TMS320DM644x DMSoC
Universal Serial Bus (USB) Controller

User's Guide

Literature Number: SPRUE35G
June 2010

Preface **12**
1 Introduction **14**
1.1 Purpose of the Peripheral 14
1.2 Features 14
1.3 Features Not Supported 15
1.4 Functional Block Diagram 15
1.5 Supported Use Case Examples 16
1.6 Industry Standard(s) Compliance Statement 22
2 Peripheral Architecture **23**
2.1 Clock Control 23
2.2 Signal Descriptions 23
2.3 Indexed and Non-Indexed Registers 23
2.4 USB PHY Initialization 24
2.5 Dynamic FIFO Sizing 24
3 USB Controller Host and Peripheral Modes Operation **25**
3.1 USB Controller Peripheral Mode Operation 27
3.2 USB Controller Host Mode Operation 45
3.3 DMA Operation 61
3.4 Interrupt Handling 73
3.5 Test Modes 75
3.6 Reset Considerations 77
3.7 Interrupt Support 77
3.8 EDMA Event Support 77
3.9 Power Management 77
4 Registers **78**
4.1 Control Register (CTRLR) 85
4.2 Status Register (STATR) 86
4.3 RNDIS Register (RNDISR) 86
4.4 Auto Request Register (AUTOREQ) 87
4.5 USB Interrupt Source Register (INTSRCR) 88
4.6 USB Interrupt Source Set Register (INTSETR) 89
4.7 USB Interrupt Source Clear Register (INTCLRR) 90
4.8 USB Interrupt Mask Register (INTMSKR) 91
4.9 USB Interrupt Mask Set Register (INTMSKSETR) 92
4.10 USB Interrupt Mask Clear Register (INTMSKCLRR) 93
4.11 USB Interrupt Source Masked Register (INTMASKEDR) 94
4.12 USB End of Interrupt Register (EOIR) 95
4.13 Transmit CPPI Control Register (TCPPICR) 96
4.14 Transmit CPPI Teardown Register (TCPPITDR) 96
4.15 CPPI DMA End of Interrupt Register (CPPIEOIR) 97
4.16 Transmit CPPI Masked Status Register (TCPPIMSKSR) 98
4.17 Transmit CPPI Raw Status Register (TCPPIRAWSR) 98

4.18 Transmit CPPI Interrupt Enable Set Register (TCPPIIENSETR) 99
4.19 Transmit CPPI Interrupt Enable Clear Register (TCPPIIENCLRR) 99
4.20 Receive CPPI Control Register (RCPPICR) 100
4.21 Receive CPPI Masked Status Register (RCPPIMSKSR) 100
4.22 Receive CPPI Raw Status Register (RCPPIRAWSR) 101
4.23 Receive CPPI Interrupt Enable Set Register (RCPPIENSETR) 101
4.24 Receive CPPI Interrupt Enable Clear Register (RCPPIIENCLRR) 102
4.25 Receive Buffer Count 0 Register (RBUFCNT0) 102
4.26 Receive Buffer Count 1 Register (RBUFCNT1) 103
4.27 Receive Buffer Count 2 Register (RBUFCNT2) 103
4.28 Receive Buffer Count 3 Register (RBUFCNT3) 104
4.29 Transmit CPPI DMA State Word 0 (TCPPIDMASTATEW0) 104
4.30 Transmit CPPI DMA State Word 1 (TCPPIDMASTATEW1) 105
4.31 Transmit CPPI DMA State Word 2 (TCPPIDMASTATEW2) 105
4.32 Transmit CPPI DMA State Word 3 (TCPPIDMASTATEW3) 106
4.33 Transmit CPPI DMA State Word 4 (TCPPIDMASTATEW4) 106
4.34 Transmit CPPI DMA State Word 5 (TCPPIDMASTATEW5) 107
4.35 Transmit CPPI Completion Pointer (TCPPICOMPPTR) 107
4.36 Receive CPPI DMA State Word 0 (RCPPIDMASTATEW0) 108
4.37 Receive CPPI DMA State Word 1 (RCPPIDMASTATEW1) 108
4.38 Receive CPPI DMA State Word 2 (RCPPIDMASTATEW2) 109
4.39 Receive CPPI DMA State Word 3 (RCPPIDMASTATEW3) 109
4.40 Receive CPPI DMA State Word 4 (RCPPIDMASTATEW4) 110
4.41 Receive CPPI DMA State Word 5 (RCPPIDMASTATEW5) 110
4.42 Receive CPPI DMA State Word 6 (RCPPIDMASTATEW6) 111
4.43 Receive CPPI Completion Pointer (RCPPICOMPPTR) 112
4.44 Function Address Register (FADDR) 112
4.45 Power Management Register (POWER) 113
4.46 Interrupt Register for Endpoint 0 Plus Transmit Endpoints 1 to 4 (INTRTX) 114
4.47 Interrupt Register for Receive Endpoints 1 to 4 (INTRRX) 115
4.48 Interrupt Enable Register for INTRTX (INTRTXE) 116
4.49 Interrupt Enable Register for INTRRX (INTRRXE) 116
4.50 Interrupt Register for Common USB Interrupts (INTRUSB) 117
4.51 Interrupt Enable Register for INTRUSB (INTRUSBE) 118
4.52 Frame Number Register (FRAME) 119
4.53 Index Register for Selecting the Endpoint Status and Control Registers (INDEX) 119
4.54 Register to Enable the USB 2.0 Test Modes (TESTMODE) 120
4.55 Maximum Packet Size for Peripheral/Host Transmit Endpoint (TXMAXP) 121
4.56 Control Status Register for Endpoint 0 in Peripheral Mode (PERI_CSR0) 122
4.57 Control Status Register for Endpoint 0 in Host Mode (HOST_CSR0) 123
4.58 Control Status Register for Peripheral Transmit Endpoint (PERI_TXCSR) 124
4.59 Control Status Register for Host Transmit Endpoint (HOST_TXCSR) 125
4.60 Maximum Packet Size for Peripheral Host Receive Endpoint (RXMAXP) 126
4.61 Control Status Register for Peripheral Receive Endpoint (PERI_RXCSR) 127
4.62 Control Status Register for Host Receive Endpoint (HOST_RXCSR) 128
4.63 Count 0 Register (COUNT0) 129
4.64 Receive Count Register (RXCOUNT) 129
4.65 Type Register (Host mode only) (HOST_TYPE0) 130
4.66 Transmit Type Register (Host mode only) (HOST_TXTYPE) 130

4.67 NAKLimit0 Register (Host mode only) (HOST_NAKLIMIT0) 131
4.68 Transmit Interval Register (Host mode only) (HOST_TXINTERVAL) 131
4.69 Receive Type Register (Host mode only) (HOST_RXTYPE) 132
4.70 Receive Interval Register (Host mode only) (HOST_RXINTERVAL) 133
4.71 Configuration Data Register (CONFIGDATA) 134
4.72 Transmit and Receive FIFO Register for Endpoint 0 (FIFO0) 135
4.73 Transmit and Receive FIFO Register for Endpoint 1 (FIFO1) 136
4.74 Transmit and Receive FIFO Register for Endpoint 2 (FIFO2) 136
4.75 Transmit and Receive FIFO Register for Endpoint 3 (FIFO3) 137
4.76 Transmit and Receive FIFO Register for Endpoint 4 (FIFO4) 137
4.77 OTG Device Control Register (DEVCTL) 138
4.78 Transmit Endpoint FIFO Size (TXFIFOSZ) 139
4.79 Receive Endpoint FIFO Size (RXFIFOSZ) 139
4.80 Transmit Endpoint FIFO Address (TXFIFOADDR) 140
4.81 Receive Endpoint FIFO Address (RXFIFOADDR) 140
4.82 Transmit Function Address (TXFUNCADDR) 141
4.83 Transmit Hub Address (TXHUBADDR) 141
4.84 Transmit Hub Port (TXHUBPORT) 141
4.85 Receive Function Address (RXFUNCADDR) 142
4.86 Receive Hub Address (RXHUBADDR) 142
4.87 Receive Hub Port (RXHUBPORT) 142
Appendix A Revision History 143

List of Figures

1 Functional Block Diagram 15
2 Interrupt Service Routine Flow Chart 26
3 CPU Actions at Transfer Phases 30
4 Sequence of Transfer 31
5 Service Endpoint 0 Flow Chart 33
6 IDLE Mode Flow Chart 34
7 TX Mode Flow Chart 35
8 RX Mode Flow Chart 36
9 Setup Phase of a Control Transaction Flow Chart 47
10 IN Data Phase Flow Chart 49
11 OUT Data Phase Flow Chart 51
12 Completion of SETUP or OUT Data Phase Flow Chart 53
13 Completion of IN Data Phase Flow Chart 55
14 Tx Queue Flow Chart 64
15 Rx Queue Flow Chart 69
16 Control Register (CTRLR) 85
17 Status Register (STATR) 86
18 RNDIS Register (RNDISR) 86
19 Auto Request Register (AUTOREQ) 87
20 USB Interrupt Source Register (INTSRCR) 88
21 USB Interrupt Source Set Register (INTSETR) 89
22 USB Interrupt Source Clear Register (INTCLRR) 90
23 USB Interrupt Mask Register (INTMSKR) 91
24 USB Interrupt Mask Set Register (INTMSKSETR) 92
25 USB Interrupt Mask Clear Register (INTMSKCLRR) 93
26 USB Interrupt Source Masked Register (INTMASKEDR) 94
27 USB End of Interrupt Register (EOIR) 95
28 Transmit CPPI Control Register (TCPPICR) 96
29 Transmit CPPI Teardown Register (TCPPITDR) 96
30 CPPI DMA End of Interrupt Register (CPPIEOIR) 97
31 Transmit CPPI Masked Status Register (TCPPIMSKSR) 98
32 Transmit CPPI Raw Status Register (TCPPIRAWSR) 98
33 Transmit CPPI Interrupt Enable Set Register (TCPPIIENSETR) 99
34 Transmit CPPI Interrupt Enable Clear Register (TCPPIIENCLRR) 99
35 Receive CPPI Control Register (RCPPICR) 100
36 Receive CPPI Masked Status Register (RCPPIMSKSR) 100
37 Receive CPPI Raw Status Register (RCPPIRAWSR) 101
38 Receive CPPI Interrupt Enable Set Register (RCPPIENSETR) 101
39 Receive CPPI Interrupt Enable Clear Register (RCPPIIENCLRR) 102
40 Receive Buffer Count 0 Register (RBUFCNT0) 102
41 Receive Buffer Count 1 Register (RBUFCNT1) 103
42 Receive Buffer Count 2 Register (RBUFCNT2) 103
43 Receive Buffer Count 3 Register (RBUFCNT3) 104
44 Transmit CPPI DMA State Word 0 (TCPPIDMASTATEW0) 104
45 Transmit CPPI DMA State Word 1 (TCPPIDMASTATEW1) 105
46 Transmit CPPI DMA State Word 2 (TCPPIDMASTATEW2) 105
47 Transmit CPPI DMA State Word 3 (TCPPIDMASTATEW3) 106

48 Transmit CPPI DMA State Word 4 (TCPPIDMASTATEW4) 106
49 Transmit CPPI DMA State Word 5 (TCPPIDMASTATEW5) 107
50 Transmit CPPI Completion Pointer (TCPPICOMPPTR) 107
51 Receive CPPI DMA State Word 0 (RCPPIDMASTATEW0) 108
52 Receive CPPI DMA State Word 1 (RCPPIDMASTATEW1) 108
53 Receive CPPI DMA State Word 2 (RCPPIDMASTATEW2) 109
54 Receive CPPI DMA State Word 3 (RCPPIDMASTATEW3) 109
55 Receive CPPI DMA State Word 4 (RCPPIDMASTATEW4) 110
56 Receive CPPI DMA State Word 5 (RCPPIDMASTATEW5) 110
57 Receive CPPI DMA State Word 6 (RCPPIDMASTATEW6) 111
58 Receive CPPI Completion Pointer (RCPPICOMPPTR) 112
59 Function Address Register (FADDR) 112
60 Power Management Register (POWER) 113
61 Interrupt Register for Endpoint 0 Plus Tx Endpoints 1 to 4 (INTRTX) 114
62 Interrupt Register for Receive Endpoints 1 to 4 (INTRRX) 115
63 Interrupt Enable Register for INTRTX (INTRTXE) 116
64 Interrupt Enable Register for INTRRX (INTRRXE) 116
65 Interrupt Register for Common USB Interrupts (INTRUSB) 117
66 Interrupt Enable Register for INTRUSB (INTRUSBE) 118
67 Frame Number Register (FRAME) 119
68 Index Register for Selecting the Endpoint Status and Control Registers (INDEX) 119
69 Register to Enable the USB 2.0 Test Modes (TESTMODE) 120
70 Maximum Packet Size for Peripheral/Host Transmit Endpoint (TXMAXP) 121
71 Control Status Register for Endpoint 0 in Peripheral Mode (PERI_CSR0) 122
72 Control Status Register for Endpoint 0 in Host Mode (HOST_CSR0) 123
73 Control Status Register for Peripheral Transmit Endpoint (PERI_TXCSR) 124
74 Control Status Register for Host Transmit Endpoint (HOST_TXCSR) 125
75 Maximum Packet Size for Peripheral Host Receive Endpoint (RXMAXP) 126
76 Control Status Register for Peripheral Receive Endpoint (PERI_RXCSR) 127
77 Control Status Register for Host Receive Endpoint (HOST_RXCSR) 128
78 Count 0 Register (COUNT0) 129
79 Receive Count Register (RXCOUNT) 129
80 Type Register (Host mode only) (HOST_TYPE0) 130
81 Transmit Type Register (Host mode only) (HOST_TXTYPE) 130
82 NAKLimit0 Register (Host mode only) (HOST_NAKLIMIT0) 131
83 Transmit Interval Register (Host mode only) (HOST_TXINTERVAL) 131
84 Receive Type Register (Host mode only) (HOST_RXTYPE) 132
85 Receive Interval Register (Host mode only) (HOST_RXINTERVAL) 133
86 Configuration Data Register (CONFIGDATA) 134
87 Transmit and Receive FIFO Register for Endpoint 0 (FIFO0) 135
88 Transmit and Receive FIFO Register for Endpoint 1 (FIFO1) 136
89 Transmit and Receive FIFO Register for Endpoint 2 (FIFO2) 136
90 Transmit and Receive FIFO Register for Endpoint 3 (FIFO3) 137
91 Transmit and Receive FIFO Register for Endpoint 4 (FIFO4) 137
92 OTG Device Control Register (DEVCTL) 138
93 Transmit Endpoint FIFO Size (TXFIFOSZ) 139
94 Receive Endpoint FIFO Size (RXFIFOSZ) 139
95 Transmit Endpoint FIFO Address (TXFIFOADDR) 140
96 Receive Endpoint FIFO Address (RXFIFOADDR) 140

97 Transmit Function Address (TXFUNCADDR) 141
98 Transmit Hub Address (TXHUBADDR) 141
99 Transmit Hub Port (TXHUBPORT) 141
100 Receive Function Address (RXFUNCADDR) 142
101 Receive Hub Address (RXHUBADDR) 142
102 Receive Hub Port (RXHUBPORT) 142

