* bit in the URCSTS register to a zero. If the *Event Ready Enable* bit is set in the URCINTR register, the URC will generate a hardware interrupt. The URC will not perform any event processing until the URCD enables event processing again. The URC must also check the event buffer periodically (every 4.096 ms) and retire it if at least one event block has been placed in that buffer. When the *Event Ready* interrupt occurs, the URCD will process all the event blocks received from the URC. The total size of the event buffer with valid event blocks is determined by the URCD by reading the *Current Offset* field in the URCEVTADDR register.

When the URCD wants to execute a command, it must assign a unique command context value in the range of 1 through 254 and set the *Command Context ID Number* field in the RCCB to that value. When the URC returns a result for this command, it must set the *Event Context ID Number* field in the RCEB to the same value as the *Command Context ID Number* in the RCCB that caused this event to occur. This allows the URCD to associate the received event with the original command. When the URC generates an asynchronous event notification, it must set the *Event Context ID Number* in the RCEB to a zero.

The URC must only accept all of the commands, corresponding events and asynchronous notifications that are detailed in Sections 4.13.2 through 4.13.11. If the URC receives a command that it cannot decode, e.g. invalid *Command Type* or *Command* field value, then the URC must return an RCEB with the *Event* field set to **UNKNOWN_COMMAND_RECEIVED** (see Section 4.13.11.6).

Note that the URC must ignore all asynchronous events that occur until the host enables event processing. However the URC must have internal storage for the results of any commands that are accepted by the URC. These results must be returned to the host when it enables event processing. If the URC does not have enough internal storage to temporarily buffer the result of another command from the URCD then it must not set the *Active* bit to zero in the URCCMD register.

The URCD can disable command and event processing of the URC by resetting the *Run/Stop* bit to a zero in the URCCMD register. The URC will set the *Halted* bit in the URCSTS register to a one and resets the *Event Processing Status* bit in the URCSTS register to a zero. The URC must cancel any pending commands when the *Run/Stop* bit is set to zero.

4.13.1 URC States

On completion of power on reset the URC transitions to the "**Un-Initialized**" state. The URC may be reset from any state by any of the following:

- A hardware driven reset or
- By writing the URCCMD register with the *UWBReset* bit set to one or

The Reset command can only be sent to the URC when it is in one of the Initialized states. See Section 4.13.2 for operational details when this command is executed.

Once the URC is initialized (See Section 4.1) or the URC receives a Reset Command, the URC will transition to the "**Not beaconing**" state. The URCD can instruct the device to move into the "**Beaconing**" state by using the Start Beaconing (See Section 4.13.5.1) command.

All the commands are executable in both the Initialized sub-states (**Beaconing** and **Not Beaconing**). If a command can only be issued to the URC when it is in one of the two sub-states then it will be clearly called out in the section that deals with that command. The various states that URC can be in, is shown in Figure 4-15.

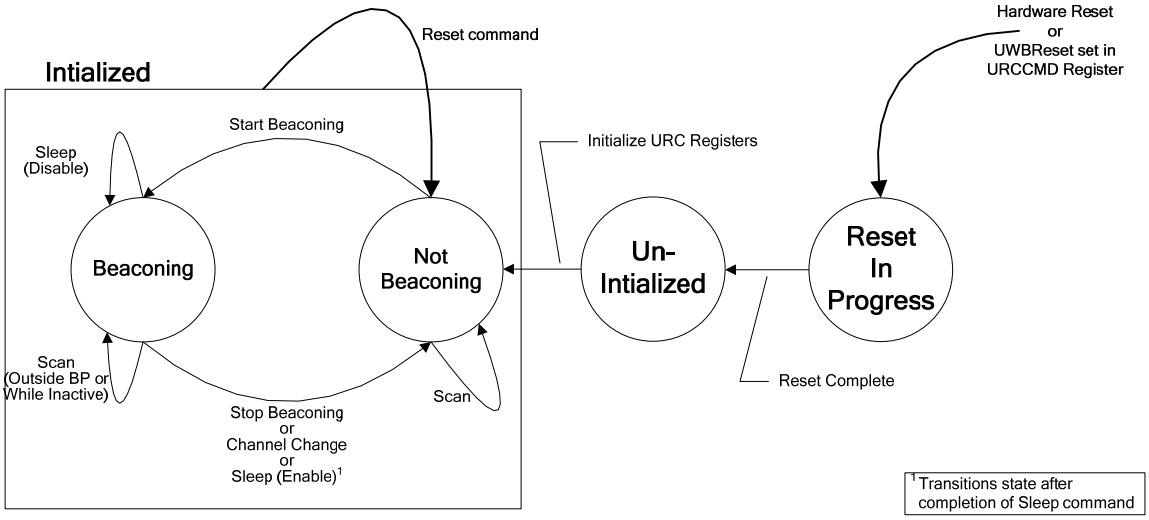


Figure 4-15. URC State Diagram

Note that if a command is issued to the URC that would cause it to change its state (e.g. a Stop Beaconing command, see Section 4.13.5.2, which would result in the URC state transition from **Beaconing** to **Not Beaconing**) then the only command that the URCD may send to the URC during this period is the Reset command (see Section 4.13.3).

4.13.2 Superframe Time Counter

The Superframe Time Counter (STC) is a simple 16-bit counter used to indicate the relative offset in the current superframe from the Beacon Period Start Time (BPST). The value is represented as a microsecond count, with 0 representing the BPST and 65535 representing the end of the superframe. After reaching 65535, the STC wraps back to 0.

In order to adjust for slow-beaconing devices, the STC may skip up to 4 microseconds each superframe, but this may only be done after the end of the Beacon Period.

The STC is a free running timer while the URC is in the "**Not Beaconing**" state. If the URC is scanning then the value of the STC when a beacon is received is returned as the *Receive Time* for that Beacon Received notification.

4.13.3 Reset

The URCD performs a Reset command (see Section 3.1.3.4) when it wants to transition the URC to the "**Not Beaconing**" sub-state. Upon receipt of this command, the URC must reset all the UWB radio control variables and other settings to their initial values as defined in Table 4-8.

Table 4-8. Default Value of URC Internal States and Variables

State / Variable Name	Default Value (after Reset)
Superframe Time Counter	0000h
Scanning State	SCAN_DISABLED
URC State	Not Beaconing
Device Address	Invalid
MAC Address	Default MAC address of the hardware
Beacon Filter State	DISABLED
Notification Filter State	DISABLED

Table 4-8. Default Value of URC Internal States and Variables (cont.)

State / Variable Name	Default Value (after Reset)
Sleep State	DISABLED
Information Elements	Default set of MAC Capabilities IE and PHY Capabilities IE
Transmit Power	Default transmit power (highest power setting)

The URC will confirm the result of the reset operation started due to this command by sending back an RCEB. The *Result Code* field indicates whether the reset operation was successful. If a reset is not succeeding, it is a vendor specific decision when to time out the operation and return failure.

4.13.4 Scan

The initial scanning state after reset is **SCAN_DISABLED**. The URCD can request the URC to scan one or more channels that the URC supports before instructing the URC to start beaconing.

In the Scan command (see Section 3.1.3.5), the *Channel Number* field value is the physical layer channel on which the URC must scan. The encoding of the *Channel Number* is determined by the Band Group and the TF Code of the channel as listed in Table 4-9. Prior to executing this command, the URCD must perform a Get IE command (see Section 4.13.6.1) to query the PHY Capabilities IE so that it can select a Band Group and a TF Code supported by the URC. If the *Channel Number* is unsupported by the URC, then the URC's response to this command is undefined.

Table 4-9. Channel Number Encoding

Channel Number Range	Band Group	TF Code
9 – 15	1	1 – 7
17 – 23	2	1 – 7
25 – 31	3	1 – 7
33 – 39	4	1 – 7
45 – 46	5	5 – 6

Note that the channels using the TF Codes 1-4 are referred to as TFI channels, and the channels using the TF Codes 5-7 are referred to as FFI channels. If the PHY Capabilities IE indicates that the URC supports a TFI channel of a particular band group, it means it can support all the TFI channels of this band group.

Upon reception of a Scan command, the URC will start or stop scanning as indicated by the *Scan State* field in the command. If the *Scan State* is **SCAN_DISABLED**, the URC stops scanning, otherwise it starts scanning a channel. It depends on the *Scan State* when the URC must listen on the specified channel as follows:

- If the URCD sends a Scan command with the *Scan State* set to **SCAN_ONLY**, the URC must only perform scanning of the specified channel until the URCD sends another Scan command with the *Scan State* set to **SCAN_DISABLED**. If the URCD requests a Scan command with the *Scan State* set to **SCAN_ONLY** when the URC is actively sending beacons then the URC's response to this command is undefined.
- If the *Scan State* is set to **SCAN_OUTSIDE_BP**, then the URC will scan on the specified channel except during its beacon period. This command is invalid if the URC is not currently beaconing.
- If the *Scan State* is set to **SCAN_WHILE_INACTIVE**, then the URC will scan on the specified channel when it is not actively transmitting or receiving. The URC must not scan during its beacon period as well. This command is invalid if the URC is not currently beaconing.
- If the *Scan State* is set to **SCAN_ONLY_STARTTIME**, then the URC must stop scanning the specified channel if it was previously scanning the channel. It must start scanning the specified channel when the STC value is equal to the *Start Time* value specified in the command. If the current STC value

is greater than the *Start Time* value then the URC must start scanning in the next superframe when the STC value equals the *Start Time* value.

Note that once the URC starts scanning, it behaves as if it had received a command with the *Scan State* set to **SCAN_ONLY**.

Table 4-10 shows the availability of the each type of scan in each URC State. “Yes” means the specified scan state is valid and the URC will accept the command. On the other hand, “No” means the scan state is invalid in the current URC state.

Table 4-10. Scan State availability in URC States

Scan State	URC State		
	Un-Initialized	Not Beaconing	Beaconing
SCAN_DISABLED	No	Yes	Yes
SCAN_ONLY	No	Yes	No
SCAN_OUTSIDE_BP	No	No	Yes
SCAN_WHILE_INACTIVE	No	No	Yes
SCAN_ONLY_STARTTIME	No	Yes	No

Additionally, the URC must accept being placed in a new scan state from its current scan state only as outlined in Table 4-11.