List of Tables

1 USB Pins 23
2 PERI_TXCSR Register Bit Configuration for Bulk IN Transactions 38
3 PERI_RXCSR Register Bit Configuration for Bulk OUT Transactions 40
4 PERI_TXCSR Register Bit Configuration for Isochronous IN Transactions 42
5 PERI_RXCSR Register Bit Configuration for Isochronous OUT Transactions 44
6 Transmit Buffer Descriptor Word 0 62
7 Transmit Buffer Descriptor Word 1 62
8 Transmit Buffer Descriptor Word 2 62
9 Transmit Buffer Descriptor Word 3 62
10 Receive Buffer Descriptor Word 0 67
11 Receive Buffer Descriptor Word 1 67
12 Receive Buffer Descriptor Word 2 67
13 Receive Buffer Descriptor Word 3 68
14 Interrupts Generated by the USB Controller 73
15 USB Interrupt Conditions 73
16 Universal Serial Bus (USB) Registers 78
17 Control Register (CTRLR) Field Descriptions 85
18 Status Register (STATR) Field Descriptions 86
19 RNDIS Register (RNDISR) Field Descriptions 86
20 Auto Request Register (AUTOREQ) Field Descriptions 87
21 USB Interrupt Source Register (INTSRCR) Field Descriptions 88
22 USB Interrupt Source Set Register (INTSETR) Field Descriptions 89
23 USB Interrupt Source Clear Register (INTCLRR) Field Descriptions 90
24 USB Interrupt Mask Register (INTMSKR) Field Descriptions 91
25 USB Interrupt Mask Set Register (INTMSKSETR) Field Descriptions 92
26 USB Interrupt Mask Clear Register (INTMSKCLRR) Field Descriptions 93
27 USB Interrupt Source Masked Register (INTMASKEDR) Field Descriptions 94
28 USB End of Interrupt Register (EOIR) Field Descriptions 95
29 Transmit CPPI Control Register (TCPPICR) Field Descriptions 96
30 Transmit CPPI Teardown Register (TCPPITDR) Field Descriptions 96
31 CPPI DMA End of Interrupt Register (CPPIEOIR) Field Descriptions 97
32 Transmit CPPI Masked Status Register (TCPPIMSKSR) Field Descriptions 98
33 Transmit CPPI Raw Status Register (TCPPIRAWSR) Field Descriptions 98
34 Transmit CPPI Interrupt Enable Set Register (TCPPIIENSETR) Field Descriptions 99
35 Transmit CPPI Interrupt Enable Clear Register (TCPPIIENCLRR) Field Descriptions 99
36 Receive CPPI Control Register (RCPPICR) Field Descriptions 100
37 Receive CPPI Masked Status Register (RCPPIMSKSR) Field Descriptions 100
38 Receive CPPI Raw Status Register (RCPPIRAWSR) Field Descriptions 101
39 Receive CPPI Interrupt Enable Set Register (RCPPIENSETR) Field Descriptions 101
40 Receive CPPI Interrupt Enable Clear Register (RCPPIIENCLRR) Field Descriptions 102
41 Receive Buffer Count 0 Register (RBUFCNT0) Field Descriptions 102
42 Receive Buffer Count 1 Register (RBUFCNT1) Field Descriptions 103
43 Receive Buffer Count 2 Register (RBUFCNT2) Field Descriptions 103
44 Receive Buffer Count 3 Register (RBUFCNT3) Field Descriptions 104
45 Transmit CPPI DMA State Word 0 (TCPPIDMASTATEW0) Field Descriptions 104
46 Transmit CPPI DMA State Word 1 (TCPPIDMASTATEW1) Field Descriptions 105
47 Transmit CPPI DMA State Word 2 (TCPPIDMASTATEW2) Field Descriptions 105

48 Transmit CPPI DMA State Word 3 (TCPPIDMASTATEW3) Field Descriptions 106
49 Transmit CPPI DMA State Word 4 (TCPPIDMASTATEW4) Field Descriptions 106
50 Transmit CPPI DMA State Word 5 (TCPPIDMASTATEW5) Field Descriptions 107
51 Transmit CPPI Completion Pointer (TCPPICOMPPTR) Field Descriptions 107
52 Receive CPPI DMA State Word 0 (RCPPIDMASTATEW0) Field Descriptions 108
53 Receive CPPI DMA State Word 1 (RCPPIDMASTATEW1) Field Descriptions 108
54 Receive CPPI DMA State Word 2 (RCPPIDMASTATEW2) Field Descriptions 109
55 Receive CPPI DMA State Word 3 (RCPPIDMASTATEW3) Field Descriptions 109
56 Receive CPPI DMA State Word 4 (RCPPIDMASTATEW4) Field Descriptions 110
57 Receive CPPI DMA State Word 5 (RCPPIDMASTATEW5) Field Descriptions 110
58 Receive CPPI DMA State Word 6 (RCPPIDMASTATEW6) Field Descriptions 111
59 Receive CPPI Completion Pointer (RCPPICOMPPTR) Field Descriptions 112
60 Function Address Register (FADDR) Field Descriptions 112
61 Power Management Register (POWER) Field Descriptions 113
62 Interrupt Register for Endpoint 0 Plus Transmit Endpoints 1 to 4 (INTRTX)Field Descriptions 114
63 Interrupt Register for Receive Endpoints 1 to 4 (INTRRX) Field Descriptions 115
64 Interrupt Enable Register for INTRTX (INTRTXE) Field Descriptions 116
65 Interrupt Enable Register for INTRRX (INTRRXE) Field Descriptions 116
66 Interrupt Register for Common USB Interrupts (INTRUSB) Field Descriptions 117
67 Interrupt Enable Register for INTRUSB (INTRUSBE) Field Descriptions 118
68 Frame Number Register (FRAME) Field Descriptions 119
69 Index Register for Selecting the Endpoint Status and Control Registers (INDEX)Field Descriptions 119
70 Register to Enable the USB 2.0 Test Modes (TESTMODE) Field Descriptions 120
71 Maximum Packet Size for Peripheral/Host Transmit Endpoint (TXMAXP) Field Descriptions 121
72 Control Status Register for Endpoint 0 in Peripheral Mode (PERI_CSR0) Field Descriptions 122
73 Control Status Register for Endpoint 0 in Host Mode (HOST_CSR0) Field Descriptions 123
74 Control Status Register for Peripheral Transmit Endpoint (PERI_TXCSR) Field Descriptions 124
75 Control Status Register for Host Transmit Endpoint (HOST_TXCSR) Field Descriptions 125
76 Maximum Packet Size for Peripheral Host Receive Endpoint (RXMAXP) Field Descriptions 126
77 Control Status Register for Peripheral Receive Endpoint (PERI_RXCSR) Field Descriptions 127
78 Control Status Register for Host Receive Endpoint (HOST_RXCSR) Field Descriptions 128
79 Count 0 Register (COUNT0) Field Descriptions 129
80 Receive Count Register (RXCOUNT) Field Descriptions 129
81 Type Register (Host mode only) (HOST_TYPE0) Field Descriptions 130
82 Transmit Type Register (Host mode only) (HOST_TXTYPE) Field Descriptions 130
83 NAKLimit0 Register (Host mode only) (HOST_NAKLIMIT0) Field Descriptions 131
84 Transmit Interval Register (Host mode only) (HOST_TXINTERVAL) Field Descriptions 131
85 Receive Type Register (Host mode only) (HOST_RXTYPE) Field Descriptions 132
86 Receive Interval Register (Host mode only) (HOST_RXINTERVAL) Field Descriptions 133
87 Configuration Data Register (CONFIGDATA) Field Descriptions 134
88 Transmit and Receive FIFO Register for Endpoint 0 (FIFO0) Field Descriptions 135
89 Transmit and Receive FIFO Register for Endpoint 1 (FIFO1) Field Descriptions 136
90 Transmit and Receive FIFO Register for Endpoint 2 (FIFO2) Field Descriptions 136
91 Transmit and Receive FIFO Register for Endpoint 3 (FIFO3) Field Descriptions 137
92 Transmit and Receive FIFO Register for Endpoint 4 (FIFO4) Field Descriptions 137
93 OTG Device Control Register (DEVCTL) Field Descriptions 138
94 Transmit Endpoint FIFO Size (TXFIFOSZ) Field Descriptions 139
95 Receive Endpoint FIFO Size (RXFIFOSZ) Field Descriptions 139
96 Transmit Endpoint FIFO Address (TXFIFOADDR) Field Descriptions 140

97 Receive Endpoint FIFO Address (RXFIFOADDR) Field Descriptions 140
98 Transmit Function Address (TXFUNCADDR) Field Descriptions 141
99 Transmit Hub Address (TXHUBADDR) Field Descriptions 141
100 Transmit Hub Port (TXHUBPORT) Field Descriptions 141
101 Receive Function Address (RXFUNCADDR) Field Descriptions 142
102 Receive Hub Address (RXHUBADDR) Field Descriptions 142
103 Receive Hub Port (RXHUBPORT) Field Descriptions 142
104 Document Revision History 143

Read This First

About This Manual

This document describes the universal serial bus (USB) controller in the TMS320DM644x Digital Media System-on-Chip (DMSoC).

NOTE: References to On-The-Go (OTG) capability in this document are not supported on the DM644x devices. For specific issues or limitations on device behavior, see the silicon errata documentation.

Notational Conventions

This document uses the following conventions.

- Hexadecimal numbers are shown with the suffix h. For example, the following number is 40 hexadecimal (decimal 64): 40h.
- Registers in this document are shown in figures and described in tables.
  - Each register figure shows a rectangle divided into fields that represent the fields of the register. Each field is labeled with its bit name, its beginning and ending bit numbers above, and its read/write properties below. A legend explains the notation used for the properties.
  - Reserved bits in a register figure designate a bit that is used for future device expansion.

Related Documentation From Texas Instruments

The following documents describe the TMS320DM644x Digital Media System-on-Chip (DMSoC). Copies of these documents are available on the Internet at www.ti.com. *Tip:* Enter the literature number in the search box provided at www.ti.com.

The current documentation that describes the DM644x DMSoC, related peripherals, and other technical collateral, is available in the C6000 DSP product folder at: www.ti.com/c6000.

SPRUE14 — *TMS320DM644x DMSoC ARM Subsystem Reference Guide.* Describes the ARM subsystem in the TMS320DM644x Digital Media System-on-Chip (DMSoC). The ARM subsystem is designed to give the ARM926EJ-S (ARM9) master control of the device. In general, the ARM is responsible for configuration and control of the device; including the DSP subsystem, the video processing subsystem, and a majority of the peripherals and external memories.

SPRUE15 — *TMS320DM644x DMSoC DSP Subsystem Reference Guide.* Describes the digital signal processor (DSP) subsystem in the TMS320DM644x Digital Media System-on-Chip (DMSoC).

SPRUE19 — *TMS320DM644x DMSoC Peripherals Overview Reference Guide.* Provides an overview and briefly describes the peripherals available on the TMS320DM644x Digital Media System-on-Chip (DMSoC).

SPRAA84 — *TMS320C64x to TMS320C64x+ CPU Migration Guide.* Describes migrating from the Texas Instruments TMS320C64x digital signal processor (DSP) to the TMS320C64x+ DSP. The objective of this document is to indicate differences between the two cores. Functionality in the devices that is identical is not included.

SPRU732 — *TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide.* Describes the CPU architecture, pipeline, instruction set, and interrupts for the TMS320C64x and TMS320C64x+ digital signal processors (DSPs) of the TMS320C6000 DSP family. The C64x/C64x+ DSP generation comprises fixed-point devices in the C6000 DSP platform. The C64x+ DSP is an enhancement of the C64x DSP with added functionality and an expanded instruction set.

SPRU871 — *TMS320C64x+ DSP Megamodule Reference Guide.* Describes the TMS320C64x+ digital signal processor (DSP) megamodule. Included is a discussion on the internal direct memory access (IDMA) controller, the interrupt controller, the power-down controller, memory protection, bandwidth management, and the memory and cache.

SPRAAA6 — *EDMA v3.0 (EDMA3) Migration Guide for TMS320DM644x DMSoC.* Describes migrating from the Texas Instruments TMS320C64x digital signal processor (DSP) enhanced direct memory access (EDMA2) to the TMS320DM644x Digital Media System-on-Chip (DMSoC) EDMA3. This document summarizes the key differences between the EDMA3 and the EDMA2 and provides guidance for migrating from EDMA2 to EDMA3.

Universal Serial Bus (USB) Controller

1 Introduction

This document describes the universal serial bus (USB) controller in the TMS320DM644x Digital Media System-on-Chip (DMSoC). The controller supports high-speed USB peripheral mode and high-speed limited host-mode operations. The USB controller can be operated by ARM through the memory-mapped registers.

NOTE: References to On-The-Go (OTG) capability in this document are not supported on the DM644x devices. For specific issues or limitations on device behavior, see the silicon errata documentation.

The High-Speed USB OTG Controller is an instantiation of the MUSBMHDRC from Mentor Graphics Corporation.

1.1 Purpose of the Peripheral

The USB controller supports data throughput rates up to 480 Mbps. It provides a mechanism for data transfer between USB devices and also supports host negotiation.

1.2 Features

The USB has the following features:

- Supports USB 2.0 peripheral at High Speed (480 Mbps) and Full Speed (12 Mbps)
- Supports USB 2.0 host at High Speed (480 Mbps), Full Speed (12 Mbps), and Low Speed (1.5 Mbps)
- Supports four simultaneous RX and TX endpoints, more can be supported by dynamically switching
- Each endpoint can support all transfer types (control, bulk, interrupt, and isochronous)
- Supports USB extensions for Session Request (SRP) and Host Negotiation (HNP)
- Includes a 4K endpoint FIFO RAM, and supports programmable FIFO sizes
- External 5V power supply for VBUS can be controlled through I2C
- Includes a DMA controller that supports four TX and four RX DMA channels
- Includes RNDIS mode of DMA for accelerating RNDIS type protocols using short packet termination over USB

1.3 Features Not Supported

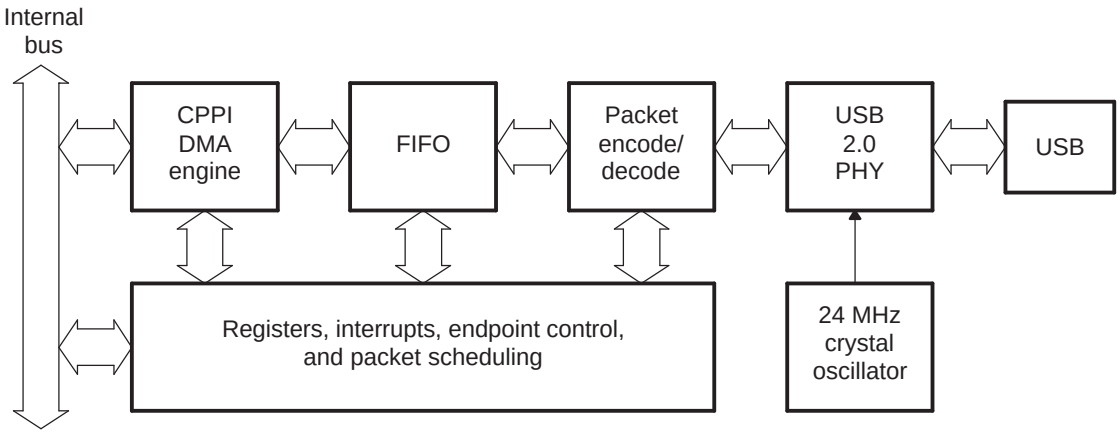
The following features are not supported:

- High Bandwidth Isochronous Transfer.
- High Bandwidth Interrupt Transfer.
- Automatic Amalgamation of Bulk Packets (CPPI DMA will indirectly handle this feature and is not supported at the core level).
- Automatic Splitting of Bulk Packets (CPPI DMA will indirectly handle this feature and is not supported at the core level).

1.4 Functional Block Diagram

The USB functional block diagram is shown in Figure 1.

Figure 1. Functional Block Diagram



1.5 Supported Use Case Examples

The USB supports the following user cases.

Detailed information about the architecture and operation of the USB controller follows in Section 2. Programming examples are also provided for each of the operational modes of the controller.

User Case 1: Example 1 shows an example of how to initialize the USB controller.

Example 1. Initializing the USB Controller

```
 // Routine to initialize USB controller
void usb_init()
{
    // local loop variable
    int I;

    // VBUS must be controlled externally. The following routine
    // should perform whatever actions are required (if any) to
    // turn off VBUS in the system.
    vbus_off();

    // Reset the USB controller
    usbRegs->CTRLR = 0x00000001;

    // Power on PHY and oscillator by clearing bits 2, 1, and 0 of USBPHY_CTL
    sysRegs->USB_PHY_CTRL = 0x000000D0;

    //Clear all pending interrupts
    usbRegs->INTCLRR = usbRegs->INTSRCR;

    // Initialize CPPI DMA
    usbRegs->RCPPICR = 0;    //Disable the RX DMA
    usbRegs->TCPPICR = 0;    //Disable the TX DMA

    // Initialize CPPI DMA state
    for(I = 0; i<4; I++) {
        usbRegs->CHANNEL[i].TCPPIDMASTATEW0 = 0;
        usbRegs->CHANNEL[i].TCPPIDMASTATEW1 = 0;
        usbRegs->CHANNEL[i].TCPPIDMASTATEW2 = 0;
        usbRegs->CHANNEL[i].TCPPIDMASTATEW3 = 0;
        usbRegs->CHANNEL[i].TCPPIDMASTATEW4 = 0;
        usbRegs->CHANNEL[i].TCPPIDMASTATEW5 = 0;
        usbRegs->CHANNEL[i].RCPPIDMASTATEW0 = 0;
        usbRegs->CHANNEL[i].RCPPIDMASTATEW1 = 0;
        usbRegs->CHANNEL[i].RCPPIDMASTATEW2 = 0;
        usbRegs->CHANNEL[i].RCPPIDMASTATEW3 = 0;
        usbRegs->CHANNEL[i].RCPPIDMASTATEW4 = 0;
        usbRegs->CHANNEL[i].RCPPIDMASTATEW5 = 0;
        usbRegs->CHANNEL[i].RCPPIDMASTATEW6 = 0;
    }
}
```

User Case 2: Example 2 shows an example of how to program the USB Endpoints in peripheral mode.