Table 4-11. New Scan State availability in Current Scan States

New Scan State	Current Scan State				
	S_D	S_O	S_O_BP	S_W_I	S_O_ST
SCAN_DISABLED (S_D)	No	Yes	Yes	Yes	Yes
SCAN_ONLY (S_O)	Yes	No	No	No	No
SCAN_OUTSIDE_BP (S_O_BP)	Yes	No	No	Yes	No
SCAN_WHILE_INACTIVE (S_W_I)	Yes	No	Yes	No	No
SCAN_ONLY_STARTTIME (S_O_ST)	No	Yes	No	No	Yes

The URC will confirm the completion, successfully or in error, of a scanning state change initiated by this command by sending back an RCEB.

The URC will send back any beacons it receives during the scan using the Beacon Received notification (see Section 4.13.11.2). The beacon filter (see Section 4.13.10.2) is ignored while the URC is scanning.

The *Result Code* field indicates whether the operation to change the scan state has successfully completed. If the new Scan State is illegal in the current URC State or current Scan State, then the URC must set the *Result Code* field value to **FAILURE_INVALID_PARAMETER**. If changing the scan state is not succeeding, it is a vendor specific decision when to time out the operation and return failure.

4.13.5 Beaconing

The URC provides commands and notifications to control beaconing as specified in the following sections.

4.13.5.1 Start Beaconing

The URCD will request the URC to start beaconing on the specified channel by sending a Start Beaconing command (see Section 3.1.3.12). This command is only valid when the URC is in the “**Not Beaconing**” state and the URC Scan State is set to **SCAN_DISABLED**.

If the URC is currently beaconing, then the URC's response to this command is undefined.

The *Channel Number* field value is the physical layer channel on which the URC must beacon and must be encoded as listed in Table 4-9. If the *Channel Number* is unsupported by the URC, then the URC's response to this command is undefined.

Upon reception of this command, the URC must listen on at least one superframe from the STC time specified by the *BPST* field before actually starting beacons. If the URC receives a beacon, then it must join the same beacon group as the first beacon received and start beaconing. If the URC does not receive any beacon, then it must establish a new beacon group. If the URCD received Beacon Received notifications (see Section 4.13.11.2) during a prior scan operation and it wants the URC to join a particular beacon group, then it may set the *BPST* field to the STC value which corresponds to the beacon period start time of that beacon group calculated from the *Receive Time* of the Beacon Received notifications.

If the URCD did not receive any Beacon Received notifications while scanning, then the URCD must set the *BPST* field value to zero.

The URC will confirm that the first beacon has been successfully transmitted or that the operation has failed by sending back an RCEB. If the URC successfully starts beaconing then it must listen to all the beacons during its beacon period and return any Beacon Received notifications that are not filtered to the URCD. In addition the Superframe Time Counter (STC) must be adjusted so that the zero of the STC corresponds to the new BPST.

The *Result Code* field in the RCEB indicates whether beacons have begun to be successfully transmitted. If an operation to start beaconing is not succeeding, it is a vendor specific decision when to time out the operation and return failure.

4.13.5.2 Stop Beaconing

The URCD will request the URC to stop beaconing by sending a Stop Beaconing command (see Section 3.1.3.13). This command is only valid when the URC is in the "**Beaconing**" state and the URC Scan State is set to **SCAN_DISABLED**.

If the URC is not currently beaconing, then the URC's response to this command is undefined.

This command is sent by the URCD to instruct the URC to stop beaconing. The URC will confirm that beaconing has been stopped by sending back an RCEB. The *Result Code* field in the RCEB indicates whether the beaconing operation was successfully stopped. A beacon period must have passed without a beacon being transmitted by this URC before the RCEB for this command is returned to the URCD. If ending beaconing is not succeeding, it is a vendor specific decision when to time out the operation and return failure.

The STC will revert back to a free running timer until the URC starts beaconing again.

4.13.5.3 BP Merge

This command (see Section 3.1.3.14) is used to start or abort a beacon period merge operation.

The URCD must issue this command if it receives one or more alien beacons (due to the URC sending a Beacon Received notification with *Beacon Type* field set to **Overlapping**/**Non-Overlapping Alien**) or after receiving a BP Switch IE Received notification from the URC which informs the URCD that the URC detected a BP Switch IE in a beacon from one of its neighbors.

The *BPST Offset* field must be set to the offset of the alien BPST with respect to the current BPST of the URC. The *Beacon Slot Offset* field value must be determined by the URCD from the alien beacons or it must be set to zero if the URCD wants the URC to join the alien BP using normal join rules.

Upon reception of this command, if the *BP Move Countdown* field is not set to zero, the URC must include a BP Switch IE in its beacon for the subsequent superframes until the URC starts beaconing in the new beacon period. The *Beacon Slot Offset* and the *BPST Offset* field in this IE must be set to the values specified in this command. Every time the URC sends a BP Switch IE, it must send a BP Switch Status Notification back to

the URCD. If the *BP Move Countdown* field is set to zero, the URC must realign its BPST immediately and send a BP Switch Status notification indicating that the merge operation was completed.

The URC will complete this command immediately by sending an RCEB to the URCD. The *Result Code* field indicates the result of the command. If this operation to merge is not succeeding then it is a vendor specific decision when to time out the operation and return failure.

If the URCD issues this command while the URC is already sending a BP Switch IE in its beacon, the URC must restart the BP merge operation by sending a modified BP Switch IE with the newly specified parameters.

The URCD may abort a previously initiated BP merge operation by executing another BP Merge command with the *BPST Offset* field set to 65535 and the *Beacon Slot Offset* field set to zero. When the URC receives this command, it must stop the BP merge operation and start sending a BP Switch IE to inform its neighbors that the URC is going to abort the BP merge operation. The *BPST Offset* in the BP Switch IE must be set to 65535, *Beacon Slot Offset* field set to zero and the *BP Move Countdown* field to 9. The URC will complete this command immediately. After sending a BP Switch IE with its *BP Move Countdown* field set to zero, the URC will remove the BP Switch IE from its beacon and send a BP Switch Status notification indicating that the BP merge was aborted successfully. If the BP merge is aborted after the URC has already relocated its beacon, the URC must simply complete the command by sending an RCEB with the *Result Code* set to **FAILURE_INVALID_STATE**.

Figure 4-16 shows the flowchart for an example BP Merge scenario.

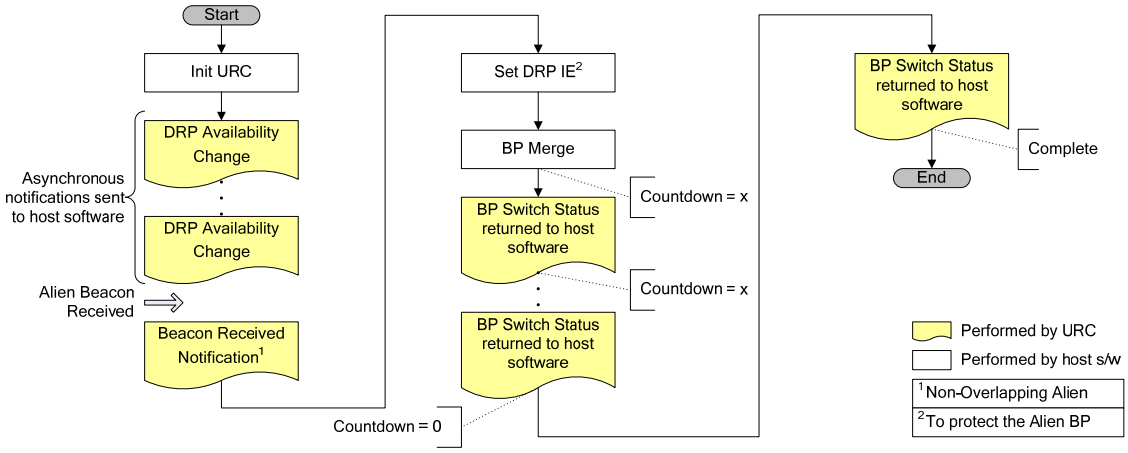


Figure 4-16. Example BP Merge Scenario

In the example above:

- The URC sends a Beacon Received notification with the *Beacon Type* field set to **Non-Overlapping Alien** when it receives a non-overlapping alien beacon.
- Upon reception of this notification, the URCD is responsible for protecting the newly detected alien beacon period and may do so by using the Set DRP IE command (see Section 4.13.8.1). The URCD will then determine whether to move its beacon from the current beacon period to the new beacon period and sends a BP Merge command to the URC if it determines that the URC needs to move to the new beacon period.
- The URCD determines the new BPST and new Beacon Slot in the new beacon group and sends this information as part of the BP Merge command. Once the URC receives the BP Merge command, it starts the BP Merge operation and sends back BP Switch Status notifications (with *Status* field set to **Continue**) to inform the URCD about the progress of the BP Merge operation.
- Once the URC sends its first beacon in the new beacon group, the URC must send a final BP Switch Status notification with the *Status* field set to **Done**.

The URCD may initiate a BP merge operation when it receives a BP Switch IE Received notification from the URC. In this case, the URCD must use the *BPST Offset* and *BP Move Countdown* values from the BP Switch IE Received notification. The WHC does not need to protect the alien beacon period in this case

If the Alien BP is overlapping then the URCD sets the *BP Move Countdown* field to zero and the *BPST Offset* field to the new BPST.

Note that the URCD is responsible for re-establishing reservations and setting the beacon filter once the URC has started beaconing in the new beacon group.

4.13.5.4 BP Switch IE Received Notification

This notification (see Section 3.1.4.6) informs the URCD that the URC detected a BP Switch IE in a beacon from a neighbor device. Upon reception of this notification, the URCD must start the beacon merge operation by issuing a BP Merge command.