Example 2. Programming the USB Endpoints in Peripheral Mode

```
// DMA channel number.  Valid values are 0, 1, 2, or 3.
int CHAN_NUM = 0;

// Fifo sizes: uncomment the desired size.
// This example uses 64-byte fifo.
// int fifosize = 0;    // 8 bytes
// int fifosize = 1;    // 16 bytes
// int fifosize = 2;    // 32 bytes
   int fifosize = 3;    // 64 bytes
// int fifosize = 4;    // 128 bytes
// int fifosize = 5;    // 256 bytes
// int fifosize = 6;    // 512 bytes
// int fifosize = 7;    // 1024 bytes
// int fifosize = 8;    // 2048 bytes
// int fifosize = 9;    // 4096 bytes

// FIFO address.  Leave 64-bytes for endpoint 0.
int fifo_start_address = 8;

// Uncomment the desired buffering.  If double-buffer is selected, actual
// FIFO space will be twice the value listed above for fifosize.
// This example uses single buffer.
   int double_buffer = 0;  // Single-buffer
// int double_buffer = 1;  // Double-buffer

// For maximum packet size this formula will usually work, but it can also be
// set to another value if needed.  If non power of 2 value is needed (such as
// 1023) set it explicitly.
#define FIFO_MAXP 8*(1<<fifosize);

// Set the following variable to the device address.
int device_address = 0;


// The following code should be run after receiving a USB reset from the host.


// Initialize the endpoint FIFO.  RX and TX will be allocated the same sizes.
usbRegs->INDEX = CHAN_NUM+1;
usbRegs->RXFIFOSZ = fifosize | ((double_buffer & 1)<<4);
usbRegs->RXFIFOADDR = fifo_start_address;
usbRegs->TXFIFOSZ =    fifosize | ((double_buffer & 1)<<4);
usbRegs->TXFIFOADDR = fifo_start_address + (1<<(fifosize+double_buffer));
usbRegs->RXMAXP = FIFO_MAXP;
usbRegs->TXMAXP = FIFO_MAXP;

// Force Data Toggle is optional for interrupt traffic.  Uncomment if needed.
// CSL_FINS(usbRegs->PERI_TXCSR,USB_PERI_TXCSR_FRCDATATOG,1);

// Uncomment below to configure the endpoint for ISO and not respond with a
// handshake packet.
// CSL_FINS(usbRegs->PERI_RXCSR,USB_PERI_RXCSR_ISO,1);
// CSL_FINS(usbRegs->PERI_TXCSR,USB_PERI_TXCSR_ISO,1);

// After receiving a successful set-address command, set the following register
// to the specified address immediately following the status stage.
usbRegs->FADDR = device_address;
```

User Case 3: Example 3 shows an example of how to program the USB endpoints in host mode.

Example 3. Programming the USB Endpoints in Host Mode

```
// DMA channel number.  Valid values are 0, 1, 2, or 3.
int CHAN_NUM = 0;

// Fifo sizes: uncomment the desired size.
// This example uses 64-byte fifo.
// int fifosize = 0;    // 8 bytes
// int fifosize = 1;    // 16 bytes
// int fifosize = 2;    // 32 bytes
   int fifosize = 3;    // 64 bytes
// int fifosize = 4;    // 128 bytes
// int fifosize = 5;    // 256 bytes
// int fifosize = 6;    // 512 bytes
// int fifosize = 7;    // 1024 bytes
// int fifosize = 8;    // 2048 bytes
// int fifosize = 9;    // 4096 bytes

// FIFO address.  Leave 64-bytes for endpoint 0.
int fifo_start_address = 8;

// Uncomment the desired buffering.  If double-buffer is selected, actual
// FIFO space will be twice the value listed above for fifosize.
// This example uses single buffer.
   int double_buffer = 0;  // Single-buffer
// int double_buffer = 1;  // Double-buffer

// Set the following variable to the device endpoint type: CONTROL ISO BULK or IN
  int device_protocol = BULK;
//int device_protocol = ISO;
//int device_protocol = INT;

// USB speeds
#define LOW_SPEED 0
#define FULL_SPEED 1
#define HIGH_SPEED 2

// TXTYPE protocol
#define CONTROL 0
#define ISO 1
#define BULK 2
#define INT 3

// For maximum packet size this formula will usually work, but it can also be
// set to another value if needed.  If non power of 2 value is needed (such as
// 1023) set it explicitly.
#define FIFO_MAXP 8*(1<<fifosize);

// Set the following variable to the device address.
int device_address = 1;

// Set the following variable to the device endpoint number.
int device_ep = 1;

// Variable used for endpoint configuration
Uint8 type = 0;

// Variable to keep track of errors
int error = 0;


// The following code should be run after resetting the attached device


// Initialize the endpoint FIFO.  RX and TX will be allocated the same sizes.
usbRegs->INDEX = CHAN_NUM+1;
usbRegs->RXFIFOSZ = fifosize | ((double_buffer & 1)<<4);
usbRegs->RXFIFOADDR = fifo_start_address;
usbRegs->TXFIFOSZ =    fifosize | ((double_buffer & 1)<<4);
```

Example 3. Programming the USB Endpoints in Host Mode (continued)

```
usbRegs->TXFIFOADDR = fifo_start_address + (1<<(fifosize+double_buffer));
usbRegs->RXMAXP = FIFO_MAXP;
usbRegs->TXMAXP = FIFO_MAXP;


//Configure the endpoint
switch (device_speed) {
    case LOW_SPEED : type = (3<<6) | ((device_protocol & 3) << 4) | (device_ep & 0xf); break;
    case FULL_SPEED: type = (2<<6) | ((device_protocol & 3) << 4) | (device_ep & 0xf); break;
    case HIGH_SPEED: type = (1<<6) | ((device_protocol & 3) << 4) | (device_ep & 0xf); break;
    default:error++;
}
usbRegs->EPCSR[CHAN_NUM+1].HOST_TYPE0 = type;  // TXTYPE
usbRegs->EPCSR[CHAN_NUM+1].HOST_RXTYPE = type;

// Set NAK limit / Polling interval (Interrupt & Iso protocols)
if ((device_protocol == INT) || (device_protocol == ISO)) {
    usbRegs->EPCSR[CHAN_NUM+1].HOST_NAKLIMIT0 = TXINTERVAL;    // TX Polling interval
    usbRegs->EPCSR[CHAN_NUM+1].HOST_RXINTERVAL = RXINTERVAL;    // RX Polling interval
} else {
    usbRegs->EPCSR[CHAN_NUM+1].HOST_NAKLIMIT0 = 2;    // Frames to timeout from NAKs
    usbRegs->EPCSR[CHAN_NUM+1].HOST_RXINTERVAL = 2;    // Frames to timeout from NAKs
}

//Set the address for transactions after SET ADDRESS successfully completed
usbRegs->EPTRG[CHAN_NUM+1].TXFUNCADDR = device_address;
usbRegs->EPTRG[CHAN_NUM+1].RXFUNCADDR = device_address;
```

User Case 4: An example of how to do host negotiation to support USB.

If the HOSTREQ bit in the DEVCTL register is set, host negotiation is performed by the hardware when the device enters suspend mode. The bit is cleared when host negotiation is complete.

User Case 5: Example 4 shows an example of how to program the USB DMA controller.

Example 4. Programming the USB DMA Controller

```
// Number of DMA channels
#define NUM_CHANNEL 4

// Number of DMA buffers to allocate
#define RX_BUFFERS 32
#define TX_BUFFERS 32

// Memory region to place DMA buffers and descriptors
#define MEMORY_TARGET ".DDREMIF_0_BUF"

// DMA defines
#define SOP            (Uint32) (1<<31)
#define EOP            (Uint32) (1<<30)
#define OWNER         (Uint32) (1<<29)
#define EOQ            (Uint32) (1<<28)
#define ZERO_BYTE     (Uint32) (1<<23)
#define RX_ABORT     (Uint32) (1<<19)

// DMA channel configuration
#define CH0 0
#define CH1 1
#define CH2 2
#define CH3 3

// Loop variable
int I;

// Variable to keep track of errors
int error = 0;

// Variable to keep track of the number of descriptors built
int tx_desc[NUM_CHANNEL];    // Current TX Buffer descriptor[channel number]
int rx_desc[NUM_CHANNEL];    // Current RX Buffer descriptor[channel number]


// Separate data section for bufferDesc.  NOTE: CPPI buffers MUST be aligned to 16-byte
boundaries.
#pragma DATA_SECTION(rx_bufferDesc, MEMORY_TARGET)
Uint32 rx_bufferDesc[NUM_CHANNEL][RX_BUFFERS];

#pragma DATA_SECTION(tx_bufferDesc, MEMORY_TARGET)
Uint32 tx_bufferDesc[NUM_CHANNEL][TX_BUFFERS];


// Initialize CPPI DMA.  This code is also included in the controller initialization.
usbRegs->RCPPICR = 0;    //Disable the RX DMA
usbRegs->TCPPICR = 0;    //Disable the TX DMA
for(I = 0; i<NUM_CHANNEL;i++) { // Initialize CPPI DMA state
    usbRegs->CHANNEL[i].TCPPIDMASTATEW0 = 0;
    usbRegs->CHANNEL[i].TCPPIDMASTATEW1 = 0;
    usbRegs->CHANNEL[i].TCPPIDMASTATEW2 = 0;
    usbRegs->CHANNEL[i].TCPPIDMASTATEW3 = 0;
    usbRegs->CHANNEL[i].TCPPIDMASTATEW4 = 0;
    usbRegs->CHANNEL[i].TCPPIDMASTATEW5 = 0;

 usbRegs->CHANNEL[i].RCPPIDMASTATEW0 = 0;
    usbRegs->CHANNEL[i].RCPPIDMASTATEW1 = 0;
    usbRegs->CHANNEL[i].RCPPIDMASTATEW2 = 0;
    usbRegs->CHANNEL[i].RCPPIDMASTATEW3 = 0;
    usbRegs->CHANNEL[i].RCPPIDMASTATEW4 = 0;
    usbRegs->CHANNEL[i].RCPPIDMASTATEW5 = 0;
    usbRegs->CHANNEL[i].RCPPIDMASTATEW6 = 0;
    tx_desc[i] = 0;
    rx_desc[i] = 0;
}
```

Example 4. Programming the USB DMA Controller (continued)

```
// Routine to flush TX fifo.
// Must call this routine twice for double-buffered FIFO
void flush_tx_fifo(int ep) {
  int index_save;
    int status;

    index_save = usbRegs->INDEX;            // Save the index to restore later
    usbRegs->INDEX = ep;                            // Set the index to the desired endpoint

    status = usbRegs->PERI_TXCSR & 3;    // Isolate the TxPktRdy and FIFONotEmpty bits
    if (!(status)) {                                    // Nothing showing in FIFO
        usbRegs->PERI_TXCSR |= 1;                // Set TxPktRdy in case there is a partial
packet already in FIFO
    }

    usbRegs->PERI_TXCSR = ((usbRegs->PERI_TXCSR & 0xFFFC) | 8);    // Write TXCSR with flush bit
set, FIFONotEmpty=0, and TxPktRdy=0.
    while (usbRegs->PERI_TXCSR & 8);    // Keep looping until the flush bit clears

    usbRegs->INDEX = index_save;        // Restore the index to previous value
}


// Routine to start the TX DMA for a given channel
void start_tx_dma(int ch) {

    // Must have at least one descriptor before turning on TX DMA
    if (rx_desc[ch] < 1) {error++;} else {

        //Flush FIFO (2 times in case it is double-buffered)
        flush_tx_fifo(ch+1);
        flush_tx_fifo(ch+1);

        // Start the DMA
        usbRegs->TCPPICR = 1; //Enable Tx CPPI DMA
        usbRegs->CHANNEL[ch].TCPPIDMASTATEW0 = (Uint32)(&tx_bufferDesc[ch][0]);
        CSL_FINS(usbRegs->PERI_TXCSR,USB_PERI_TXCSR_DMAEN,1);    // TXCSR, bit DMAReqEnab
    }
}

// Routine to add a TX descriptor
void add_tx_descriptor(int ch, unsigned char * inBuf, int bytes) {
    if ((bytes < 0) || (bytes >65535)) {bytes = 64; error++;}

    // Link previous buffer to this one if this is not the first descriptor of the channel
    if (tx_desc[ch] > 0) tx_bufferDesc[ch][4*(tx_desc[ch]-1)] =
(Uint32)(&tx_bufferDesc[ch][4*tx_desc[ch]]);


// Set up DMA buffer descriptors
    tx_bufferDesc[ch][4*tx_desc[ch]+0] = (Uint32)(0x00000000);        // Next Descriptor pointer
    tx_bufferDesc[ch][4*tx_desc[ch]+1] = (Uint32)inBuf;                // Buffer pointer
    tx_bufferDesc[ch][4*tx_desc[ch]+2] = (0x0000 << 16) | bytes;    // [31:16] Buffer offset
[15:0] Buffer length
    if (bytes == 0) bytes = ZERO_BYTE | 1;    // Set the ZERO_BYTE bit and size 1 byte
    tx_bufferDesc[ch][4*tx_desc[ch]+3] = SOP | EOP | OWNER | bytes;    // [31]=SOP, [30]=EOP,
[29]=owner, [28]=EOQ, [23]=zero-byte, [19] = rxabort, [15:0]=packet length


    // If DMA already enabled and has stopped, write this to the TX Queue head pointer
    if ((usbRegs->TCPPICR == 1) && (usbRegs->CHANNEL[ch].TCPPIDMASTATEW0 == 0))
        usbRegs->CHANNEL[ch].TCPPIDMASTATEW0 = (Uint32)(&tx_bufferDesc[ch][4*tx_desc[ch]+0]);

    // Increment descriptor counter
    tx_desc[ch]++;
}
```

Example 4. Programming the USB DMA Controller (continued)

```
// Routine to start the RX DMA for a given channel
void start_rx_dma(int ch) {
  int index_save;
    index_save = usbRegs->INDEX;            // Save the index to restore later

    // Must have at least 3 descriptors to receive anything
    if (rx_desc[ch] < 2) {error++;} else {

        usbRegs->INDEX = ch+1;
        usbRegs->RCPPICR = 1; //Enable Rx CPPI DMA
        usbRegs->CHANNEL[ch].RCPPIDMASTATEW1 = (Uint32)(&rx_bufferDesc[ch][0]);
        CSL_FINS(usbRegs->PERI_RXCSR,USB_PERI_RXCSR_DMAEN,1);

        // Update the buffer count register.  This ADDS to the existing value, does not overwrite.
        switch (ch) {
            case 0: usbRegs->RBUFCNT0 = rx_desc[ch]; break;
            case 1: usbRegs->RBUFCNT1 = rx_desc[ch]; break;
            case 2: usbRegs->RBUFCNT2 = rx_desc[ch]; break;
            case 3: usbRegs->RBUFCNT3 = rx_desc[ch]; break;
        }
    }
    usbRegs->INDEX = index_save;        // Restore the index to previous value
}

//Function to build Rx DMA descriptors
void add_rx_descriptor(int ch, unsigned char * outBuf, int bytes) {
  int index_save;
    index_save = usbRegs->INDEX;            // Save the index to restore later

    if (bytes < 1) {bytes = 64; error++;}
    usbRegs->INDEX = ch+1;

    // Link previous buffer to this one if this is not the first descriptor of the channel
    if (rx_desc[ch] > 0) rx_bufferDesc[ch][4*(rx_desc[ch]-1)] =
(Uint32)(&rx_bufferDesc[ch][4*rx_desc[ch]]);

    rx_bufferDesc[ch][4*rx_desc[ch]+0] = (Uint32)(0x00000000);    // Next descriptor
    rx_bufferDesc[ch][4*rx_desc[ch]+1] = (Uint32)outBuf;           // Buffer address
    rx_bufferDesc[ch][4*rx_desc[ch]+2] = (0x0000 << 16) | bytes;    // Rx buffer size in bytes
    rx_bufferDesc[ch][4*rx_desc[ch]+3] = OWNER | 0;                // OWNER
    rx_desc[ch]++;

    usbRegs->INDEX = index_save;        // Restore the index to previous value
}
```

1.6 Industry Standard(s) Compliance Statement

This device conforms to USB 2.0 Specification and On-The-Go Supplement to the USB 2.0 Specification Rev 1.0a.

2 Peripheral Architecture

2.1 Clock Control

Information related to clock generation and control for the USB peripheral will be added in a future revision of this document. Clocks for USB are generated based on a crystal oscillator on the M24XI and M24XO pins. The oscillator is enabled by bit OSCPDWN of the USBPHY_CTL register in the system module.

2.2 Signal Descriptions

The USB controller provides the following I/O signals. Table 1 shows the USB port pins used in each mode.