Note that the URC must continue to send this notification to the URCD as long as it detects a BP Switch IE in beacons from neighboring devices even if the URCD has initiated a BP Merge operation.

4.13.5.5 BP Switch Status Notification

This notification (see Section 3.1.4.10) informs the URCD that the URC has sent its beacon including a BP Switch IE, relocated its beacon to an alien BP or stopped the BP merge operation.

The URC must generate this notification whenever it sends a BP Switch IE in its beacon. The *Beacon Slot Offset* field, the *BPST Offset* field and the *BP Move Countdown* field are set to the values in the BP Switch IE respectively. The *Status* field must be set to one (**Continue**) when this notification is generated upon transmission of a beacon with a BP Switch IE (except for halting the relocation process).

The URC is responsible for updating the *BPST Offset*, the *Beacon Slot Offset* and the *BP Move Countdown* fields in the BP Switch IE it transmits upon reception of a neighbor's beacon with a BP Switch IE or alien beacons in every superframe.

After the URC transmits a beacon including a BP Switch IE with its *BP Move Countdown* field set to zero or the URC receives a BP Merge command with the *BP Move Countdown* field set to zero, the URC will adjust its BPST with the alien BP and start sending its beacon. The URC must generate this notification with the *Status* field set to zero (**Done**) after sending the first beacon in the new BP.

When the URC detects that the peer device has stopped sending a BP Switch IE or it sees a BP Switch IE in alien beacons that indicate the alien devices will relocate its BP earlier than the URC, it must halt the relocation process by including a BP Switch IE with the *BPST Offset* field set to 65535 in its beacon. The *Status* field in this notification must be set to either two (**Halted (Neighbor halts the relocation process)**) or three (**Halted (Alien relocates its beacon earlier)**).

When the URC receives a BP Merge command with the *BPST Offset* field set to 65535 and the *Beacon Slot offset* field set to zero, the URC must halt the BP relocation process by including a BP Switch IE with the *BPST Offset* field set to 65535 in its beacon. The *Status* field in this notification must be set to four (**Aborted**).

4.13.5.6 DRP Modification for New BP

The URCD may modify the DRP IE for the existing reservations after the URC realigns its BPST with the alien BP. In order to transmit the new set of DRP IEs with the first beacon in the new BP, the URCD should set it by using a Set DRP IE command after it receives a BP Switch Status Notification with the *Status* field set to one (**Continue**) and the *BP Move Countdown* field set to zero.

Once the URC has relocated its beacon, the URCD must remove the DRP IE that it had created to protect the alien BP.

4.13.6 IE Management

4.13.6.1 Get IE

The URCD can get the set of IEs currently sent in beacons by using a Get IE command (see Section 3.1.3.3).

If the URC is beaconing then the URC must return all the IEs that are being transmitted in the beacon by the URC. If the URC is not beaconing then it must either:

- Return the IEs that have been set by the URCD by a previous Set IE command or
- Return the default set of MAC Capabilities IE and PHY Capabilities IE if the URCD has not set any IEs.

The URC will return the IEs in the RCEB for this command.

4.13.6.2 Set IE

The URCD can set one or more IEs that are being transmitted in its beacons by using a Set IE command (see Section 3.1.3.8).

The *IE Data* field in this command is the data to be added to the beacon. This may contain one or more IEs which are defined by the URCD. This data replaces any data from a previous Set IE command. The length of the *IE Data* field is specified by *IE Length*. This field must be set to zero if the URCD wants to remove all the IEs that it had previously set. The *IE Data* field is not present when this field is set to zero.

If the URC cannot decode the contents of the *IE Data* field correctly, then it must set the Result Code field in the RCEB for this field to **FAILURE_INVALID_IE_DATA** and fail this command.

If the amount of time in a beacon slot is insufficient to send the URC's beacon (by adding this IE block), then URC must not add this IE block to its beacon, set the Result Code field in the RCEB for this field to **FAILURE_BEACON_SIZE_EXCEEDED** and fail this command.

The complete list of Information Elements available is listed in the MAC layer specification. The URCD can set the IEs listed in Table 4-12 using this command. The behavior of the URC is undefined if other IEs are set using this command.

Table 4-12. Host Settable IEs

IE Name	Notes
MAC Capabilities IE	Used by Wireless USB
PHY Capabilities IE	
Identification IE	
PCA Availability IE	
Probe IE	
Application-specific Probe IE	
Master Key Identifier (MKID) IE	
Application Specific IE (ASIE)	
Relinquish Request IE	
Multicast Address Binding IE	

The URC must not reorder the IEs set by this command. However it must correctly insert the other IEs it generates as well as the DRP IEs that may be set using the Set DRP IE (see Section 4.13.8.1) command in the set of IEs that it transmits in its beacon.

The URC will confirm the result of the operation to set zero or more IEs in the beacon by sending back an RCEB. The URC must send the RCEB after sending the first beacon that includes the IEs specified in this command. Note that the URC must locally store the contents of the *IE Data* field in the RCCB.

4.13.7 Device Address Management

The URCD can query or set the 16-bit device address or the 48-bit MAC address (EUI-48) used by the URC by using the Device Address Management command (see Section 3.1.3.2).

The type of operation to perform is indicated by the *Operation Type*. When the address type is zero and the *Set* bit is zero, the command is used to get the current 16-bit device address used by the URC. The URC is not required to generate a device address. This command just returns the address which was previously set by the URCD.

When the address type is zero and the *Set* bit is one, the command is used to set a new WiMedia MAC 16-bit device address. The URCD is responsible for generating this device address. The URC must use the specified address as the source address of all the outgoing MAC frames including beacons in subsequent superframes. The URC does not have a valid device address after a reset operation, therefore the URCD must generate a valid device address and set it to the URC by using this command before it issues a Start Beaconing command.

When the address type is one and the *Set* bit is zero, the command is used to get the current 48-bit MAC address (EUI-48) used by the URC. The address is either statically preconfigured in the UWB hardware or temporally assigned by the URCD.

When the address type is one and the *Set* bit is one, the command is used to set a new 48-bit MAC address (EUI-48) to the URC. The device must use this address as the *Device Identifier* field in the *Beacon Parameters* of its beacons in subsequent superframes. The URCD is also required to set the 16 bit Device Address if it changes the 48-bit MAC address.

If the address type is invalid then the URC's response to this command is undefined.

The *Address* field contains the address information to be used in the case of set operation, i.e. the *Set* bit is one. The valid bytes depend on the address type. If the address type is zero (16-bit device address) then the device address must be stored in the first 2 bytes of the *Address* field.

The *Result Code* field in the RCEB indicates whether the device address management operation was successful. If the operation is not succeeding, it is a vendor specific decision when to time out the operation and return failure.

The address information is returned to the URCD in the *Address* field in the case of a query operation, i.e. the *Set* bit in the RCCB is set to a zero. The valid bytes depend on the address type specified in the associated RCCB. If the address type is zero (16-bit device address) then the device address must be stored in the first 2 bytes (byte offsets 5 and 6) of the *Address* field.

4.13.7.1 Device Address Conflict Notification

This notification (see Section 3.1.4.7) informs the URCD that the URC has detected a device address conflict. The URCD must re-generate a new device address and set it to the URC by using the Device Address Management command (see Section 4.13.7).

4.13.8 DRP Management

The URC provides a number of commands and notifications for handling DRP (Distributed Reservation Protocol) used to reserve medium access slots on the UWB radio.

4.13.8.1 Set DRP IE

This command (see Section 3.1.3.7) is used to set one or more DRP IEs to be transmitted in the beacon by the URC. This command is only valid in the "**Beaconing**" state.

The *IE Data* field in this command contains zero or more DRP IEs to be sent in the beacon. It may also contain a DRP Availability IE when the URCD wants the URC to send it. The length of the *IE Data* field is specified by the *IE Length* field.

If the URC cannot decode the contents of the *IE Data* field correctly, then it must set the *Result Code* field in the RCEB for this field to **FAILURE_INVALID_IE_DATA** and fail this command.

The URCD must set the DRP IE block (zero or more DRP IEs and/or a DRP Availability IE) that it wants the URC to include in its beacon and the URC must replace the existing DRP IE block that it is currently sending in its beacon with the new DRP IE block specified in this command, e.g. if the URCD includes three DRP IEs in the first Set DRP IE command and then includes two DRP IEs in the next Set DRP IE command then the URC will send the DRP IEs specified in the second Set DRP IE command only.

If the amount of time in a beacon slot is insufficient to send the URC's beacon (by adding this DRP IE block), then the URC must not add this DRP IE block to its beacon, set the Result Code field in the RCEB for this field to **FAILURE_BEACON_SIZE_EXCEEDED** and fail this command.

The URCD can also use this command to request the URC to remove the DRP IE block that is currently being transmitted in the beacon by setting the *IE Length* field to zero.

The URC will confirm the result of the operation to set zero or more DRP IEs in the beacon by sending back an RCEB. The URC must send the RCEB after sending the first beacon that includes the DRP IEs specified in this command. Note that the URC must locally store the contents of the *IE Data* field in the RCCB.

The *Result Code* field indicates the result of the attempt to add a DRP IE to the beacon. If an operation to add the DRP IE to the beacon is not succeeding, it is a vendor specific decision when to time out the operation and return failure. The number of remaining bytes left in the beacon must be indicated by the value of *Remaining Space* field.

Note that DRP IEs that must be sent in a command frame are sent to the URC using the Send Command Frame command (see Section 4.13.9.1).

4.13.8.2 DRP Availability Change Notification

This notification (see Section 3.1.4.8) informs the URCD that the URC's local DRP availability has changed. The URC is responsible for maintaining its local DRP availability according to the following rules:

- All the MASs occupied by the beacon period must be marked **Unavailable** and all the bits in the DRP availability corresponding to them must be set to zero. The length of the protected beacon period must be determined by the maximum *BP Length* field value of the BPOIEs in all the beacons received from the neighbors.
- All the MASs declared in DRP IEs in beacons received from its neighbors with the *Target/Owner DevAddr* other than the URC's own address must be marked **Unavailable** and the corresponding bits in the DRP availability must be set to zero.

If the DRP Availability changes then the URC must send a DRP Availability Change Notification.

Note that the URCD is responsible for maintaining the actual DRP Availability IE. If it needs the URC to send this IE, then it does so by including the DRP Availability IE along with the final DRP IEs in a Set DRP IE command.

4.13.8.3 DRP Notification

This notification (see Section 3.1.4.9) reports information about DRP reservations for which the URCD is a participant.

If the URC receives one or more DRP IEs destined to the URC's own device address as part of a peer device's beacon, the URC must generate a single DRP Notification that contains each of these DRP IEs. If the beacon contains a DRP Availability IE, then the URC must also include this IE in the *IE Data* field of the notification. The *Reason Code* field in the RCEB must be set to zero (**DRP IE Received**), the *Source Address* field must be set to the source address of the received beacon and the *Beacon Slot Number* field must be set to the slot number of the received beacon.

The URC must also generate this notification when it receives a multicast DRP reservation request for a multicast group that one or more PALs join and the UMC is configured to receive traffic to that multicast address. A multicast DRP reservation request is received if the URC receives a beacon containing a DRP IE with its *Target/Owner DevAddr* field set to the multicast address. How to specify the multicast addresses to be used is outside the scope of this specification and must be defined in each interface capability. Note that Wireless USB does not use a multicast address and the URC does not need to generate a DRP notification for a multicast reservation request if WHCI is the only interface capability exposed by the UMC.

If a peer device removes all the DRP IEs targeted to the URC from its beacon then the URC must send this notification to inform the URCD that the peer device has terminated all existing reservations. The notification does not include any IE in the *IE Data* field, the *IE Length* field is set to zero, the *Source Address* field must be set to the source address of the received beacon and the *Reason Code* field must be set to three (**Termination**). Note that if the peer device only terminates some of reservations and continues to send DRP IEs for the other reservations then the URC must send a DRP Notification with the *Reason Code* field set to zero (**DRP IE Received**). The URCD will determine the reservations that have been terminated by detecting the DRP IEs that were removed from the peer device's beacon.

This notification is also used to notify the URCD when there is a DRP reservation conflict. The URC must include the DRP IE(s) that caused the conflict in the *IE Data* field. The *Reason Code* field in the RCEB must be set to two (**Conflict**), the *Source Address* field must be set to the source address of the beacon which contains the DRP IE(s) that caused the conflict and the *Beacon Slot Number* field must be set to the slot number of that beacon.

The URCD will evaluate the reservation request/response/conflict/termination and respond to this DRP Notification by using the Set DRP IE (see Section 4.13.8.1) command.

Note that DRP reservations received in command frames are sent to the URCD by using the Command Frame received notification (see Section 4.13.9.2).

4.13.8.4 DRP Command Usage Model

The following sections detail the list of operations that the URCD needs to perform to use the DRP command and notifications. It also describes the duties of the URC when it receives these commands. Each section deals with a DRP reservation request initiated by the URC or by another peer device. The last section deals with DRP reservation conflict resolution and termination.

4.13.8.4.1 DRP Negotiation Initiated by URC

Figure 4-17 shows the flowchart for an Implicit DRP negotiation initiated by the URC, i.e. the URC is the reservation owner.

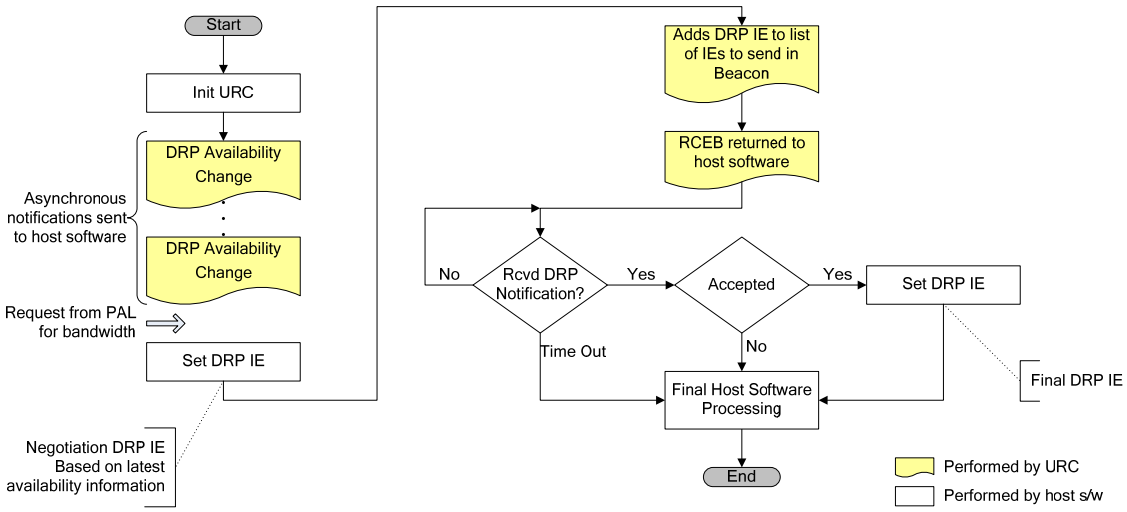


Figure 4-17. Implicit Negotiation Initiated by URC

After the initialization sequence (see Section 4.1), the URC listens for beacons sent from its neighbors. The URC generates a DRP Availability Change notification if it detects any change in its DRP availability. Upon reception of this notification, the URCD must update its local copy of the DRP availability information.

When bandwidth reservation is requested by a PAL, the URCD will determine which MASs to be reserved according to the current DRP availability information and create a Set DRP IE command. The URC will add the DRP IEs to the beacon it is currently sending.

The peer device that was the target of the reservation will respond to the URC sending the above DRP IE by including a reservation response in its beacon. When the URC sees a DRP IE destined to itself in the peer device's beacon, it will generate a DRP notification. The *Reason Code* must be set to zero (**DRP IE Received**).

When the URCD receives a DRP notification including a reservation response to the original request, it will determine whether the DRP reservation request was accepted by the peer or not. If the request was accepted, the URCD will execute another Set DRP IE command to instruct the URC to include the DRP IE for the final reservation (with the *Reservation Status* bit set to one in the DRP IE).

The URCD will then complete the bandwidth reservation request from the requesting PAL. If the request was not accepted by the peer device, the URCD will either inform the result to the PAL or retry the reservation request with modified DRP IEs.

Figure 4-18 shows the flowchart for an Explicit DRP negotiation initiated by the URC, i.e. the URC is the reservation owner.

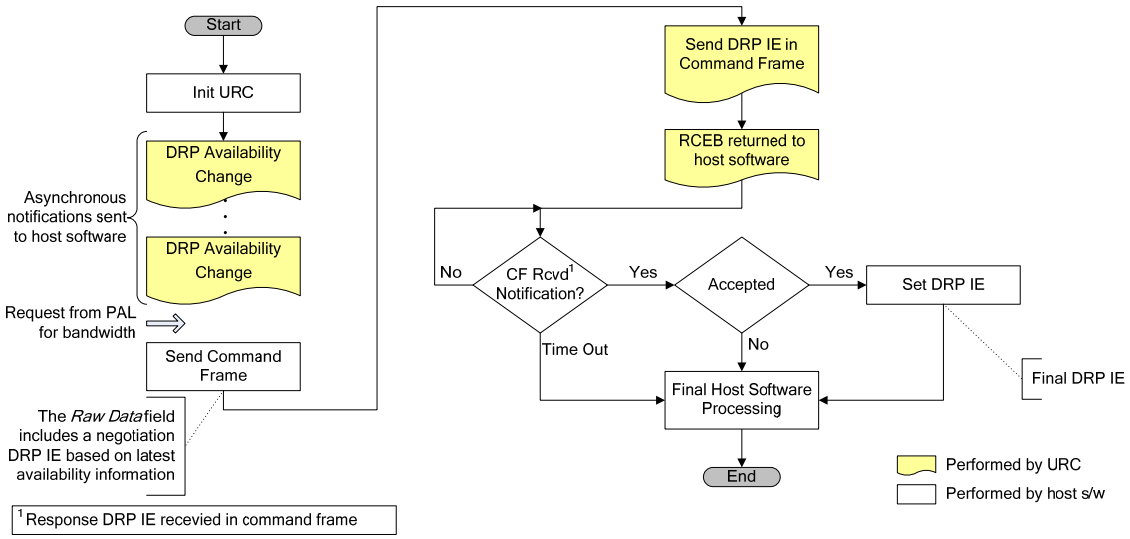


Figure 4-18. Explicit Negotiation Initiated by URC

If the URCD receives a request from a PAL to perform explicit DRP negotiations, the URCD will determine which MASs are to be reserved according to the current DRP availability information and creates a reservation request and sends the DRP IE to the URC in a Send Command frame command. The URC must send this command frame as specified in Section 4.13.9.1.

The peer device will respond to this command frame with another command frame. The URC must send any command frames received using the Command Frame Received notification. It is the responsibility of the URCD to decode the received command frames and detect whether this was a response to the explicit DRP negotiation.

When the URCD receives a command frame including a reservation response to the original request, it will determine whether the DRP reservation request was accepted by the peer or not. If the request was accepted, the URCD will execute another Set DRP IE command to instruct the URC to include the DRP IE for the final reservation (with the *Reservation Status* bit set to one in the DRP IE).

The URCD will then complete the bandwidth reservation request from the requesting PAL. If the request was not accepted by the peer device, the URCD will either inform the result to the PAL or retry the reservation request with modified DRP IEs.

4.13.8.4.2 DRP Negotiation Initiated by Peer Device

Figure 4-19 shows the flowchart for an implicit DRP negotiation initiated by the peer device, i.e. the URC is the reservation target.