Table 1. USB Pins

Pin	Type [1]	Function
M24XI	I	Crystal input for M24 oscillator (24 MHZ for USB)
M24X0	O	Crystal output for M24 oscillator
$M24V_{DD}$	S	1.8V power supply for M24 oscillator
$M24V_{SS}$		Ground for M24 oscillator
$PLLV_{DD18}$	GND	1.8 Volt power supply for PLLs (system and USB)
USB_VBUS	I/O	5V input that signifies that VBUS is connected. The OTG section of the PHY can also pull up or down on this signal for HNP and SRP.
USB_ID	I/O	USB_ID is an input that is open or pulled to ground depending on OTG connector configuration. The state determines if controller starts in HOST or PERIPHERAL mode.
USB_DP	I/O	USB bi-directional Data Differential signal pair [positive/negative]. Input/output DP signal of the differential signal pair.
USB_DM	I/O	USB bi-directional Data Differential signal pair [positive/negative]. Input/output DM signal of the differential signal pair.
USB_R1	I/O	Reference current output. This must be connected via a 10 kΩ± 1% resistor to USB_V_{SSREF}.
USB_V_{SSREF}	GND	Ground for reference current
USB_V_{DDA3P3}	S	Analog 3.3 V power supply for USB phy
USB_V_{SSA3P3}	GND	Analog ground for USB phy
USB_V_{DD1P83}	S	1.8 V I/O power supply for USB phy
USB_V_{SS1P8}	GND	I/O Ground for USB phy
$USB\_V_{DDA1P2LDO}$	S	Core Power supply LDO output for USB phy. This must be connected via 1 μF capacitor to $USB\_V_{SSA1P2LDO}$. Do not connect this to other supply pins.
$USB\_V_{SSA1P2LDO}$	GND	Core Ground for USB phy. This must be connected via 1 μF capacitor to $USB\_V_{DDA1P2LDO}$.

[1] I = Input, O = Output, Z = High impedance, S = Supply voltage, GND = Ground, A = Analog signal

2.3 Indexed and Non-Indexed Registers

USB controller provides two mechanism of accessing the endpoint control and status registers:

- Indexed Endpoint Control/Status Registers – These registers are memory-mapped at offset 410h to 41Fh. The endpoint is selected by programming the INDEX register of the controller.
- Non-indexed Endpoint Control/Status Registers – These registers are memory-mapped at offset 500h to 54Fh. Registers at offset 500h-50Fh map to Endpoint 0; offset 510h-51Fh map to Endpoint 1, and so on.

For detailed information about the USB controller registers, see Section 4.

2.4 USB PHY Initialization

The USB PHY and its interface to the USB controller, UTMI Interface, and interface to the bus require a low jitter clock, sourced external to the device, for PHY operation.

The input clock frequency can either be 12 MHz or 24 MHz. The PLL within the PHY is used to generate the clock required for the UTMI interface and for the bit clock. The clock frequency supplied is selected using the System Module USBPHY_CTL.CLK01SEL bit (0 = 24 MHz; 1 = 12 MHz). This clock source can either be an oscillator or a crystal. If using an oscillator, you are advised to disable the internal oscillator. If using a crystal, you are required to enable the internal oscillator. The oscillator control is handled using the USBPHY_CTL.OSCPDWN bit (0 = internal oscillator is powered; 1 = internal oscillator is disabled). The PLL state is controlled using the USBPHY_CTL.PHYPLLON bit (0 = PLL is enabled; 1 = PLL is disabled).

The PHY control is handled using the USBPHY_CTL.PHYPDWN bit (0 = powered on; 1 = powered off). Once the PHY is supplied with the desired input clock source and is powered up (USBPHY_CTL.PHYPDWN = 0), the firmware should wait until the PLL locks prior to usage. The PLL lock status is communicated to you using the USBPHY_CTL.PHYCLKGD status bit (0 = PLL is not locked; 1 = PLL is locked).

In addition to providing the input clock, you are also responsible for enabling the voltage comparators within the PHY. The OTG controller expects to observe different levels of voltages depending upon the role it assumes and its operation state. Voltage comparators within the PHY exist to identify the voltage level on the VBUS line and communicate the level to the USB controller. The voltage comparators status is communicated using the USBPHY_CTL.VBDTCTEN and USBPHY_CTL_SESNDEN bits (0 = comparator is disabled; 1 = comparator is enabled).

For information on the USBPHY_CTL register, refer to the device-specific data manual.

2.5 Dynamic FIFO Sizing

The USB controller supports a total of 4K RAM to dynamically allocate FIFO to all endpoints. The allocation of FIFO space to the different endpoints requires the specification for each Tx and Rx endpoint of:

- The start address of the FIFO within the RAM block
- The maximum size of packet to be supported
- Whether double-buffering is required.

These details are specified through four registers, which are added to the indexed area of the memory map. That is, the registers for the desired endpoint are accessed after programming the INDEX register with the desired endpoint value. Section 4.78, Section 4.79, Section 4.80, and Section 4.81 provide details of these registers.

NOTE: The option of setting FIFO sizes dynamically only applies to Endpoints 1 … 4. Endpoint 0 FIFO has a fixed size (64 bytes) and a fixed location (start address 0).

It is the responsibility of the firmware to ensure that all the Tx and Rx endpoints that are active in the current USB configuration have a block of RAM assigned exclusively to that endpoint that is at least as large as the maximum packet size set for that endpoint.

3 USB Controller Host and Peripheral Modes Operation

The USB controller can be used in a range of different environments. It can be used as either a high-speed or a full-speed USB peripheral device attached to a conventional USB host (such as a PC). It can be used as either host or peripheral device in point-to-point data transfers with another peripheral device - or, if the other device also contains a Dual-Role Controller, the two devices can switch roles as required. (This second device may be either a high-speed, full-speed or low-speed USB function.) Or the controller can be used as the host to a range of such peripheral devices in a multi-point setup.

Whether the controller expects to behave as a host or as a peripheral device depends on the way the devices are cabled together. Each USB cable has an A end and a B end. If the A end of the cable is plugged into the controller, it will take the role of the Host device and go into host mode. If the B end of the cable is plugged in, the controller will go instead into peripheral mode.

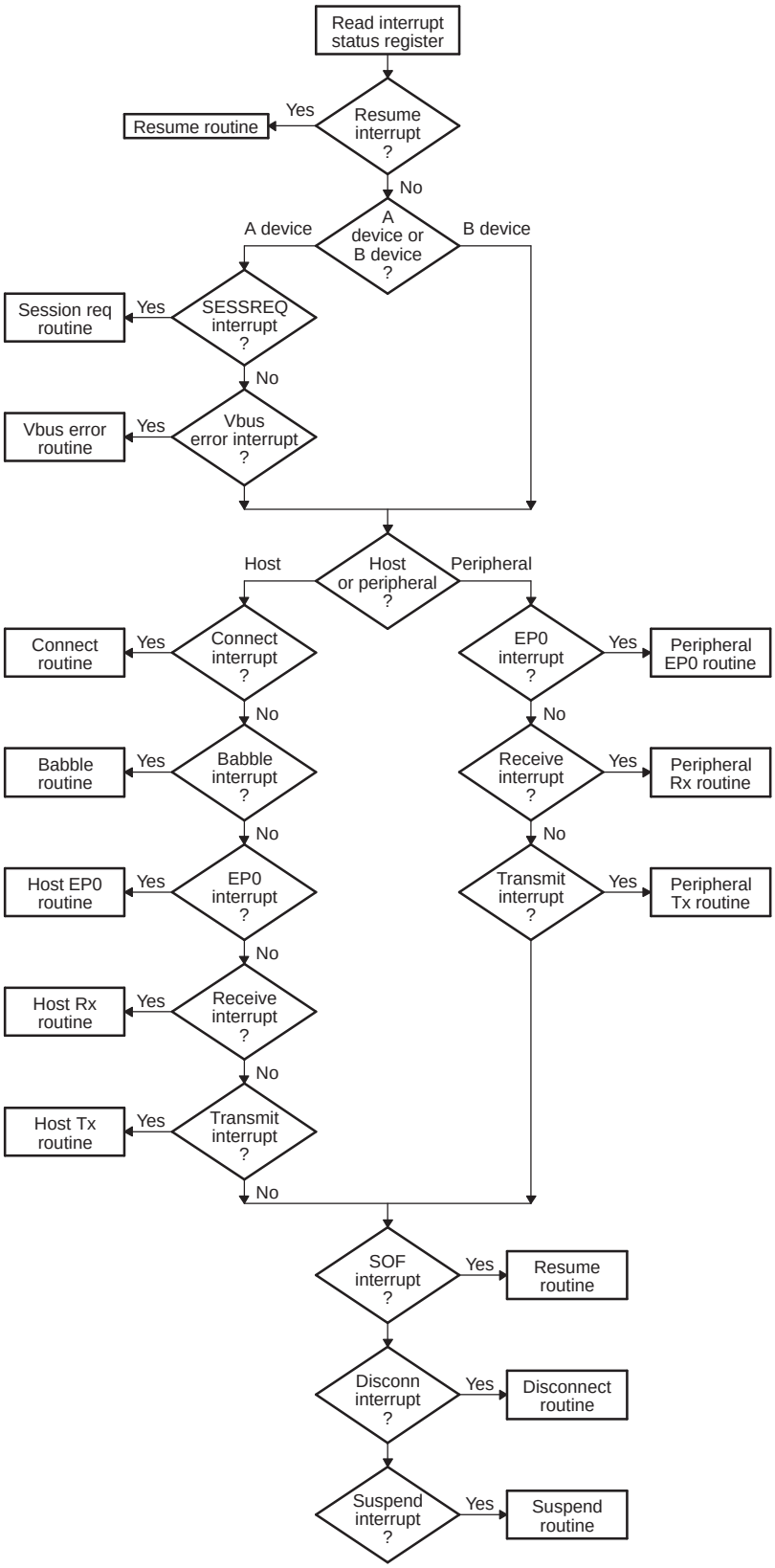
The USB controller interrupts the ARM on completion of the data transfer on any of the endpoints or on detecting reset, resume, suspend, connect, disconnect, or SOF on the bus.

When the ARM is interrupted with a USB interrupt, it needs to read the interrupt status register to determine the endpoints that have caused the interrupt and jump to the appropriate routine. If multiple endpoints have caused the interrupt, endpoint 0 should be serviced first, followed by the other endpoints. The suspend interrupt should be serviced last.

The flowchart in Figure 2 describes the interrupt service routine for the USB module.

The following sections describe the programming of USB controller in Peripheral mode and Host mode. DMA Operations and Interrupt Handler mechanisms are common to both peripheral and host mode operations and are discussed after the programming in Peripheral and Host Mode.

Figure 2. Interrupt Service Routine Flow Chart



3.1 USB Controller Peripheral Mode Operation

- *Soft connect* - After a reset, the SOFTCONN bit of POWER register (bit 6) is cleared to 0. The controller will therefore appear disconnected until the software has set the SOFTCONN bit to 1. The application software can then choose when to set the PHY into its normal mode. Systems with a lengthy initialization procedure may use this to ensure that initialization is complete and the system is ready to perform enumeration before connecting to the USB.

  Once the SOFTCONN bit has been set, the software can also simulate a disconnect by clearing this bit to 0.
- *Entry into suspend mode*

  When operating as a peripheral device, the controller monitors activity on the bus and when no activity has occurred for 3 ms, it goes into Suspend mode. If the Suspend interrupt has been enabled, an interrupt will be generated at this time.

  At this point, the controller can then be left active (and hence able to detect when Resume signaling occurs on the USB), or the application may arrange to disable the controller by stopping its clock. However, the controller will not then be able to detect Resume signaling on the USB. As a result, some external hardware will be needed to detect Resume signaling (by monitoring the DM and DP signals), so that the clock to the controller can be restarted.
- *Resume Signaling* - When resume signaling occurs on the bus, first the clock to the controller must be restarted if necessary. Then the controller will automatically exit Suspend mode. If the Resume interrupt is enabled, an interrupt will be generated.
- *Initiating a remote wakeup* - If the software wants to initiate a remote wakeup while the controller is in Suspend mode, it should write to the Power register to set the RESUME bit to 1. The software should leave then this bit set for approximately 10 ms (minimum of 2 ms, a maximum of 15 ms) before resetting it to 0.

  NOTE: No resume interrupt will be generated when the software initiates a remote wakeup.
- *Reset Signaling* - When reset signaling occurs on the bus, the controller will perform the following actions:
  - Sets FADDR register to 0
  - Sets INDEX register to 0
  - Flushes all endpoint FIFOs
  - Clears all control/status registers
  - Generates a reset interrupt.

  If the HSENA bit in the POWER register (bit 5) was set, the controller also tries to negotiate for high-speed operation.

  Whether high-speed operation is selected is indicated by HSMODE bit of POWER register (bit 4).

  When the application software receives a reset interrupt, it should close any open pipes and wait for bus enumeration to begin.

3.1.1 Peripheral Mode: Control Transactions

Endpoint 0 is the main control endpoint of the core. The software is required to handle all the standard device requests that may be sent or received via endpoint 0. These are described in Universal Serial Bus Specification, Revision 2.0, Chapter 9. The protocol for these device requests involves different numbers and types of transactions per transfer. To accommodate this, the software needs to take a state machine approach to command decoding and handling.

The Standard Device Requests received by a USB peripheral device can be divided into three categories: Zero Data Requests (in which all the information is included in the command), Write Requests (in which the command will be followed by additional data), and Read Requests (in which the device is required to send data back to the host).

This section looks at the sequence of actions that the software must perform to process these different types of device request.

NOTE: The Setup packet associated with any standard device request should include an 8-byte command. Any setup packet containing a command field of anything other than 8 bytes will be automatically rejected by the controller.

3.1.1.1 Zero Data Requests

Zero data requests have all their information included in the 8-byte command and require no additional data to be transferred. Examples of Zero Data standard device requests are:

- SET_FEATURE
- CLEAR_FEATURE
- SET_ADDRESS
- SET_CONFIGURATION
- SET_INTERFACE

The sequence of events will begin, as with all requests, when the software receives an endpoint 0 interrupt. The RXPKTRDY bit of PERI_CSR0 (bit 0) will also have been set. The 8-byte command should then be read from the endpoint 0 FIFO, decoded and the appropriate action taken.

For example, if the command is SET_ADDRESS, the 7-bit address value contained in the command should be written to the FADDR register. The PERI_CSR0 register should then be written to set the SERV_RXPKTRDY bit (bit 6) (indicating that the command has been read from the FIFO) and to set the DATAEND bit (bit 3) (indicating that no further data is expected for this request). The interval between setting SERV_RXPKTRDY bit and DATAEND bit should be very small to avoid getting a SetupEnd error condition.

When the host moves to the status stage of the request, a second endpoint 0 interrupt will be generated to indicate that the request has completed. No further action is required from the software. The second interrupt is just a confirmation that the request completed successfully. For SET_ADDRESS command, the address should be set in FADDR register only after the status stage interrupt is received.

If the command is an unrecognized command, or for some other reason cannot be executed, then when it has been decoded, the PERI_CSR0 register should be written to set the SERV_RXPKTRDY bit (bit 6) and to set the SENDSTALL bit (bit 5). When the host moves to the status stage of the request, the controller will send a STALL to tell the host that the request was not executed. A second endpoint 0 interrupt will be generated and the SENTSTALL bit (bit 2 of PERI_CSR0) will be set.

If the host sends more data after the DATAEND bit has been set, then the controller will send a STALL. An endpoint 0 interrupt will be generated and the SENTSTALL bit (bit 2 of PERI_CSR0) will be set.

NOTE: DMA is not supported for endpoint 0, so the command should be read by accessing the endpoint 0 FIFO register.

3.1.1.2 Write Requests

Write requests involve an additional packet (or packets) of data being sent from the host after the 8-byte command. An example of a Write standard device request is: SET_DESCRIPTOR.

The sequence of events will begin, as with all requests, when the software receives an endpoint 0 interrupt. The RXPKTRDY bit of PERI_CSR0 will also have been set. The 8-byte command should then be read from the Endpoint 0 FIFO and decoded.

As with a zero data request, the PERI_CSR0 register should then be written to set the SERV_RXPKTRDY bit (bit 6) (indicating that the command has been read from the FIFO) but in this case the DATAEND bit (bit 3) should not be set (indicating that more data is expected).

When a second endpoint 0 interrupt is received, the PERI_CSR0 register should be read to check the endpoint status. The RXPKTRDY bit of PERI_CSR0 should be set to indicate that a data packet has been received. The COUNT0 register should then be read to determine the size of this data packet. The data packet can then be read from the endpoint 0 FIFO.

If the length of the data associated with the request (indicated by the wLength field in the command) is greater than the maximum packet size for endpoint 0, further data packets will be sent. In this case, PERI_CSR0 should be written to set the SERV_RXPKTRDY bit, but the DATAEND bit should not be set.

When all the expected data packets have been received, the PERI_CSR0 register should be written to set the SERV_RXPKTRDY bit and to set the DATAEND bit (indicating that no more data is expected).

When the host moves to the status stage of the request, another endpoint 0 interrupt will be generated to indicate that the request has completed. No further action is required from the software, the interrupt is just a confirmation that the request completed successfully.