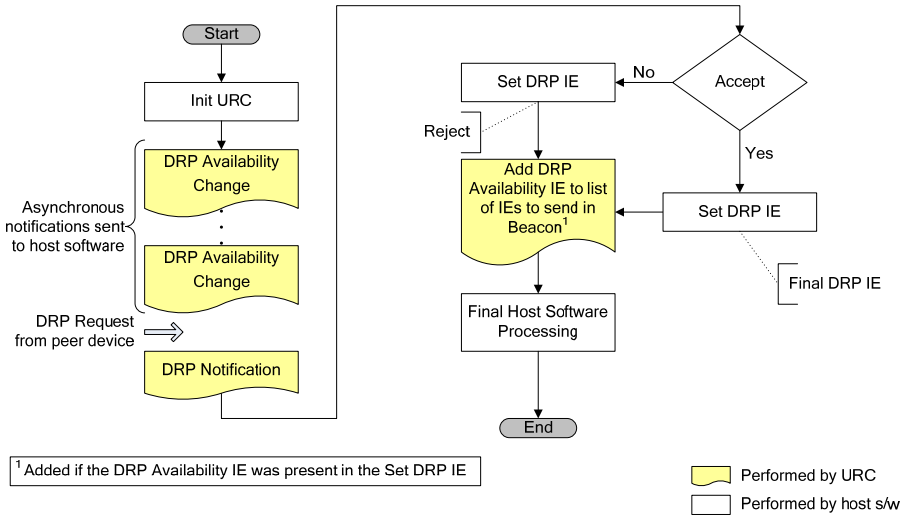


Figure 4-19. Implicit Negotiation Initiated by Peer Device

When the URC receives a beacon containing DRP IEs with its Target/Owner DevAddr set to the URC's own address, it will generate a DRP notification with the *Reason Code* field is set to zero (**DRP IE Received**).

Upon reception of the DRP notification, the URCD will determine whether it will accept the reservation request or not. If it accepts the reservation request, it will create a DRP IE for the response and instruct the URC to send it by using a Set DRP IE command.

If the URCD does not accept the incoming reservation request, it will create a DRP response with a valid reason code and send it to the URC by using a Set DRP IE command. If necessary, the URCD will also include the DRP Availability IE in the *IE Data* field of the command.

If it will take longer than one superframe for the URCD to determine whether or not to accept the reservation request then the URCD must immediately instruct the URC to send a DRP response with the *Reason Code* field set to two (**Pending**) in the DRP IE.

Figure 4-20 shows the flowchart for an explicit DRP negotiation initiated by the peer device, i.e. the URC is the reservation target.

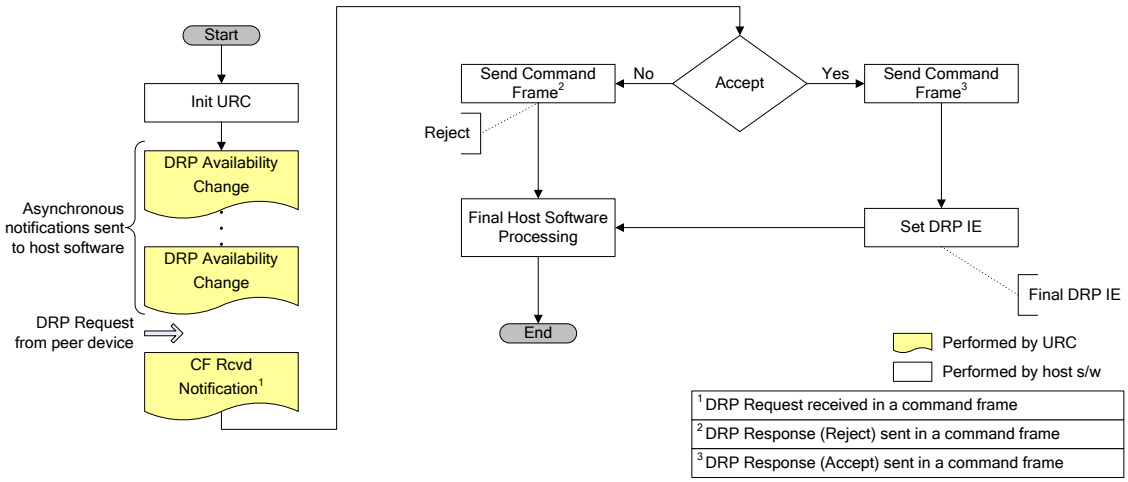


Figure 4-20. Explicit Negotiation Initiated by Peer Device

When the URC receives a DRP reservation request in a command frame destined to the URC, it will generate a Command Frame received notification.

Upon reception of this notification, the URCD will determine whether it will accept the reservation request or not. If it accepts the reservation request, it will create a DRP IE for the response and instruct the URC to send it by using a Send Command Frame command. The URCD must then set the final DRP IE to be sent in the beacon by using a Set DRP IE command.

If the URCD does not accept the incoming reservation request, it will create a DRP response with a valid reason code and send it to the URC by using a Send Command Frame command. If necessary the URCD will include the DRP Availability IE in the *IE Data* field of the command.

4.13.8.4.3 DRP Multicast Reservation

Figure 4-21 shows the flowchart for multicast final reservation.

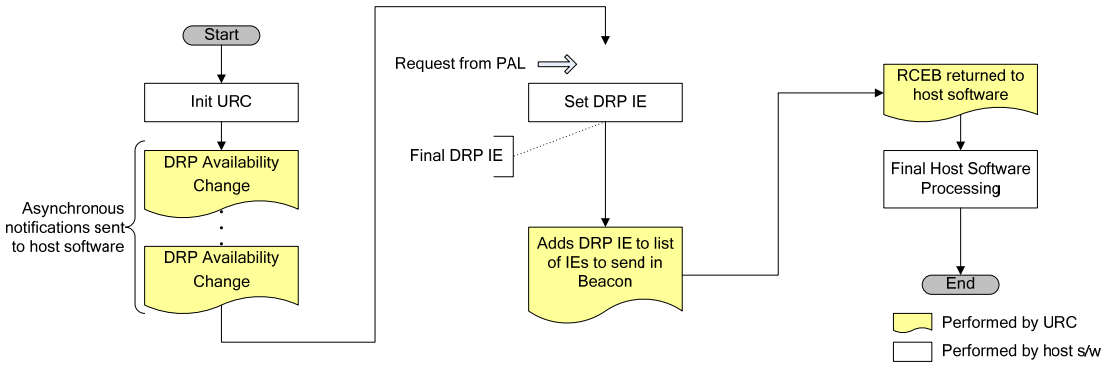


Figure 4-21. Multicast Reservation

When a multicast final reservation is requested by the PAL, the URCD creates a DRP IE and informs the URC by using a Set DRP IE command. The *Reservation Status* field in the DRP IE must be set to one for final reservation. The URC will include the specified set of DRP IEs in its beacon and complete the command by sending an RCEB to the URCD.

4.13.8.4.4 DRP Reservation Conflict Resolution/Termination

Figure 4-22 shows the flowchart for conflict resolution or termination of existing reservations.

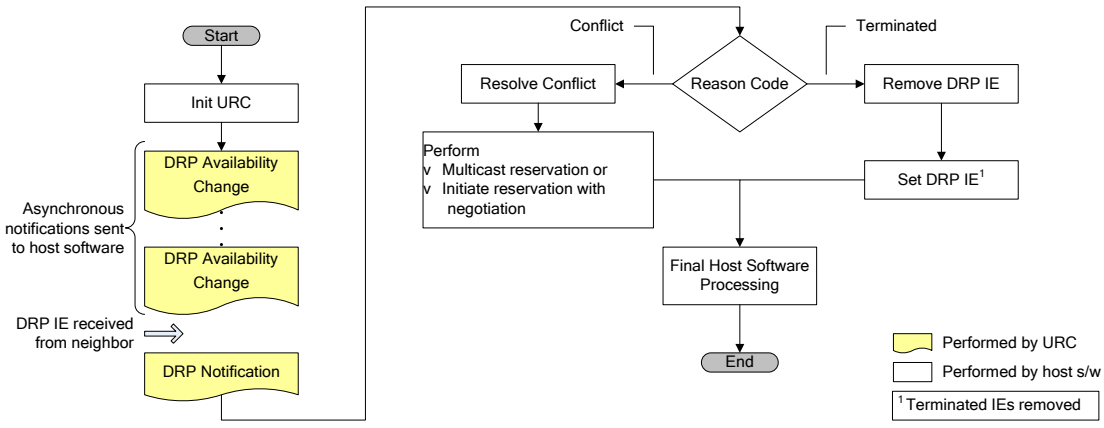


Figure 4-22. Conflict Resolution/Termination

The URC must decode all the DRP IEs in the beacons received from its neighbors and see if they cause a conflict with its existing reservations or not. If one or more DRP IEs in the received beacon contain reservations of the MASs that are already reserved by the URC and the *Target/Owner DevAddr* of these DRP IEs are not equal to the URC's device address, it must generate a DRP notification with its *Reason Code* set to one (**Conflict**). The *IE Data* field in the notification must contain the DRP IEs that caused the conflict. The URCD is responsible for resolving the conflict.

When the URC does not see any DRP IE destined to itself in a beacon received from a neighbor and this neighbor included one or more DRP IEs targeted to the URC in its previous beacon, then it must generate a DRP notification with the *Reason Code* set to two (**Termination**). The *Source Address* field must be set to the address of the peer device which sent this beacon. The URCD will remove all the reservations with this device.

4.13.9 Command Frames

This section explains how the URC manages the transmission and reception of WiMedia command frames. There are 7 types of command frames:

- DRP Reservation Request
- DRP Reservation Response
- Probe
- PTK
- GTK
- Range Measurement
- Application-Specific

DRP Reservation Requests and Response command frames are sent and received using this command when using "Explicit Negotiation."

Probe requests and responses can also be sent in a beacon (implicit) or as a command frame (explicit). Implicit Probe requests and responses are controlled by the SET_IE command. Explicit Probe requests and responses are sent using the Send Command Frame command. Explicit Probe requests and responses are received via the Command Frame Received notification.

The PTK command frames are used to perform the 4-way handshake. There is no implicit version, so these can only be sent and received using the Send Command Frame command and the Command Frame Received notification.