If the command is an unrecognized command, or for some other reason cannot be executed, then when it has been decoded, the PERI_CSR0 register should be written to set the SERV_RXPKTRDY bit (bit 6) and to set the SENDSTALL bit (bit 5). When the host sends more data, the controller will send a STALL to tell the host that the request was not executed. An endpoint 0 interrupt will be generated and the SENTSTALL bit of PERI_CSR0 (bit 2) will be set.

If the host sends more data after the DATAEND has been set, then the controller will send a STALL. An endpoint 0 interrupt will be generated and the SENTSTALL bit of PERI_CSR0 (bit 2) will be set.

3.1.1.3 Read Requests

Read requests have a packet (or packets) of data sent from the function to the host after the 8-byte command. Examples of Read Standard Device Requests are:

- GET_CONFIGURATION
- GET_INTERFACE
- GET_DESCRIPTOR
- GET_STATUS
- SYNCH_FRAME

The sequence of events will begin, as with all requests, when the software receives an endpoint 0 interrupt. The RXPKTRDY bit of PERI_CSR0 (bit 0) will also have been set. The 8-byte command should then be read from the endpoint 0 FIFO and decoded. The PERI_CSR0 register should then be written to set the SERV_RXPKTRDY bit (bit 6) (indicating that the command has read from the FIFO).

The data to be sent to the host should then be written to the endpoint 0 FIFO. If the data to be sent is greater than the maximum packet size for endpoint 0, only the maximum packet size should be written to the FIFO. The PERI_CSR0 register should then be written to set the TXPKTRDY bit (bit 1) (indicating that there is a packet in the FIFO to be sent). When the packet has been sent to the host, another endpoint 0 interrupt will be generated and the next data packet can be written to the FIFO.

When the last data packet has been written to the FIFO, the PERI_CSR0 register should be written to set the TXPKTRDY bit and to set the DATAEND bit (bit 3) (indicating that there is no more data after this packet).

When the host moves to the status stage of the request, another endpoint 0 interrupt will be generated to indicate that the request has completed. No further action is required from the software: the interrupt is just a confirmation that the request completed successfully.

If the command is an unrecognized command, or for some other reason cannot be executed, then when it has been decoded, the PERI_CSR0 register should be written to set the SERV_RXPKTRDY bit (bit 6) and to set the SENDSTALL bit (bit 5). When the host requests data, the controller will send a STALL to tell the host that the request was not executed. An endpoint 0 interrupt will be generated and the SENTSTALL bit of PERI_CSR0 (bit 2) will be set.

If the host requests more data after DATAEND (bit 3) has been set, then the controller will send a STALL. An endpoint 0 interrupt will be generated and the SENTSTALL bit of PERI_CSR0 (bit 2) will be set.

3.1.1.4 Endpoint 0 States

When the USB controller is operating as a peripheral device, the endpoint 0 control needs three modes – IDLE, TX and RX – corresponding to the different phases of the control transfer and the states endpoint 0 enters for the different phases of the transfer (described in later sections).

The default mode on power-up or reset should be IDLE. RXPKTRDY bit of PERI_CSR0 (bit 0) becoming set when endpoint 0 is in IDLE state indicates a new device request. Once the device request is unloaded from the FIFO, the controller decodes the descriptor to find whether there is a data phase and, if so, the direction of the data phase of the control transfer (in order to set the FIFO direction).See Figure 3.

Depending on the direction of the data phase, endpoint 0 goes into either TX state or RX state. If there is no Data phase, endpoint 0 remains in IDLE state to accept the next device request.

The actions that the CPU needs to take at the different phases of the possible transfers (e.g., loading the FIFO, setting TXPKTRDY) are indicated in Figure 4 .

NOTE: The controller changes the FIFO direction, depending on the direction of the data phase independently of the CPU.

Figure 3. CPU Actions at Transfer Phases

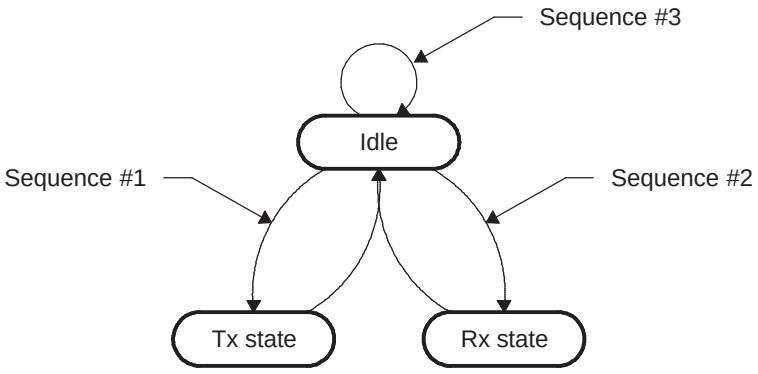
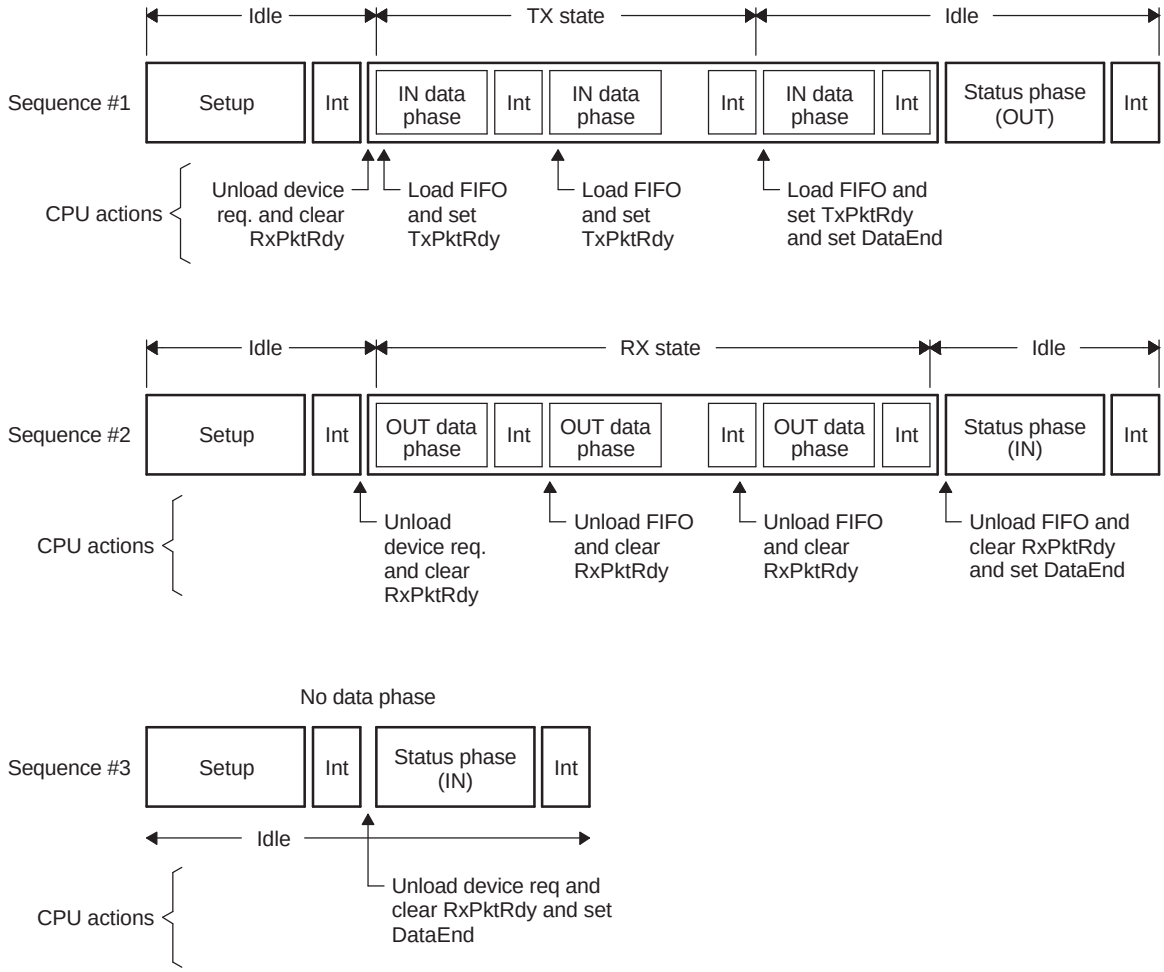


Figure 4. Sequence of Transfer



3.1.1.5 Endpoint 0 Service Routine

An Endpoint 0 interrupt is generated when:

- The controller sets the RXPKTRDY bit of PERI_CSR0 (bit 0) after a valid token has been received and data has been written to the FIFO.
- The controller clears the TXPKTRDY bit of PERI_CSR0 (bit 1) after the packet of data in the FIFO has been successfully transmitted to the host.
- The controller sets the SENTSTALL bit of PERI_CSR0 (bit 2) after a control transaction is ended due to a protocol violation.
- The controller sets the SETUPEND bit of PERI_CSR0 (bit 4) because a control transfer has ended before DATAEND (bit 3 of PERI_CSR0) is set.

Whenever the endpoint 0 service routine is entered, the software must first check to see if the current control transfer has been ended due to either a STALL condition or a premature end of control transfer. If the control transfer ends due to a STALL condition, the SENTSTALL bit would be set. If the control transfer ends due to a premature end of control transfer, the SETUPEND bit would be set. In either case, the software should abort processing the current control transfer and set the state to IDLE.

Once the software has determined that the interrupt was not generated by an illegal bus state, the next action taken depends on the endpoint state. Figure 5 shows the flow of this process.

If endpoint 0 is in IDLE state, the only valid reason an interrupt can be generated is as a result of the controller receiving data from the bus. The service routine must check for this by testing the RXPKTRDY bit of PERI_CSR0 (bit 0). If this bit is set, then the controller has received a SETUP packet. This must be unloaded from the FIFO and decoded to determine the action the controller must take. Depending on the command contained within the SETUP packet, endpoint 0 will enter one of three states:

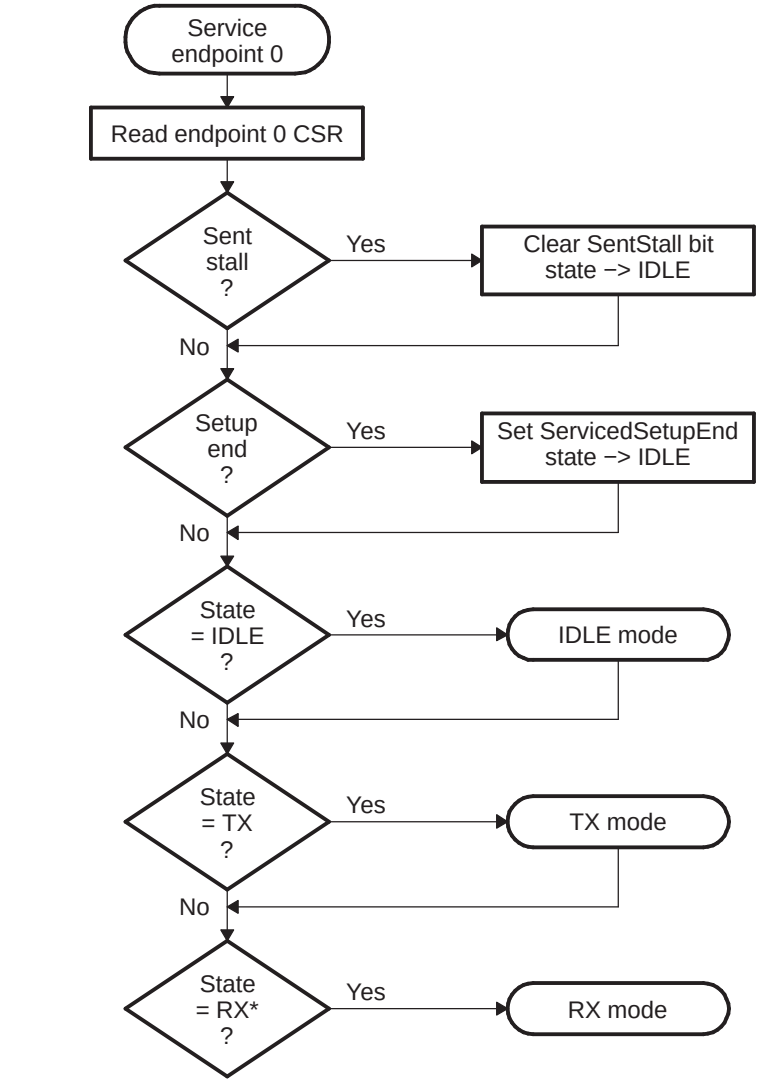
- If the command is a single packet transaction (SET_ADDRESS, SET_INTERFACE etc.) without any data phase, the endpoint will remain in IDLE state.
- If the command has an OUT data phase (SET_DESCRIPTOR etc.), the endpoint will enter RX state.
- If the command has an IN data phase (GET_DESCRIPTOR etc.), the endpoint will enter TX state.

If the endpoint 0 is in TX state, the interrupt indicates that the core has received an IN token and data from the FIFO has been sent. The software must respond to this either by placing more data in the FIFO if the host is still expecting more data or by setting the DATAEND bit to indicate that the data phase is complete. Once the data phase of the transaction has been completed, endpoint 0 should be returned to IDLE state to await the next control transaction.

NOTE: All command transactions include a field that indicates the amount of data the host expects to receive or is going to send.

If the endpoint is in RX state, the interrupt indicates that a data packet has been received. The software must respond by unloading the received data from the FIFO. The software must then determine whether it has received all of the expected data. If it has, the software should set the DATAEND bit and return endpoint 0 to IDLE state. If more data is expected, the firmware should set the SERV_RXPKTRDY bit of PERI_CSR0 (bit 6) to indicate that it has read the data in the FIFO and leave the endpoint in RX state.

Figure 5. Service Endpoint 0 Flow Chart

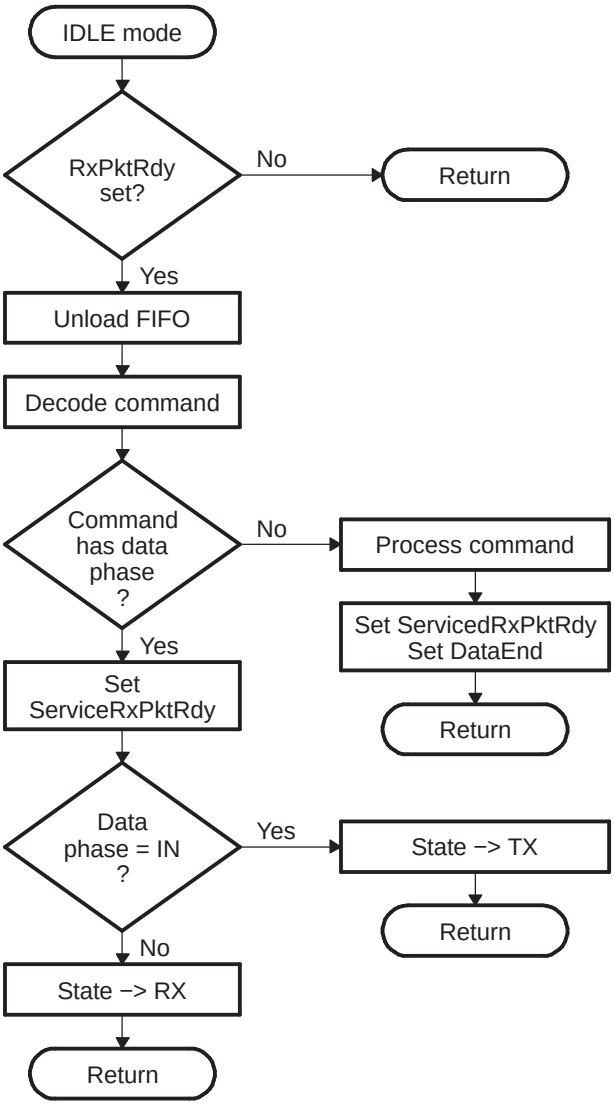


* By default

3.1.1.5.1 IDLE Mode

IDLE mode is the mode the endpoint 0 control must select at power-on or reset and is the mode to which the endpoint 0 control should return when the RX and TX modes are terminated. It is also the mode in which the SETUP phase of control transfer is handled (as outlined in Figure 6).

Figure 6. IDLE Mode Flow Chart



3.1.1.5.2 TX Mode

When the endpoint is in TX state all arriving IN tokens need to be treated as part of a data phase until the required amount of data has been sent to the host. If either a SETUP or an OUT token is received while the endpoint is in the TX state, this will cause a SetupEnd condition to occur as the core expects only IN tokens. See Figure 7.

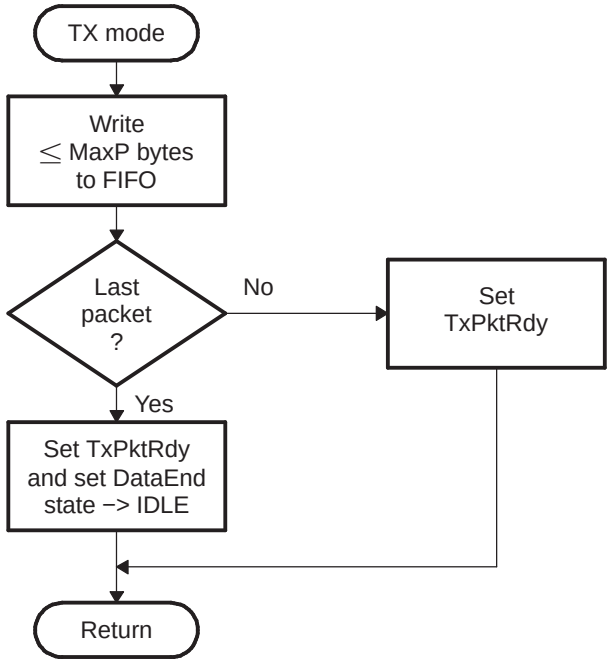
Three events can cause TX mode to be terminated before the expected amount of data has been sent:

1. The host sends an invalid token causing a SETUPEND condition (bit 4 of PERI_CSR0 set).
2. The software sends a packet containing less than the maximum packet size for endpoint 0.
3. The software sends an empty data packet.

Until the transaction is terminated, the software simply needs to load the FIFO when it receives an interrupt that indicates a packet has been sent from the FIFO. (An interrupt is generated when TXPKTRDY is cleared.)

When the software forces the termination of a transfer (by sending a short or empty data packet), it should set the DATAEND bit of PERI_CSR0 (bit 3) to indicate to the core that the data phase is complete and that the core should next receive an acknowledge packet.

Figure 7. TX Mode Flow Chart



3.1.1.5.3 RX Mode

In RX mode, all arriving data should be treated as part of a data phase until the expected amount of data has been received. If either a SETUP or an IN token is received while the endpoint is in RX state, a SetupEnd condition will occur as the controller expects only OUT tokens.

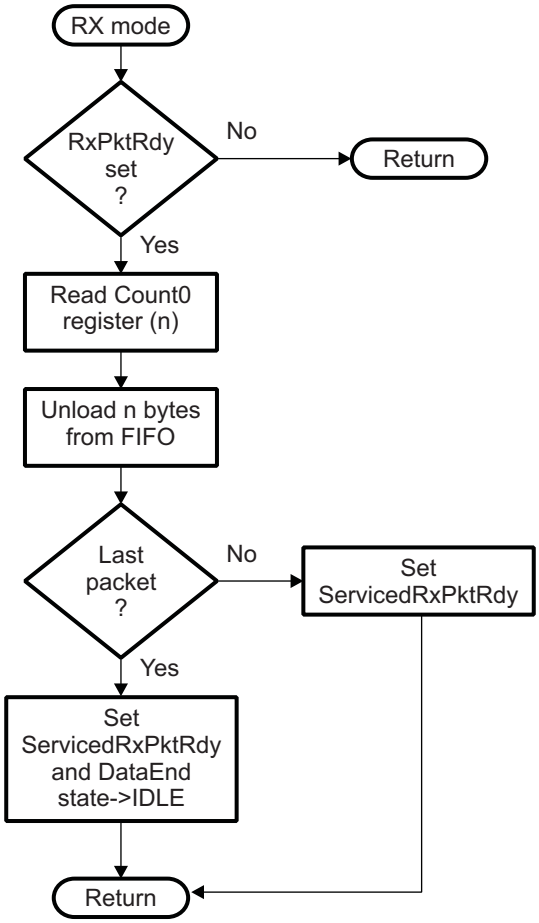
Three events can cause RX mode to be terminated before the expected amount of data has been received as shown in Figure 8:

1. The host sends an invalid token causing a SETUPEND condition (setting bit 4 of PERI_CSR0).
2. The host sends a packet which contains less than the maximum packet size for endpoint 0.
3. The host sends an empty data packet.

Until the transaction is terminated, the software unloads the FIFO when it receives an interrupt that indicates new data has arrived (setting RXPKTRDY bit of PERI_CSR0) and to clear RXPKTRDY by setting the SERV_RXPKTRDY bit of PERI_CSR0 (bit 6).

When the software detects the termination of a transfer (by receiving either the expected amount of data or an empty data packet), it should set the DATAEND bit (bit 3 of PERI_CSR0) to indicate to the controller that the data phase is complete and that the core should receive an acknowledge packet next.

Figure 8. RX Mode Flow Chart



3.1.1.5.4 Error Handling

A control transfer may be aborted due to a protocol error on the USB, the host prematurely ending the transfer, or if the software wishes to abort the transfer (e.g., because it cannot process the command).

The controller automatically detects protocol errors and sends a STALL packet to the host under the following conditions:

- The host sends more data during the OUT Data phase of a write request than was specified in the command. This condition is detected when the host sends an OUT token after the DATAEND bit (bit 3 of PERI_CSR0) has been set.
- The host requests more data during the IN Data phase of a read request than was specified in the command. This condition is detected when the host sends an IN token after the DATAEND bit in the PERI_CSR0 register has been set.
- The host sends more than Max Packet Size data bytes in an OUT data packet.
- The host sends a non-zero length DATA1 packet during the STATUS phase of a read request.

When the controller has sent the STALL packet, it sets the SENTSTALL bit (bit 2 of PERI_CSR0) and generates an interrupt. When the software receives an endpoint 0 interrupt with the SENTSTALL bit set, it should abort the current transfer, clear the SENTSTALL bit, and return to the IDLE state.

If the host prematurely ends a transfer by entering the STATUS phase before all the data for the request has been transferred, or by sending a new SETUP packet before completing the current transfer, then the SETUPEND bit (bit 4 of PERI_CSR0) will be set and an endpoint 0 interrupt generated. When the software receives an endpoint 0 interrupt with the SETUPEND bit set, it should abort the current transfer, set the SERV_SETUPEND bit (bit 7 of PERI_CSR0), and return to the IDLE state. If the RXPKTRDY bit (bit 0 of PERI_CSR0) is set this indicates that the host has sent another SETUP packet and the software should then process this command.

If the software wants to abort the current transfer, because it cannot process the command or has some other internal error, then it should set the SENDSTALL bit (bit 5 of PERI_CSR0). The controller will then send a STALL packet to the host, set the SENTSTALL bit (bit 2 of PERI_CSR0) and generate an endpoint 0 interrupt.

3.1.1.5.5 Additional Conditions

When working as a peripheral device, the controller automatically responds to certain conditions on the USB bus or actions by the host. The details are given below:

- Stall Issued to Control Transfers
  - The host sends more data during an OUT Data phase of a Control transfer than was specified in the device request during the SETUP phase. This condition is detected by the controller when the host sends an OUT token (instead of an IN token) after the software has unloaded the last OUT packet and set DataEnd.
  - The host requests more data during an IN data phase of a Control transfer than was specified in the device request during the SETUP phase. This condition is detected by the controller when the host sends an IN token (instead of an OUT token) after the software has cleared TXPKTRDY and set DataEnd in response to the ACK issued by the host to what should have been the last packet.
  - The host sends more than MaxPktSize data with an OUT data token.
  - The host sends the wrong PID for the OUT Status phase of a Control transfer.
  - The host sends more than a zero length data packet for the OUT Status phase.
- Zero Length Out Data Packets In Control Transfer
  - A zero length OUT data packet is used to indicate the end of a Control transfer. In normal operation, such packets should only be received after the entire length of the device request has been transferred (i.e., after the software has set DataEnd). If, however, the host sends a zero length OUT data packet before the entire length of device request has been transferred, this signals the premature end of the transfer. In this case, the controller will automatically flush any IN token loaded by software ready for the Data phase from the FIFO and set SETUPEND bit (bit 4 of PERI_CSR0).

3.1.2 Bulk Transactions

3.1.2.1 Peripheral Mode: Bulk In Transactions

A Bulk IN transaction is used to transfer non-periodic data from the USB peripheral device to the host.

The following optional features are available for use with a Tx endpoint used in peripheral mode for Bulk IN transactions:

- Double packet buffering: When enabled, up to two packets can be stored in the FIFO awaiting transmission to the host. Double packet buffering is enabled by setting the DPB bit of TXFIFOSZ register (bit 4).
- DMA: If DMA is enabled for the endpoint, a DMA request will be generated whenever the endpoint is able to accept another packet in its FIFO. This feature allows the DMA controller to load packets into the FIFO without processor intervention.

  When DMA is enabled and DMAMODE bit of PERI_TXCSR is set, an endpoint interrupt is not generated for completion of the packet transfer. An endpoint interrupt is generated only in the error conditions.

3.1.2.1.1 Setup

In configuring a TX endpoint for bulk transactions, the TXMAXP register must be written with the maximum packet size (in bytes) for the endpoint. This value should be the same as the wMaxPacketSize field of the Standard Endpoint Descriptor for the endpoint and the PERI_TXCSR register should be set as shown in Table 2:

Table 2. PERI_TXCSR Register Bit Configuration for Bulk IN Transactions

Bit Position	Bit Field Name	Configuration
Bit 14	ISO	Cleared to 0 for bulk mode operation
Bit 13	MODE	Set to 1 to make sure the FIFO is enabled (only necessary if the FIFO is shared with an RX endpoint)
Bit 12	DMAEN	Set to 1 if DMA requests must be enabled
Bit 11	FRCDATATOG	Cleared to 0 to allow normal data toggle operations
Bit 10	DMAMODE	Set to 1 when DMA is enabled and EP interrupt is not needed for each packet transmission

When the endpoint is first configured (following a SET_CONFIGURATION or SET_INTERFACE command on Endpoint 0), the lower byte of PERI_TXCSR should be written to set the CLRDATATOG bit (bit 6). This will ensure that the data toggle (which is handled automatically by the controller) starts in the correct state.

Also if there are any data packets in the FIFO (indicated by the FIFONOTEMPTY bit (bit 1 of PERI_TXCSR) being set), they should be flushed by setting the FLUSHFIFO bit (bit 3 of PERI_TXCSR).

NOTE: It may be necessary to set this bit twice in succession if double buffering is enabled.

3.1.2.1.2 Operation

When data is to be transferred over a Bulk IN pipe, a data packet needs to be loaded into the FIFO and the PERI_TXCSR register written to set the TXPKTRDY bit (bit 0). When the packet has been sent, the TXPKTRDY bit will be cleared by the USB controller and an interrupt generated so that the next packet can be loaded into the FIFO. If double packet buffering is enabled, then after the first packet has been loaded and the TXPKTRDY bit set, the TXPKTRDY bit will immediately be cleared by the USB controller and an interrupt generated so that a second packet can be loaded into the FIFO. The software should operate in the same way, loading a packet when it receives an interrupt, regardless of whether double packet buffering is enabled or not.

In the general case, the packet size must not exceed the size specified by the lower 11 bits of the TXMAXP register. This part of the register defines the payload (packet size) for transfers over the USB and is required by the USB Specification to be either 8, 16, 32, 64 (Full-Speed or High-Speed) or 512 bytes (High-Speed only).

The host may determine that all the data for a transfer has been sent by knowing the total amount of data that is expected. Alternatively it may infer that all the data has been sent when it receives a packet which is smaller than the stated payload (TXMAXP [bit 10 : bit 0]). In the latter case, if the total size of the data block is a multiple of this payload, it will be necessary for the function to send a null packet after all the data has been sent. This is done by setting TXPKTRDY when the next interrupt is received, without loading any data into the FIFO.

If large blocks of data are being transferred, then the overhead of calling an interrupt service routine to load each packet can be avoided by using DMA.

3.1.2.1.3 Error Handling

If the software wants to shut down the Bulk IN pipe, it should set the SENDSTALL bit (bit 4 of PERI_TXCSR). When the controller receives the next IN token, it will send a STALL to the host, set the SENTSTALL bit (bit 5 of PERI_TXCSR) and generate an interrupt.

When the software receives an interrupt with the SENTSTALL bit (bit 5 of PERI_TXCSR) set, it should clear the SENTSTALL bit. It should however leave the SENDSTALL bit set until it is ready to re-enable the Bulk IN pipe.

NOTE: If the host failed to receive the STALL packet for some reason, it will send another IN token, so it is advisable to leave the SENDSTALL bit set until the software is ready to re-enable the Bulk IN pipe. When a pipe is re-enabled, the data toggle sequence should be restarted by setting the CLRDATATOG bit in the PERI_TXCSR register (bit 6).

3.1.2.2 Peripheral Mode: Bulk OUT Transactions

A Bulk OUT transaction is used to transfer non-periodic data from the host to the function controller.

The following optional features are available for use with an Rx endpoint used in peripheral mode for Bulk OUT transactions:

- Double packet buffering: When enabled, up to two packets can be stored in the FIFO on reception from the host. Double packet buffering is enabled by setting the DPB bit of the RXFIFOSZ register (bit 4).
- DMA: If DMA is enabled for the endpoint, a DMA request will be generated whenever the endpoint has a packet in its FIFO. This feature can be used to allow the DMA controller to unload packets from the FIFO without processor intervention.

  When DMA is enabled, endpoint interrupt will not be generated for completion of packet reception. Endpoint interrupt will be generated only in the error conditions.

3.1.2.2.1 Setup

In configuring an Rx endpoint for Bulk OUT transactions, the RXMAXP register must be written with the maximum packet size (in bytes) for the endpoint. This value should be the same as the wMaxPacketSize field of the Standard Endpoint Descriptor for the endpoint. In addition, the relevant interrupt enable bit in the INTRRXE register should be set (if an interrupt is required for this endpoint) and the PERI_RXCSR register should be set as shown in Table 3.

Table 3. PERI_RXCSR Register Bit Configuration for Bulk OUT Transactions

Bit Position	Bit Field Name	Configuration
Bit 14	ISO	Cleared to 0 to enable Bulk protocol
Bit 13	DMAEN	Set to 1 if a DMA request is required for this endpoint
Bit 12	DISNYET	Cleared to 0 to allow normal PING flow control. This will affect only high speed transactions.
Bit 11	DMAMODE	Always set this bit to 0

When the endpoint is first configured (following a SET_CONFIGURATION or SET_INTERFACE command on Endpoint 0), the lower byte of PERI_RXCSR should be written to set the CLRDATATOG bit (bit 7). This will ensure that the data toggle (which is handled automatically by the USB controller) starts in the correct state.

Also if there are any data packets in the FIFO (indicated by the RXPKTRDY bit (bit 0 of PERI_RXCSR) being set), they should be flushed by setting the FLUSHFIFO bit (bit 4 of PERI_RXCSR).

NOTE: It may be necessary to set this bit twice in succession if double buffering is enabled.

3.1.2.2.2 Operation

When a data packet is received by a Bulk Rx endpoint, the RXPKTRDY bit (bit 0 of PERI_RXCSR) is set and an interrupt is generated. The software should read the RXCOUNT register for the endpoint to determine the size of the data packet. The data packet should be read from the FIFO, then the RXPKTRDY bit should be cleared.

The packets received should not exceed the size specified in the RXMAXP register (as this should be the value set in the wMaxPacketSize field of the endpoint descriptor sent to the host). When a block of data larger than wMaxPacketSize needs to be sent to the function, it will be sent as multiple packets. All the packets will be wMaxPacketSize in size, except the last packet which will contain the residue. The software may use an application specific method of determining the total size of the block and hence when the last packet has been received. Alternatively it may infer that the entire block has been received when it receives a packet which is less than wMaxPacketSize in size. (If the total size of the data block is a multiple of wMaxPacketSize, a null data packet will be sent after the data to signify that the transfer is complete.)

In the general case, the application software will need to read each packet from the FIFO individually. If large blocks of data are being transferred, the overhead of calling an interrupt service routine to unload each packet can be avoided by using DMA.

3.1.2.2.3 Error Handling

If the software wants to shut down the Bulk OUT pipe, it should set the SENDSTALL bit (bit 5 of PERI_RXCSR). When the controller receives the next packet it will send a STALL to the host, set the SENTSTALL bit (bit 6 of PERI_RXCSR) and generate an interrupt.

When the software receives an interrupt with the SENTSTALL bit (bit 6 of PERI_RXCSR) set, it should clear this bit. It should however leave the SENDSTALL bit set until it is ready to re-enable the Bulk OUT pipe.

NOTE: If the host failed to receive the STALL packet for some reason, it will send another packet, so it is advisable to leave the SENDSTALL bit set until the software is ready to re-enable the Bulk OUT pipe. When a Bulk OUT pipe is re-enabled, the data toggle sequence should be restarted by setting the CLRDATATOG bit (bit 7) in the PERI_RXCSR register.

3.1.3 Interrupt Transactions

An Interrupt IN transaction uses the same protocol as a Bulk IN transaction and can be used the same way. Similarly, an Interrupt OUT transaction uses almost the same protocol as a Bulk OUT transaction and can be used the same way.