The GTK command frames are used to distribute group keys to members of a multicast group. They can only be sent and received using the Send Command Frame command and the Command Frame Received notification.

The Range Measurement command is used to perform range measurement requests or responses. Range measurement requires special MAC and PHY support not specified here. They can be sent and received using the Send Command Frame command and the Command Frame Received notification.

The content and use of Application-Specific command frames are application specific. They can be sent and received using the Send Command Frame command and the Command Frame Received notification.

4.13.9.1 Send Command Frame

This command is used to transmit a command frame to a receiver or receivers.

The *Destination Address* field is the unicast or multicast destination for the command frame.

The *Frame Subtype* field indicates the type of command to be sent. The value is defined in the MAC layer specification.

Command frames may be sent with a No-ACK or Imm-ACK policy, and this is represented in the *ACK Policy* field in the MAC header. The Imm-ACK policy should be used unless the Destination Address targets a multicast group, in which case the policy must be No-ACK.

When the *Access Method* field is set to zero, the URC must send the frame using PCA. The URC must determine the PCA parameters such as AIFS using the access category specified in the *Access Category* field.

Frames sent using PCA can also be protected using the RTS/CTS exchange. If the *RTS/CTS Policy* field is set to one, the URC will use a RTS/CTS exchange before transmitting the command frame. The *RTS/CTS Policy* field is only valid when the *Access Method* field is set to PCA.

The *MAS Number* field indicates the Medium Access Slot during which the URC must attempt to transmit the command frame. The URCD will fill in the *MAS Number* field after considering when the destination device(s) will be listening based on existing DRP reservations or the PCA Availability IE.

When the *Access Method* field is set to PCA, the URC may be unable to send the command frame during the specified MAS because the link is busy or the MAS is part of a DRP reservation. If the *MAS Number* field is set to zero when the *Access Method* field is set to PCA, the URC may send the frame in any MAS that is available for PCA.

When the *Access Method* field is set to DRP, the URC may be unable to send the command frame during the specified MAS because the MAS is not part of an owned, active DRP reservation with the destination device. If the *MAS Number* field is set to zero when the *Access Method* field is set to DRP, the URC may send the frame in any MAS slot that corresponds to an owned, active reservation with the destination device.

The *PHYRate* field tells the URC which rate to use when transmitting the payload of the command frame.

When the *Secure Policy* field is set to one, the URC must encrypt the command frame before transmission using the encryption key identified by the *TKID* field. If the Destination Address of the command frame is a unicast address, then the *TKID* field must contain a valid PTKID. If the Destination Address of the command frame is a multicast address, then the *TKID* field must contain a valid GTKID.

When the *Secure Policy* field is set to zero, the URC must not encrypt the command frame before transmission and the *TKID* field is not valid.

The *Retry Count* field is the number of attempts the URC must make to transmit the command frame. The *Retry Count* field is not valid when the *ACK Policy* field is set to No-ACK.

The *Encryption Offset* field is only valid when the *Secure Policy* bit is set to one. It specifies the number of bytes of payload that must be sent encrypted.

The *Command Payload* field contains the raw data for the payload portion of the command frame to be transmitted. The length of the command payload data is specified in the *Command Payload Data Length* field. Fragmentation is not supported for command frames and the URC must transmit the entire command frame in a single MAC frame. The URC must set all the fields in the MAC header to the appropriate values including the *Sequence Number*, the *Duration* and the *More Frames* fields. The *Sequence Number* must be determined using a dedicated counter for all the command frames (including command frames generated via other interfaces exposed by the hardware).

If the command frame is transmitted and the *ACK Policy* was set to Imm-ACK, the URC will listen for an ACK frame from the destination device. If an ACK is received, the URC will return an RCEB with the Result Code equal to SUCCESS. If an ACK is not received, the URC will attempt to retransmit the command frame using the Access Method specified by the URCD for up to *Retry Count* times. If none of the retries are successful the URC will return an RCEB with the Result Code equal to **FAILURE_ACK_NOT_RECEIVED**.

If the *ACK Policy* is set to No-ACK, the URC will return an RCEB with the Result Code equal to **SUCCESS** after the command frame is transmitted.

If the URC is unable to transmit the frame using the Access Method specified by the URCD for 3 superframes, it will return a RCEB with the Result Code **FAILURE_TIME_OUT**.

If the *Secure Policy* field is set to one and the URC does not have an encryption key for the key id identified in the *TKID* field, it will return a RCEB with the Result Code **FAILURE_INVALID_PARAMETER**.

If the *Secure Policy* field is set to one and the value in the *Encryption Offset* field is greater than the value in the *Command Payload Data Length* field then the URC must return an RCEB with the Result Code equal to **FAILURE_INVALID_PARAMETER**.

If the URC is transmitting the Command Frame directly from the Command Block, the URC may not set the Active bit in the URCCMD register to zero until after it has returned an RCEB for the transmission, which could be up to 3 superframes.

4.13.9.2 Command Frame Received

This notification indicates that the URC received a command frame targeted to the URC's device address during a period of time that it was listening to the medium. If the URC has been set to filter Command Frame Received notifications then the URC will not notify the URCD of any received command frame.

The URC must also generate this notification when it receives command frame targeted to a multicast group that one or more PALs join and the URC is configured to receive traffic to that multicast address. How to specify the multicast addresses to be used is outside the scope of this specification and must be defined in each interface capability. Note that Wireless USB does not use a multicast address and the URC does not need to generate a Command Frame Received notification targeted to a multicast address if WHCI is the only interface capability exposed by the URC.

The STC value at the time the command frame was received is reported by the URC in the *Receive Time* field.

The *Source Address* field gives the address of the device that sent the Command Frame.

The *Destination Address* field gives the address of the targeted device.

The *Frame Subtype* field indicates the type of command received.

The *Ack Policy* field indicates the ACK Policy used by the device that sent the Command Frame.

The *PHYRate* field indicates the rate at which the payload of the command frame was received.

The *Secure Frame Bit* is set if this was a command frame was received as a secure frame.

The *Command Payload* field contains the raw data for the payload portion of the command frame received. The length of the command payload data is specified in the *Command Payload Data Length* field.

4.13.10 Other Commands

4.13.10.1 Channel Change

This command (see Section 3.1.3.1) is used to inform other devices that the URC is going to change the channel. The URC must send a Channel Change IE in its beacons for the number of times specified in the command. This command is only valid when the URC is in the "**Beaconing**" state and the URC Scan State is set to **SCAN_DISABLED**.

If the *Channel Change Countdown* field is not zero, then this field indicates the number of superframes until the URC actually changes to the new channel. The URC must send the IE in the beacon until it completes the channel change operation. The URC must set the *Channel Change Countdown* field in the Channel Change IE to the specified value in the first beacon after receiving this command and it must decrement the countdown value by one in each subsequent beacon.

If the *Channel Change Countdown* field is zero, then this command will cancel a previous Channel Change command. If a previous Channel Change command was not sent to the URC, then the URC's response to this command is undefined.

The *New Channel Number* field specifies the channel number of the new channel and the URC sets the *New Channel Number* field in the Channel Change IE to this value.

The URC will confirm that it has sent the last beacon with its *Channel Change Countdown* value set to zero successfully on the current channel or that the operation has failed by sending back an RCEB.

The *Result Code* field indicates whether the channel change operation was successful. If the operation is not succeeding, it is a vendor specific decision when to time out the operation and return failure. The STC (see Section 4.13.2) will revert back to a free running timer until the device starts beaconing again. On completion of this command the URC transitions to the "**Not Beaconing**" state.

4.13.10.2 Set Beacon Filter

This command (see Section 3.1.3.6) is used to set the Beacon filter. This filter is based on the beacon slot number. This command is only valid in the "**Beaconing**" state and only applicable to the neighbor beacons.

The beacon filter is disabled after reset and the URC must generate a Beacon Received notification for any beacon received.

Upon reception of this command with the *Enable State* bit set to one, the URC must store the *Beacon Slot Filter Mask* internally and determine whether it generates a Beacon Received notification or not for each received beacon. If the URC detects a beacon in a slot whose bit is not set in *Beacon Slot FilterMask* or the Device Address of the device sending the beacon in a slot changes from one superframe to the next then it must send a Beacon Received notification to the host.

If a beacon is not detected in a slot for more than 3 superframes then the URC must generate a BPOIE Change notification (See Section 4.13.11.4) regardless of the bit value in the *Beacon Slot Filter Mask.*

This filter information is only valid when the URC is actively sending beacons and during the beacon group which the URC has joined.

The *Result Code* field indicates whether the Set Beacon Filter operation was successful. If the operation is not succeeding, it is a vendor specific decision when to time out the operation and return failure.

4.13.10.3 Set Notification Filter

This command (see Section 3.1.3.9) instructs the URC to filter one or more notifications.

The notification filter is disabled after reset and the URC must generate a notification for any asynchronous event.

Upon reception of this command with the *Enable State* bit set to one, the URC must store the *Notification Mask* internally and determine whether it generates a notification or not for each asynchronous event. If the bit in the *Notification Mask* is set to one, then the URC must not generate the corresponding notification.

The URC will confirm the result of the set notification filter operation by sending back an RCEB. The *Result Code* field indicates whether the set beacon filter operation was successful. If the operation is not succeeding, it is a vendor specific decision when to time out the operation and return failure.

4.13.10.4 Set TX Power

The URCD will use a Set TX Power command (see Section 3.1.3.10) to set the default transmit power for transmissions by the URC as per the *Wireless USB Specification Revision 1.0*.

If the power level for transmission of a packet is not set by other means, then the URC and all the other interfaces must use the power level specified in *Power Level* to transmit that packet.

The default state on power up or after a reset command completes successfully is the highest power level.

Note that in the case of WHCI, the WHC uses this power level only for transmitting MMCs. As for transmission of data packets to a particular endpoint, it uses the power level specified in the *TxPwr* field of the QHead (see Section 3.2.6 or Section 3.2.7).

4.13.10.5 Sleep

The URC Sleep command (see Section 3.1.3.11) enables (or disables) the URC for low power operation mode which takes effect after the interface has been stopped. The externally visible operation of the host is that it cycles in an out of hibernation mode. The *Hibernation Count* parameter specifies the number of superframes the host will be beaconing before entering hibernation mode. The *Hibernation Duration* parameter is the number of superframes the host will be in hibernation mode. Essentially, *Hibernation Count* is the number of superframes ‘up’ per cycle and *Hibernation Duration* is the number of superframes ‘down’ per sleep cycle. Note that these parameters must not conflict with the WHC’s *Remote Wake Poll Interval* value. The general requirement is that the *Remote Wake Poll Interval* must coincide with when the host in the awake (or ‘up’) portion of a sleep cycle. This command is only valid when the URC is in the “**Beaconing**” state and the URC Scan State is set to **SCAN_DISABLED**.

This command will have no effect on operation if submitted to the URC interface when the URCCMC.run/stop register bit is a zero. It will only effect operation when the run/stop bit is a one. When enabled for Sleep mode operation, the URC will remain in Sleep mode operation until all power (including auxiliary power) is removed, a hard PCI reset occurs or the URCD sets the run/stop bit of the URCCMD register to a one.

The URCD uses the Notification filter (see Section 3.1.3.9) to select which WiMedia MAC events will cause the URC to alert the system. If the device function is in D3, it will assert PME#, otherwise it will assert an interrupt. The URC is not required to keep the host out of hibernation after observing a valid event matching the Notification filter (one for which the URC is enabled to pass on to the system). Rather, it is minimally required to retain the specific notification/event and assert PME# or an interrupt (which ever is appropriate for the device state). As the system returns to operational mode, the URC device driver will be notified of the stored event after the interface has been correctly reactivated.

The Sleep command completes after the URC has transmitted the last beacon with the *Hibernation Countdown* field set to zero in the Hibernation Mode IE.

The URCD can cancel a previously issued Sleep command by sending a new Sleep command with the *Enable/Disable* field in the RCCB set to zero.

The *Result Code* field (in the Sleep RCEB) indicates whether the URC transitioned into the hibernation mode (*Result Code* set to **SUCCESS**) or failed (*Result Code* set to **FAILURE** or **FAILURE_HARDWARE**). If the URC successfully cancelled the Sleep command then the *Result Code* field will be set to **FAILURE_CANCELLED**. The URCD must retrieve this RCEB prior to idling the interface and especially transitioning the device out of the D0 state.

See Section 4.11 for additional operational requirements.

4.13.10.6 Set Application Specific IE Notification

This command (see Section 3.1.3.16) instructs the URC to notify the URCD when it detects an Application Specific IE in a beacon from a peer device. The URC must send an IE Received notification (see Section 4.13.11.1) when it detects the ASIE specified in this command. The URC must send the IE Received notification to the URCD even if the Application Specific IE was detected in a beacon slot that is currently being filtered by the URCD.

The *Device Address* field specifies the device address of the peer device whose beacon the URC must monitor for the Application Specific IE. The value in this field is only valid if the *Ignore Device Address* bit is set to zero. If the *Ignore Device Address* field is set to one then the URC must scan all beacons received for the Application Specific IE specified and send an IE Received notification for each beacon in which it detects the Application Specific IE.

The *ASIE Identifier* field identifies the Application Specific IE in peer devices' beacons that the URCD wants the URC to scan for. The URC must maintain all the ASIE Identifiers requested by the URCD with this command and send an IE Received notification when it receives an Application Specific IE which matches any of the ASIE Identifiers. Note that the URC uses this field in conjunction with the *Device Address* and the *Ignore Device Address* fields to determine whether or not to send an IE Received notification to the URCD.

If the URC receives a new Set Application Specific IE Notification command with a value in the *ASIE Identifier* field that matches the value of the *ASIE Identifier* field in a prior Set Application Specific IE Notification command then the URC must do one of the following:

- If the *Ignore Device Address* field is set to one then it must replace the existing information (*Device Address* and *Ignore Device Address*) with the newly specified values. If the URC has received more than one prior Set Application Specific IE Notification commands with the specified ASIE Identifier and valid device addresses, it must invalidate all of them and start scanning for the Application Specific IE from any device.
- If the *Ignore Device Address* field is set to zero and the value in the *Device Address* field matches the value of the *Device Address* field in any prior Set Application Specific IE Notification command then it must replace the existing information with the newly specified values.
- If the *Ignore Device Address* field is set to zero and the value in the *Device Address* field does not match the value of the *Device Address* field in any prior Set Application Specific IE Notification command then the URC must scan the peer device beacon whose device address matches that specified in this command in addition to the other peer devices that the URC is already scanning for the Application Specific IE.

If the *Enable/Disable* bit set to one when the URC receives this command, then it must start monitoring the beacons of peer devices for the Application Specific IE specified in the *ASIE Identifier* field. If the *Enable/Disable* bit is set to zero, then the URC must stop scanning for the Application Specific IE with the specified ASIE Identifier either from a specific device address or from any device address (depending on the *Ignore Device Address* bit field setting).

The URC will confirm that it received the Set Application Specific IE Notification command by sending back an RCEB. The *Result Code* field indicates whether the URC has successfully accepted the command or

not. If the URC cannot store any more ASIE Identifier, it must return an RCEB with the *Result Code* set to **FAILURE_NO_MORE_ASIE_NOTIFICATION**. If the URC receives a Set Application Specific IE Notification command with the *Enable/Disable* bit set to zero but scanning for the specified ASIE is not enabled, then the *Result Code* will be set to **FAILURE_INVALID_PARAMETER**.

4.13.11 Notifications

4.13.11.1 IE Received Notification

The URC will generate this notification (see Section 3.1.4.1) when it receives a beacon from another device including an IE with the *Target DevAddr* field set to the URC's own device address or an Application-specific IE specified in a Set Application Specific IE Notification command (see Section 4.13.10.6). The *Source Address* field in the notification must be set to the *SrcAddr* field in the MAC header of the received beacon. The *IE Length* field is set to the length of data placed in the *IE Data* field. The *IE Data* field contains the received IE including *Element ID* and *Length* fields.

A list of IEs that the URC must generate this notification for is:

- Probe IE
- Application-specific IE
- Application-specific Probe IE
- Relinquish Request IE

The URC must generate this notification for a Probe IE only when it contains at least one Element ID for an information element it cannot generate by itself.

Note that the DRP IEs that are targeted at this URC are not sent using this notification. See Section 4.13.8.3 for the proper handling of DRP notifications.

4.13.11.2 Beacon Received Notification

The URC will generate this notification (see Section 3.1.4.2) to inform the URCD that a beacon was received unless the beacon slot of the received beacon is filtered by a Set Beacon Filter command (see Section 4.13.10.2).

The *Beacon Information* field is an array including the MAC Header, Beacon Parameters, and all information elements for the received beacon. The length of the *Beacon Information* array is specified in the *Beacon Information Length* field.

The *Channel Number* field is the PHY channel on which the beacon was received.

The *Beacon Type* field indicates the type of the received beacon and must be set to an appropriate value according to the following rules:

- If the URC is not beaconing and performing a scan (with the scan state of **SCAN_ONLY** or **SCAN_ONLY_STARTTIME**), this field must be set to zero (**Scan**).
- If the URC is beaconing and performing a scan (with the scan state of **SCAN_OUTSIDE_BP** or **SCAN_WHILE_INACTIVE**) on a different channel from the channel on which the URC is sending its beacon, this field must be set to zero (**Scan**).
- If the URC is beaconing and the beacon is received on the channel on which the URC is sending its beacon, this field must be determined based on the BPST calculated from the reception time of the beacon.
    - If the received beacon's BPST aligns with the URC's own BPST, this beacon is considered a neighbor beacon and this field must be set to one (**Neighbor**).

- If the received beacon's BPST does not align with the URC's own BPST, this beacon is considered an alien beacon. If the alien BP overlaps with the URC's BP, this field must be set to two (**Overlapping Alien**), otherwise it must be set to three (**Non-Overlapping Alien**).

- If the URC is beaconing and the beacon is received in a signaling slot (i.e. the Signaling Slot bit in the Beacon Parameters field is set to one) then this field is undefined and must be ignored by the URCD.

The STC value at the time the beacon was received is returned in the *Receive Time* field. The value in this field can be used by the URCD to determine the actual BPST for this beacon group. The URCD can use the calculated BPST to subsequently join a beacon group. This must be a value between 0 and 65535.

The Link Quality Indication value associated with the received beacon frame is specified the *LQI* field. This is a value between 0 and 255.

The Received Signal Strength Indication value associated with the received beacon frame is returned in the *RSSI* field.

4.13.11.3 Beacon Size Change Notification

This notification (see Section 3.1.4.3) informs the URCD that the size of the beacon has changed due to a modification of one or more of the IEs handled by the URC.

4.13.11.4 BPOIE Change Notification

This notification (see Section 3.1.4.4) informs the URCD that the BPOIE that is being transmitted by the URC has changed.

4.13.11.5 BP Slot Change Notification

This notification (see Section 3.1.4.5) informs the URCD that the URC has changed the slot it is sending out its beacon in. This notification must be sent in the following cases:

- After sending a beacon the first time (after a successful Start Beaconing command)

- Every time the URC changes the slot it sends its beacon in due to contraction of the beacon period or beacon slot conflict resolution

The *Slot Number* field must be set to the new beacon slot number and the *No Slot* bit must be set to zero.

This notification is also used to inform the URCD that the URC cannot continue sending its beacons because it is unable to choose a new beacon slot after a beacon slot conflict. The URC must set the *No Slot* bit to one and the URCD must ignore the *Slot Number* field. The URC must transition to the "**Not Beaconing**" state (see Section 4.13.1) after sending this notification (with the *No Slot* bit set to one).

4.13.11.6 Channel Change IE Received Notification

This notification (see Section 3.1.4.12) informs the URCD that the URC detected a Channel Change IE in a beacon from a neighbor device.