Tx endpoints in the USB controller have one feature for Interrupt IN transactions that they do not support in Bulk IN transactions. In Interrupt IN transactions, the endpoints support continuous toggle of the data toggle bit.

This feature is enabled by setting the FRCDATATOG bit in the PERI_TXCSR register (bit 11). When this bit is set, the controller will consider the packet as having been successfully sent and toggle the data bit for the endpoint, regardless of whether an ACK was received from the host.

Another difference is that interrupt endpoints do not support PING flow control. This means that the controller should never respond with a NYET handshake, only ACK/NAK/STALL. To ensure this, the DISNYET bit in the PERI_RXCSR register (bit 12) should be set to disable the transmission of NYET handshakes in high-speed mode.

Though DMA can be used with an interrupt OUT endpoint, it generally offers little benefit as interrupt endpoints are usually expected to transfer all their data in a single packet.

3.1.4 Isochronous Transactions

3.1.4.1 Isochronous IN Transactions

An Isochronous IN transaction is used to transfer periodic data from the function controller to the host.

The following optional features are available for use with a Tx endpoint used in Peripheral mode for Isochronous IN transactions:

- Double packet buffering: When enabled, up to two packets can be stored in the FIFO awaiting transmission to the host. Double packet buffering is enabled by setting the DPB bit of TXFIFOSZ register (bit 4).

  NOTE: Double packet buffering is generally advisable for Isochronous transactions in order to avoid Underrun errors as described in later section.

- DMA: If DMA is enabled for the endpoint, a DMA request will be generated whenever the endpoint is able to accept another packet in its FIFO. This feature allows the DMA controller to load packets into the FIFO without processor intervention.

  However, this feature is not particularly useful with Isochronous endpoints because the packets transferred are often not maximum packet size and the PERI_TXCSR register needs to be accessed following every packet to check for Underrun errors.

  When DMA is enabled and DMAMODE bit of PERI_TXCSR is set, endpoint interrupt will not be generated for completion of packet transfer. Endpoint interrupt will be generated only in the error conditions.

3.1.4.1.1 Setup

In configuring a Tx endpoint for Isochronous IN transactions, the TXMAXP register must be written with the maximum packet size (in bytes) for the endpoint. This value should be the same as the wMaxPacketSize field of the Standard Endpoint Descriptor for the endpoint. In addition, the relevant interrupt enable bit in the INTRTXE register should be set (if an interrupt is required for this endpoint) and the PERI_TXCSR register should be set as shown in Table 4.

Table 4. PERI_TXCSR Register Bit Configuration for Isochronous IN Transactions

Bit Position	Bit Field Name	Configuration
Bit 14	ISO	Set to 1 to enable Isochronous transfer protocol
Bit 13	MODE	Set to 1 to ensure the FIFO is enabled (only necessary if the FIFO is shared with an Rx endpoint).
Bit 12	DMAEN	Set to 1 if DMA Requests have to be enabled
Bit 11	FRCDATATOG	Ignored in Isochronous mode
Bit 10	DMAMODE	Set it to 1, when DMA is enabled and EP interrupt is not needed for each packet transmission

3.1.4.1.2 Operation

An Isochronous endpoint does not support data retries, so if data underrun is to be avoided, the data to be sent to the host must be loaded into the FIFO before the IN token is received. The host will send one IN token per frame (or microframe in High-speed mode), however the timing within the frame (or microframe) can vary. If an IN token is received near the end of one frame and then at the start of the next frame, there will be little time to reload the FIFO. For this reason, double buffering of the endpoint is usually necessary.

An interrupt is generated whenever a packet is sent to the host and the software may use this interrupt to load the next packet into the FIFO and set the TXPKTRDY bit in the PERI_TXCSR register (bit 0) in the same way as for a Bulk Tx endpoint. As the interrupt could occur almost any time within a frame(/microframe), depending on when the host has scheduled the transaction, this may result in irregular timing of FIFO load requests. If the data source for the endpoint is coming from some external hardware, it may be more convenient to wait until the end of each frame(/microframe) before loading the FIFO as this will minimize the requirement for additional buffering. This can be done by using either the SOF interrupt or the external SOF_PULSE signal from the controller to trigger the loading of the next data packet. The SOF_PULSE is generated once per frame(/microframe) when a SOF packet is received. (The controller also maintains an external frame(/microframe) counter so it can still generate a SOF_PULSE when the SOF packet has been lost.) The interrupts may still be used to set the TXPKTRDY bit in PERI_TXCSR (bit 0) and to check for data overruns/underruns.

Starting up a double-buffered Isochronous IN pipe can be a source of problems. Double buffering requires that a data packet is not transmitted until the frame(/microframe) after it is loaded. There is no problem if the function loads the first data packet at least a frame(/microframe) before the host sets up the pipe (and therefore starts sending IN tokens). But if the host has already started sending IN tokens by the time the first packet is loaded, the packet may be transmitted in the same frame(/microframe) as it is loaded, depending on whether it is loaded before, or after, the IN token is received. This potential problem can be avoided by setting the ISOUPDATE bit in the POWER register (bit 7). When this bit is set, any data packet loaded into an Isochronous Tx endpoint FIFO will not be transmitted until after the next SOF packet has been received, thereby ensuring that the data packet is not sent too early.

3.1.4.1.3 Error Handling

If the endpoint has no data in its FIFO when an IN token is received, it will send a null data packet to the host and set the UNDERRUN bit in the PERI_TXCSR register (bit 2). This is an indication that the software is not supplying data fast enough for the host. It is up to the application to determine how this error condition is handled.

If the software is loading one packet per frame(/microframe) and it finds that the TXPKTRDY bit in the PERI_TXCSR register (bit 0) is set when it wants to load the next packet, this indicates that a data packet has not been sent (perhaps because an IN token from the host was corrupted). It is up to the application how it handles this condition: it may choose to flush the unsent packet by setting the FLUSHFIFO bit in the PERI_TXCSR register (bit 3), or it may choose to skip the current packet.

3.1.4.2 Isochronous OUT Transactions

An Isochronous OUT transaction is used to transfer periodic data from the host to the function controller.

Following optional features are available for use with an Rx endpoint used in Peripheral mode for Isochronous OUT transactions:

- Double packet buffering: When enabled, up to two packets can be stored in the FIFO on reception from the host. Double packet buffering is enabled by setting the DPB bit of RXFIFOSZ register (bit 4).

  NOTE: Double packet buffering is generally advisable for Isochronous transactions in order to avoid Overrun errors.

- DMA: If DMA is enabled for the endpoint, a DMA request will be generated whenever the endpoint has a packet in its FIFO. This feature can be used to allow the DMA controller to unload packets from the FIFO without processor intervention.

  However, this feature is not particularly useful with Isochronous endpoints because the packets transferred are often not maximum packet size and the PERI_RXCSR register needs to be accessed following every packet to check for Overrun or CRC errors.

  When DMA is enabled, endpoint interrupt will not be generated for completion of packet reception. Endpoint interrupt will be generated only in the error conditions.

3.1.4.2.1 Setup

In configuring an Rx endpoint for Isochronous OUT transactions, the RXMAXP register must be written with the maximum packet size (in bytes) for the endpoint. This value should be the same as the wMaxPacketSize field of the Standard Endpoint Descriptor for the endpoint. In addition, the relevant interrupt enable bit in the INTRRXE register should be set (if an interrupt is required for this endpoint) and the PERI_RXCSR register should be set as shown in Table 5.

Table 5. PERI_RXCSR Register Bit Configuration for Isochronous OUT Transactions

Bit Position	Bit Field Name	Configuration
Bit 14	ISO	Set to 1 to enable isochronous protocol
Bit 13	DMAEN	Set to 1 if a DMA request is required for this endpoint
Bit 12	DISNYET	Ignored in isochronous transfers
Bit 11	DMAMODE	Always set this bit to 0

3.1.4.2.2 Operation

An Isochronous endpoint does not support data retries so, if a data overrun is to be avoided, there must be space in the FIFO to accept a packet when it is received. The host will send one packet per frame (or microframe in High-speed mode); however, the time within the frame can vary. If a packet is received near the end of one frame(/microframe) and another arrives at the start of the next frame, there will be little time to unload the FIFO. For this reason, double buffering of the endpoint is usually necessary.

An interrupt is generated whenever a packet is received from the host and the software may use this interrupt to unload the packet from the FIFO and clear the RXPKTRDY bit in the PERI_RXCSR register (bit 0) in the same way as for a Bulk Rx endpoint. As the interrupt could occur almost any time within a frame(/microframe), depending on when the host has scheduled the transaction, the timing of FIFO unload requests will probably be irregular. If the data sink for the endpoint is going to some external hardware, it may be better to minimize the requirement for additional buffering by waiting until the end of each frame(/microframe) before unloading the FIFO. This can be done by using either the SOF interrupt or the external SOF_PULSE signal from the controller to trigger the unloading of the data packet. The SOF_PULSE is generated once per frame(/microframe) when a SOF packet is received. (The controller also maintains an external frame(/microframe) counter so it can still generate a SOF_PULSE when the SOF packet has been lost.) The interrupts may still be used to clear the RXPKTRDY bit in PERI_RXCSR and to check for data overruns/underruns.

3.1.4.2.3 Error Handling

If there is no space in the FIFO to store a packet when it is received from the host, the OVERRUN bit in the PERI_RXCSR register (bit 2) will be set. This is an indication that the software is not unloading data fast enough for the host. It is up to the application to determine how this error condition is handled.

If the controller finds that a received packet has a CRC error, it will still store the packet in the FIFO and set the RXPKTRDY bit (bit 0 of PERI_RXCSR) and the DATAERROR bit (bit 3 of PERI_RXCSR). It is left up to the application how this error condition is handled.

3.2 USB Controller Host Mode Operation

- *Entry into Suspend mode*. When operating as a host, the controller can be prompted to enter Suspend mode by setting the SUSPENDM bit in the POWER register. When this bit is set, the controller will complete the current transaction then stop the transaction scheduler and frame counter. No further transactions will be started and no SOF packets will be generated. If the ENSUSPM bit (bit 0 of POWER register) is set, PHY will go into low-power mode when the controller enters Suspend mode.
- *Sending Resume Signaling.* When the application requires the controller to leave Suspend mode, it must clear the SUSPENDM bit in the POWER register (bit 1), set the RESUME bit (bit 2) and leave it set for 20ms. While the RESUME bit is high, the controller will generate Resume signaling on the bus. After 20 ms, the application should clear the Resume bit, at which point the frame counter and transaction scheduler will be started.
- *Responding to Remote Wake-up.* If Resume signaling is detected from the target while the controller is in Suspend mode, the PHY will be brought out of low-power mode. The controller will then exit Suspend mode and automatically set the RESUME bit in the POWER register (bit 2) to take over generating the Resume signaling from the target. If the Resume interrupt is enabled, an interrupt will be generated.
- *Reset Signaling.* If the RESET bit in the POWER register (bit 3) is set while the controller is in Host mode, it will generate Reset signaling on the bus. If the HSENAB bit in the POWER register (bit 5) was set, it will also try to negotiate for high-speed operation. The software should keep the RESET bit set for at least 20 ms to ensure correct resetting of the target device. After the software has cleared the bit, the controller will start its frame counter and transaction scheduler. Whether high-speed operation is selected will be indicated by HSMODE bit of POWER register (bit 4).

3.2.1 Host Mode: Control Transactions

Host Control Transactions are conducted through Endpoint 0 and the software is required to handle all the Standard Device Requests that may be sent or received via Endpoint 0 (as described in Universal Serial Bus Specification, Revision 2.0, Chapter 9).

As for a USB peripheral device, there are three categories of Standard Device Requests to be handled: Zero Data Requests (in which all the information is included in the command), Write Requests (in which the command will be followed by additional data), and Read Requests (in which the device is required to send data back to the host).

1. Zero Data Requests consist of a SETUP command followed by an IN Status Phase
2. Write Requests consist of a SETUP command, followed by an OUT Data Phase which is in turn followed by an IN Status Phase
3. Read Requests consist of a SETUP command, followed by an IN Data Phase which is in turn followed by an OUT Status Phase

A timeout may be set to limit the length of time for which the controller will retry a transaction which is continually NAKed by the target. This limit can be between 2 and 215 frames/ microframes and is set through the HOST_NAKLIMIT0 register. The following sections describe the CPU actions required for these different types of requests by examining the steps to take in the different Control Transaction phases.

3.2.1.1 Setup Phase

For the SETUP Phase of a control transaction (Figure 9), the software driving the US host device needs to:

1. Load the 8 bytes of the required Device request command into the Endpoint 0 FIFO.
2. Set SETUPPKT and TXPKTRDY (bits 3 and 1 of HOST_CSR0, respectively).

   NOTE: These bits must be set together.

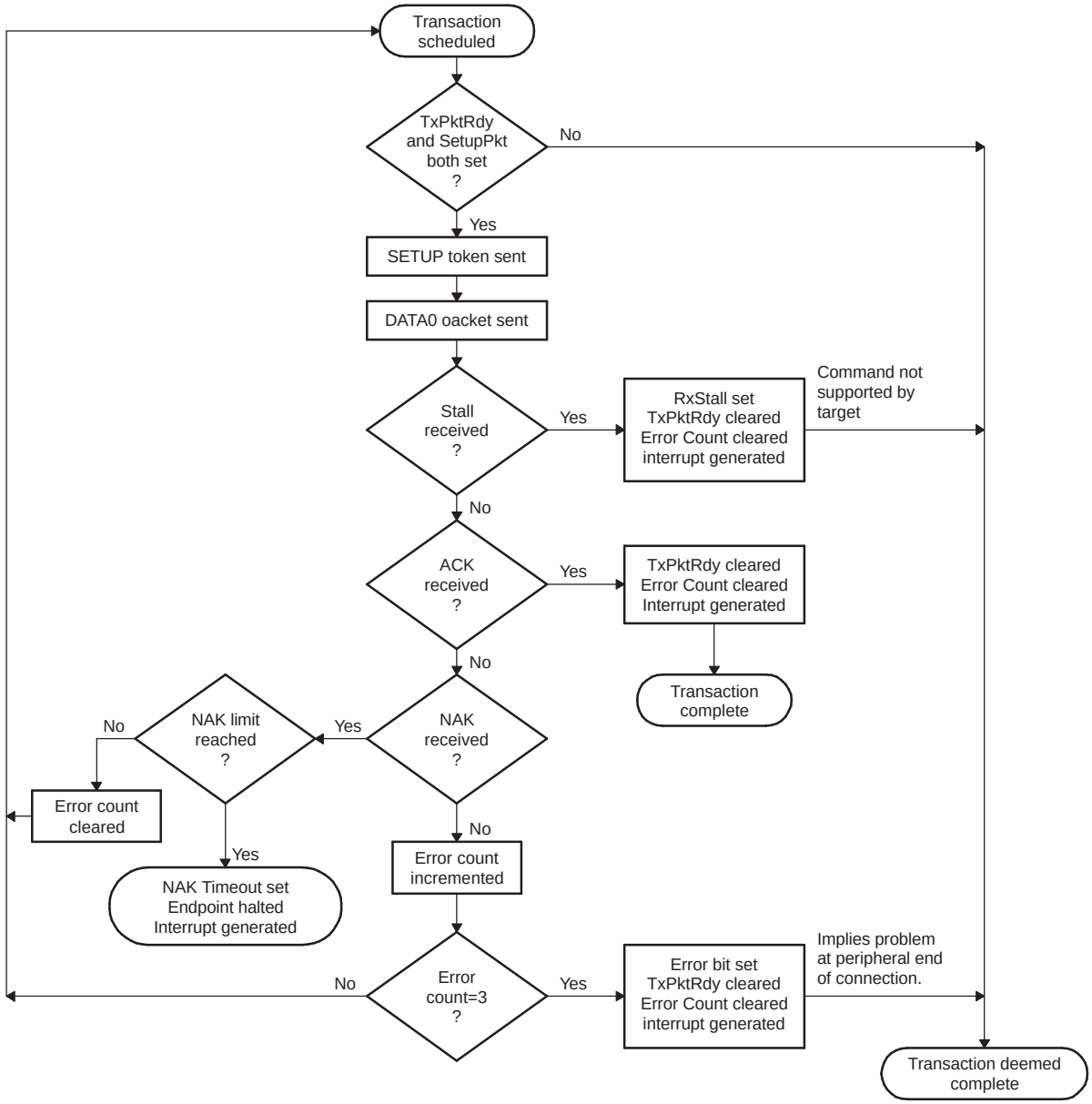
   The controller then proceeds to send a SETUP token followed by the 8-byte command to Endpoint 0 of the addressed device, retrying as necessary. (On errors, controller retries the transaction three times.)
3. At the end of the attempt to send the data, the controller will generate an Endpoint 0 interrupt. The software should then read HOST_CSR0 to establish whether the RXSTALL bit (bit 2), the ERROR bit (bit 4) or the NAK_TIMEOUT bit (bit 7) has been set.