4.13.11.7 Unknown Command or Command Type Received Notification

This notification (see Section 3.1.4.12) informs the URCD that the URC received an RCCB with a Command and/or Command Type that is was unable to decode. The *Event Context ID Number* field must be set to the *Command Context ID* field value in the RCCB with the unknown Command or Command Type. This notification can be sent to the URCD when the URC is any state.

4.14 URC Interrupts

For efficient operation of the UMC, the URC must be able to generate interrupts when various conditions occur as described in the following sections.

4.14.1 Ready for Command Interrupt

When the URC is ready to accept the next command from the URCD after it completes reading a radio control command block from the command buffer pointed to by the URCCMDADDR register and the *Interrupt when Ready* bit is set to a one in the URCCMD register, it sets the *Ready for Command Interrupt (URCRCI)* bit to a one in the URCSTS register. If the *Ready for Command Interrupt Enable* bit in the URCINTR register is a one, then the URC will issue a hardware interrupt. See Section 4.13 for the detailed operation of command processing.

4.14.2 Event Ready

The URC sets the *Event Ready (URCER)* bit to a one in the URCSTS register when it retires an event buffer. If the *Event Ready Enable* bit in the URCINTR register is a one, then the URC will issue a hardware interrupt. See Section 4.13 for the detailed operation of event processing.

4.14.3 Host System Error

The URC is a bus master and any interaction between the URC and the system may experience errors. The type of host error may be catastrophic to the URC (such as a Master Abort) making it impossible for the URC to continue in a coherent fashion. In the presence of non-catastrophic host errors, such as parity errors, the URC could potentially continue operation. The recommended behavior for these types of errors is to escalate it to a catastrophic error and halt the whole UMC including all the other interface functions exposed by the hardware. Host-based error must result in the following actions:

- The *Run/Stop* bit in the URCCMD register is set to a zero.
- The following bits in the URCSTS register are set:
  - *Host System Error* bit is to a one.
  - *Halted* bit is set to a one.
- If the *Host System Error Enable* bit in the URCINTR register is a one, then the URC will issue a hardware interrupt.

Note: After a *Host System Error*, software must reset the UMC via *UWBReset* in the URCCMD register before re-initializing and restarting the UMC.

4.14.4 Interrupt from Other Interface Functions

When a hardware interrupt is generated by any interface function exposed by the UMC, the URC sets the corresponding bit of the *Interrupt Source Identification (URCISI)* field to a one in the URCSTS register. If the corresponding bit of the *Interrupt Source Identification Enable* field is set to a one in the URCINTR register, then the URC will issue a hardware interrupt.

Each bit in the *URCISI* field is associated with the interface capability declared in the UWB Interface Capability registers (see Section 2.2). The ordering of the bits in this field is the same as the order of the capabilities in the UWBCAPDATA register array.

The URCD may handle interrupt from each interface function only when the corresponding bit in the *URCISI* field is set to a one.

This field is read only and the URC must reset each bit when the hardware interrupt signal from the source interface function is cleared.

Appendix A
URC PCI Power Management Interface

An advanced power management capabilities interface compliant with *PCI Bus Power Management Interface Specification Revision 1.1* is incorporated into this standard. This interface allows the URC and associated UWB Multi-Interface Controller (UMC) sub-functions to be placed in various power management states offering a variety of power savings for a host system.

The URC sub-function is the interface to the device-level PCI power management control. Table A-1 highlights the UMC function support for power management states and features supported for each of the power management states. A UMC function implementation may internally gate-off clocks and suspend logic functions in the URC, WHC and other interface sub-functions (low power consumption mode) to provide these power savings. The methods utilized by each UMC hardware vendor to achieve the required behavior are implementation specific. The URC sub-function will assert PME# and retain chip context in accordance with the rules defined in the *PCI Bus Power Management Interface Specification Revision 1.1* and this specification

The system software must prepare all sub-functions in a UMC implementation for low power operation before exiting the D0 state via command to the URC sub-function. Depending on the architecture and implementation of system software, this may require two or more device drivers to cooperate and possibly synchronize the preparation activities. This is to ensure all interface sub-functions are in an inactive, low-power mode prior to the command to exit the D0 state.

Table A-1. UMC Support for Power Management States

PCI Power Management State	State Required/ Optional by Spec	Comments
D0	Required	Fully awake. All logic in full power mode.
D1	Optional	UMC Sleep state with all sub-functions idled with bus master capabilities disabled. All logic in low latency power savings mode because of low latency returning to D0 state.
D2	Optional	UMC Sleep state with all sub-functions idled with bus master capabilities disabled.
D3hot	Required	Deep UMC Sleep state with all sub-functions idled with bus master capabilities disabled.
D3cold	Required	Fully asleep.

A.1 PCI Power Management Register Interface

UMC implementations follow the PCI Power Management register interface specified in the PCI Power Management Specification Rev 1.1. Specific requirements and clarifications for UMC implementations are:

- The device must be capable of asserting PME# when in any supported device state. However, if the device supports systems in which the PME# assertion from D3cold is not possible (i.e. insufficient or non-existent Auxiliary power), then the “PME_Support” bit for D3cold (bit 15 of the PMC Register) must be modifiable. Motherboard-down devices may use a software (BIOS) scheme for modifying the value reported in this read-only bit, while other devices may use a pin-strapping to determine the value that is reported.
- The Aux_Current or Data Register value reported by the device should represent the maximum current that the device including the entire set of associated device functions will consume. Note that if the

device has been configured to not generate PME# from D3cold, then Aux_Current field or Data Register (D3 Power Consumed, D3 Power Dissipated) must report “000”.

All other registers and field should follow the PCI PM specification.

A.1.1 Power State Transitions

The UMC enters the D0 power state from the D3cold power state when Vcc is applied and a hardware or software reset occurs. A software reset should not affect the PCI power management registers. The hardware reset may be either a PCI reset input or an optional power-on reset input.

Power management software transitions the UMC through D0, D1, D2, and D3hot power states via URC-owned PCI Power Management register accesses. Additional power management policy may be implemented to switch or continuously apply an auxiliary power supply, VAUX, to the URC, WHC, etc. sub-functions when Vcc is removed. While in this power state, referred to as D3cold, the UMC exhibits identical behavior as the D3hot power state (except that configuration space accesses are not supported) and no additional UMC hardware support is required to distinguish between D3hot and D3cold.

Per the PCI Power Management specification, the UMC asserts an internal reset during the D3hot to D0 transition. The UMC must retain all relevant wake context when transitioning from D3hot to D0 in order for system software to process a wake request. In PCI configuration space, this means that the *PMCSR.PME_Status* and *PMCSR.PME_En* bits must be maintained. Additionally, the *PMC.PME_Support(D3cold)* bit must be maintained.

Additionally, the UMC must retain function-specific context that meets any of the following criteria:

1. BIOS-configured registers that are programmed during system initialization
2. Context needed to avoid USB re-enumeration
3. Context needed for properly generating wake events
4. URC Event Buffer and WHC Notification Buffer for software to determine the source of a wake event

Specifically, the following URC and WHC resources must not be reset during the D3hot to D0 transition *and* must be maintained in the auxiliary power well (see Section A.1.2 below):

- URC Event Buffer
- WHC Security Key Store
- WHC Notification Buffer

Note that all of the resources mentioned above are only reset upon initial Aux power-up or software reset. Software must specifically clear any of these bits during subsequent initialization sequences, if desired.

A.1.2 Power State Definitions

This section defines the UMC behavior per power state when programmed using *PMCSR.PowerState.* Power management software may use alternate register mechanisms to place the UMC in similar states. The UMC must support the D0, D3hot, and D3cold power states and is recommended to support the D1, D2 power states.

Any wakeup events as specified in Table A-2 will set *PMCSR.PME_Status* when the UMC is programmed with *PMCSR.Power_State* set to D0, and a PCI PME# wake-up must be signaled if enabled via *PMCSR.PME_En.* It is possible for one interrupt event which is also a wakeup event to cause the UMC to signal both a PCI interrupt and a PME# to the host. Power management software shall either be designed to handle this condition or to mask the PME# signal when the UMC is in D0.

System software must place each sub-function in the UMC into an appropriate low-power mode, with bus master capabilities idled before it attempts to move the UMC out of the D0 power state.

All UMC contexts are retained in all power states except D3cold. For D3cold, the same context that is described in the previous section relative to the D3hot-to-D0 internal reset must be retained.

The functional and wake-up characteristics for the UMC power states are summarized in Table A-2.

Table A-2. UMC Power State Summary

Power State	Functional Characteristics	Wake-up Characteristics (Associated Enables must be Set)
D0	▪ Fully functional UMC device state ▪ Unmasked interrupts are fully functional	▪ Resume Detected on URC/WHC sub-function (this includes all events / device notifications that require the UMC to wake the system)
D1	▪ UMC must preserve PCI configuration ▪ UMC must preserve URC, WHC and other sub-function block configurations ▪ Hardware masks functional interrupts	▪ Resume Detected on URC/WHC sub-function (this includes all events / device notifications that require the UMC to wake the system)
D2	▪ UMC must preserve PCI configuration ▪ UMC must preserve URC, WHC and other sub-function block configurations ▪ Hardware masks functional interrupts	▪ Resume Detected on URC/WHC sub-function (this includes all events / device notifications that require the UMC to wake the system)
D3hot	▪ UMC must preserve PCI configuration ▪ UMC must preserve URC, WHC and other sub-function block configurations ▪ Hardware masks functional interrupts	▪ Resume Detected on URC/WHC sub-function (this includes all events / device notifications that require the UMC to wake the system)
D3cold	▪ PME Context in PCI Configuration space is preserved ▪ Wake Context in URC, WHC and other appropriate sub-function Memory Spaces is preserved	▪ Resume Detected on URC/WHC sub-function (this includes all events / device notifications that require the UMC to wake the system)

A.1.3 PCI PME# Signal

The PCI PME# signal must be implemented as an open drain, active low signal that is driven low by the UMC to request a change in its current power management state. PME# has additional electrical requirements over and above standard open drain signals that allow it to be shared between devices that are powered off and those which are powered on. Refer to the PCI Power Management specification for more details.