   If RXSTALL is set, it indicates that the target did not accept the command (e.g., because it is not supported by the target device) and so has issued a STALL response.

   If ERROR is set, it means that the controller has tried to send the SETUP Packet and the following data packet three times without getting any response.

   If NAK_TIMEOUT is set, it means that the controller has received a NAK response to each attempt to send the SETUP packet, for longer than the time set in HOST_NAKLIMIT0. The controller can then be directed either to continue trying this transaction (until it times out again) by clearing the NAK_TIMEOUT bit or to abort the transaction by flushing the FIFO before clearing the NAK_TIMEOUT bit.
4. If none of RXSTALL, ERROR or NAK_TIMEOUT is set, the SETUP Phase has been correctly ACKed and the software should proceed to the following IN Data Phase, OUT Data Phase or IN Status Phase specified for the particular Standard Device Request.

Figure 9. Setup Phase of a Control Transaction Flow Chart



3.2.1.2 IN Data Phase

For the IN Data Phase of a control transaction (Figure 10), the software driving the USB host device needs to:

1. Set REQPKT bit of HOST_CSR0 (bit 5).
2. Wait while the controller sends the IN token and receives the required data back.
3. When the controller generates the Endpoint 0 interrupt, read HOST_CSR0 to establish whether the RXSTALL bit (bit 2), the ERROR bit (bit 4), the NAK_TIMEOUT bit (bit 7) or RXPKTRDY bit (bit 0) has been set.

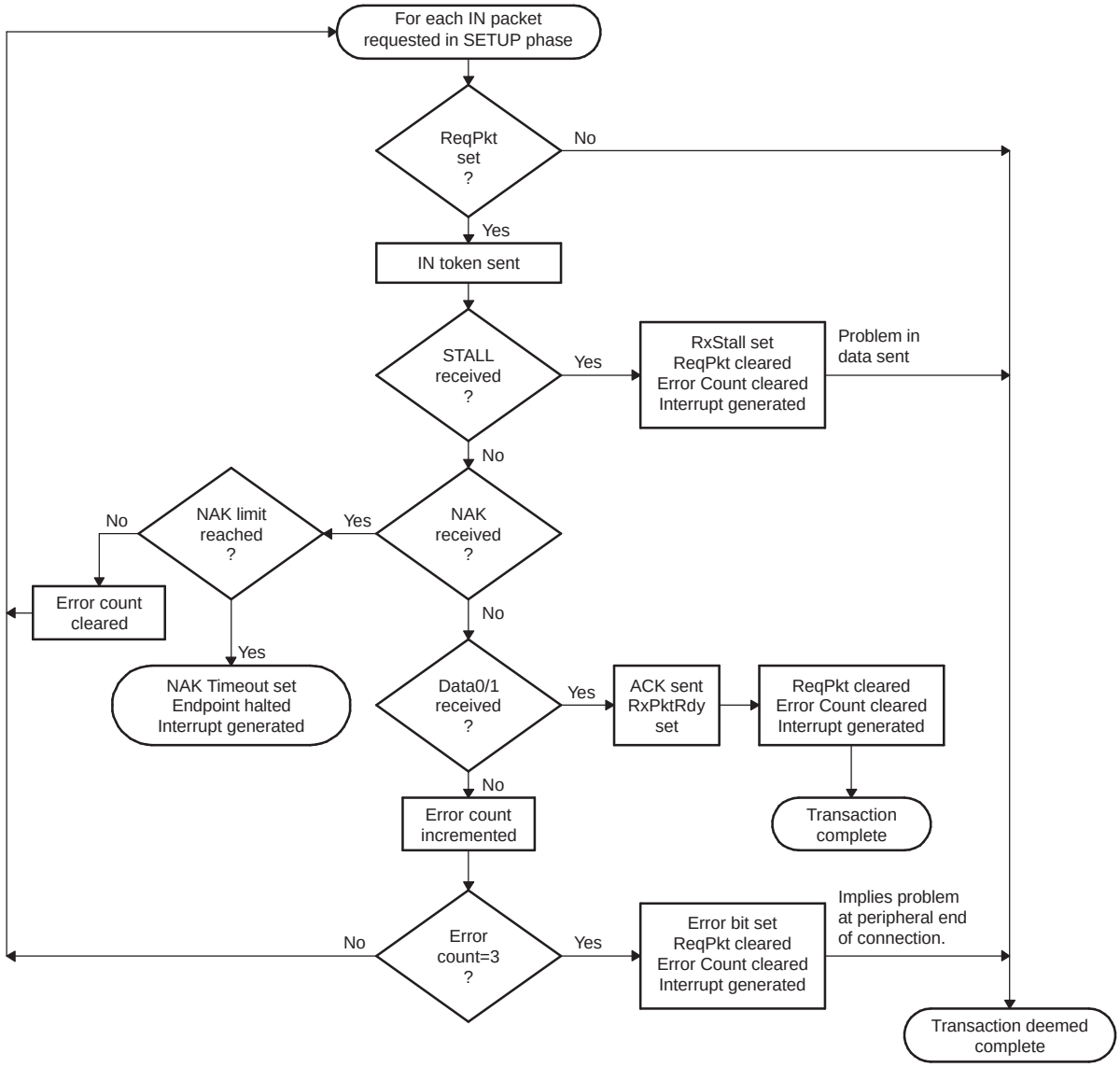
   If RXSTALL is set, it indicates that the target has issued a STALL response.

   If ERROR is set, it means that the controller has tried to send the required IN token three times without getting any response.

   If NAK_TIMEOUT bit is set, it means that the controller has received a NAK response to each attempt to send the IN token, for longer than the time set in HOST_NAKLIMIT0. The controller can then be directed either to continue trying this transaction (until it times out again) by clearing the NAK_TIMEOUT bit or to abort the transaction by clearing REQPKT before clearing the NAK_TIMEOUT bit.
4. If RXPKTRDY has been set, the software should read the data from the Endpoint 0 FIFO, then clear RXPKTRDY.
5. If further data is expected, the software should repeat Steps 1-4.

When all the data has been successfully received, the CPU should proceed to the OUT Status Phase of the Control Transaction.

Figure 10. IN Data Phase Flow Chart



3.2.1.3 OUT Data Phase

For the OUT Data Phase of a control transaction (Figure 11), the software driving the USB host device needs to:

1. Load the data to be sent into the endpoint 0 FIFO.
2. Set the TXPKTRDY bit of HOST_CSR0 (bit 1). The controller then proceeds to send an OUT token followed by the data from the FIFO to Endpoint 0 of the addressed device, retrying as necessary.
3. At the end of the attempt to send the data, the controller will generate an Endpoint 0 interrupt. The software should then read HOST_CSR0 to establish whether the RXSTALL bit (bit 2), the ERROR bit (bit 4) or the NAK_TIMEOUT bit (bit 7) has been set.

   If RXSTALL bit is set, it indicates that the target has issued a STALL response.

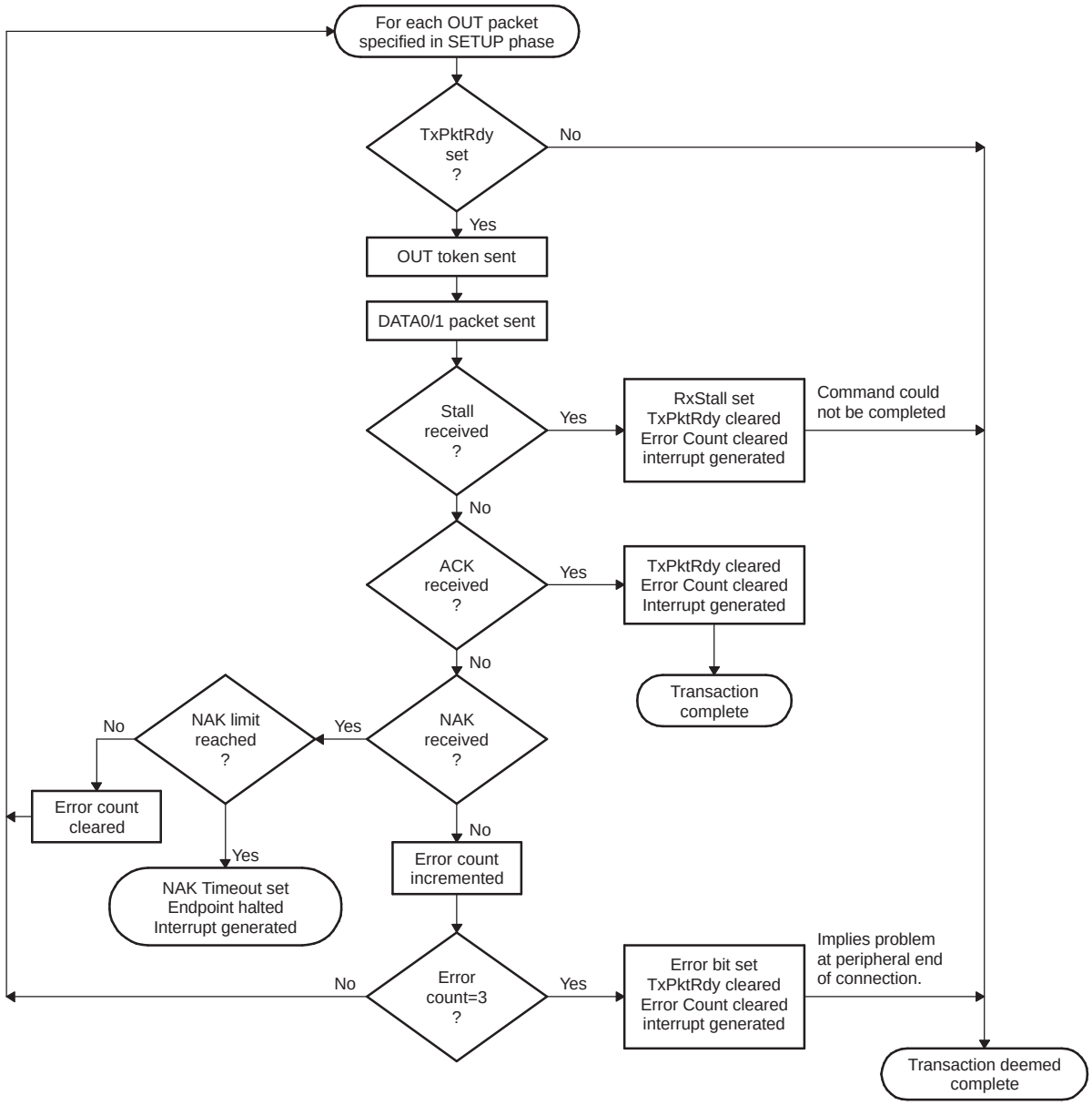
   If ERROR bit is set, it means that the controller has tried to send the OUT token and the following data packet three times without getting any response.

   If NAK_TIMEOUT is set, it means that the controller has received a NAK response to each attempt to send the OUT token, for longer than the time set in the HOST_NAKLIMIT0 register. The controller can then be directed either to continue trying this transaction (until it times out again) by clearing the NAK_TIMEOUT bit or to abort the transaction by flushing the FIFO before clearing the NAK_TIMEOUT bit.

   If none of RXSTALL, ERROR or NAKLIMIT is set, the OUT data has been correctly ACKed.
4. If further data needs to be sent, the software should repeat Steps 1-3.

When all the data has been successfully sent, the software should proceed to the IN Status Phase of the Control Transaction.

Figure 11. OUT Data Phase Flow Chart



3.2.1.4 IN Status Phase (following SETUP Phase or OUT Data Phase)

For the IN Status Phase of a Control Transaction (Figure 12), the software driving the USB Host device needs to:

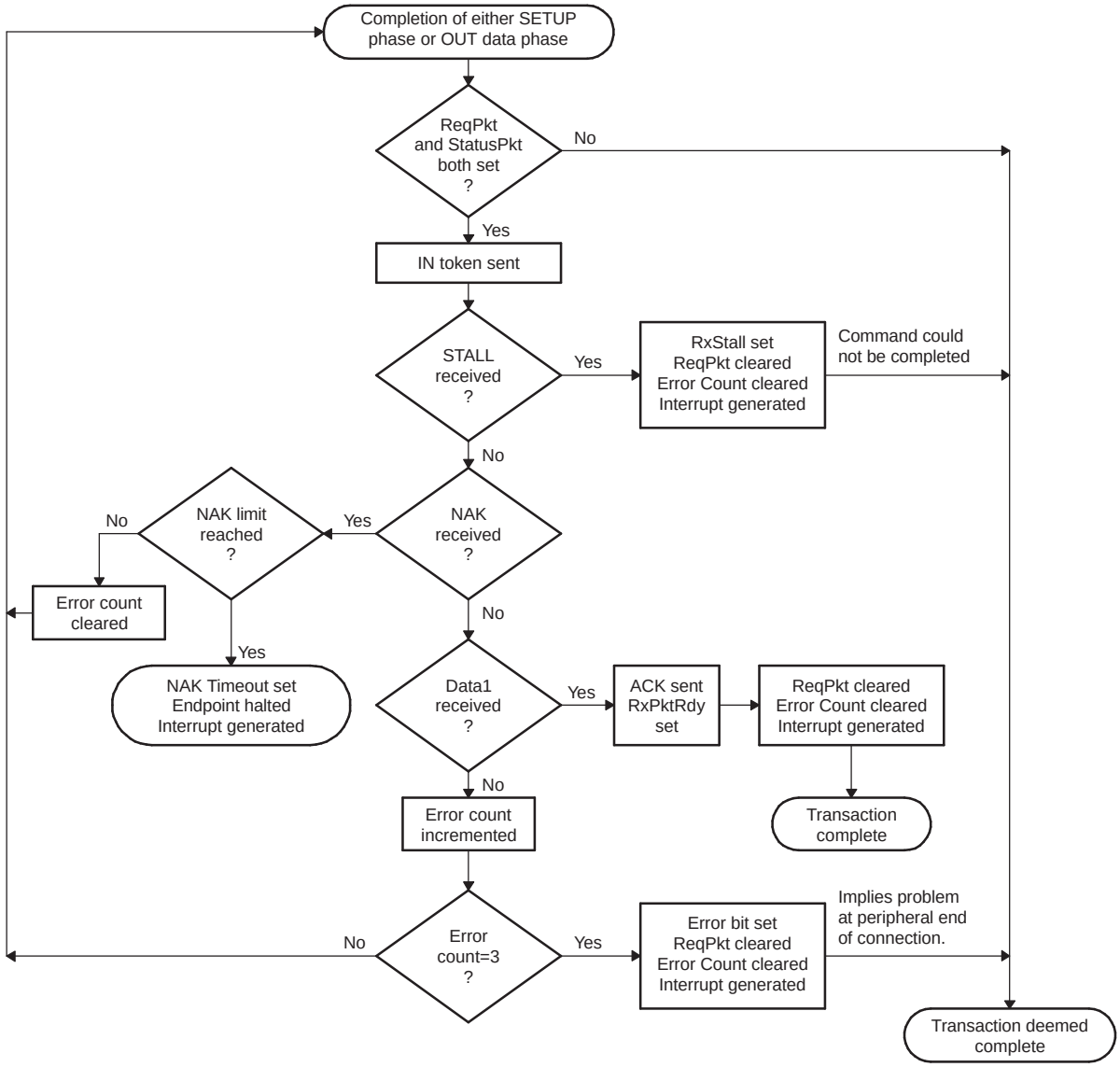
1. Set the STATUSPKT and REQPKT bits of HOST_CSR0 (bit 6 and bit 5, respectively).
2. Wait while the controller sends an IN token and receives a response from the USB peripheral device.
3. When the controller generates the Endpoint 0 interrupt, read HOST_CSR0 to establish whether the RXSTALL bit (bit 2), the ERROR bit (bit 4), the NAK_TIMEOUT bit (bit 7) or RXPKTRDY bit (bit 0) has been set.

   If RXSTALL bit is set, it indicates that the target could not complete the command and so has issued a STALL response.

   If ERROR bit is set, it means that the controller has tried to send the required IN token three times without getting any response.

   If NAK_TIMEOUT bit is set, it means that the controller has received a NAK response to each attempt to send the IN token, for longer than the time set in the HOST_NAKLIMIT0 register. The controller can then be directed either to continue trying this transaction (until it times out again) by clearing the NAK_TIMEOUT bit or to abort the transaction by clearing REQPKT bit and STATUSPKT bit before clearing the NAK_TIMEOUT bit.
4. If RxPktRdy has been set, the CPU should simply clear RxPktRdy.

Figure 12. Completion of SETUP or OUT Data Phase Flow Chart



3.2.1.5 OUT Status Phase (following IN Data Phase)

For the OUT Status Phase of a control transaction (Figure 13), the CPU driving the host device needs to:

1. Set STATUSPKT and TXPKTRDY bits of HOST_CSR0 (bit 6 and bit 1, respectively).

   NOTE: These bits need to be set together.

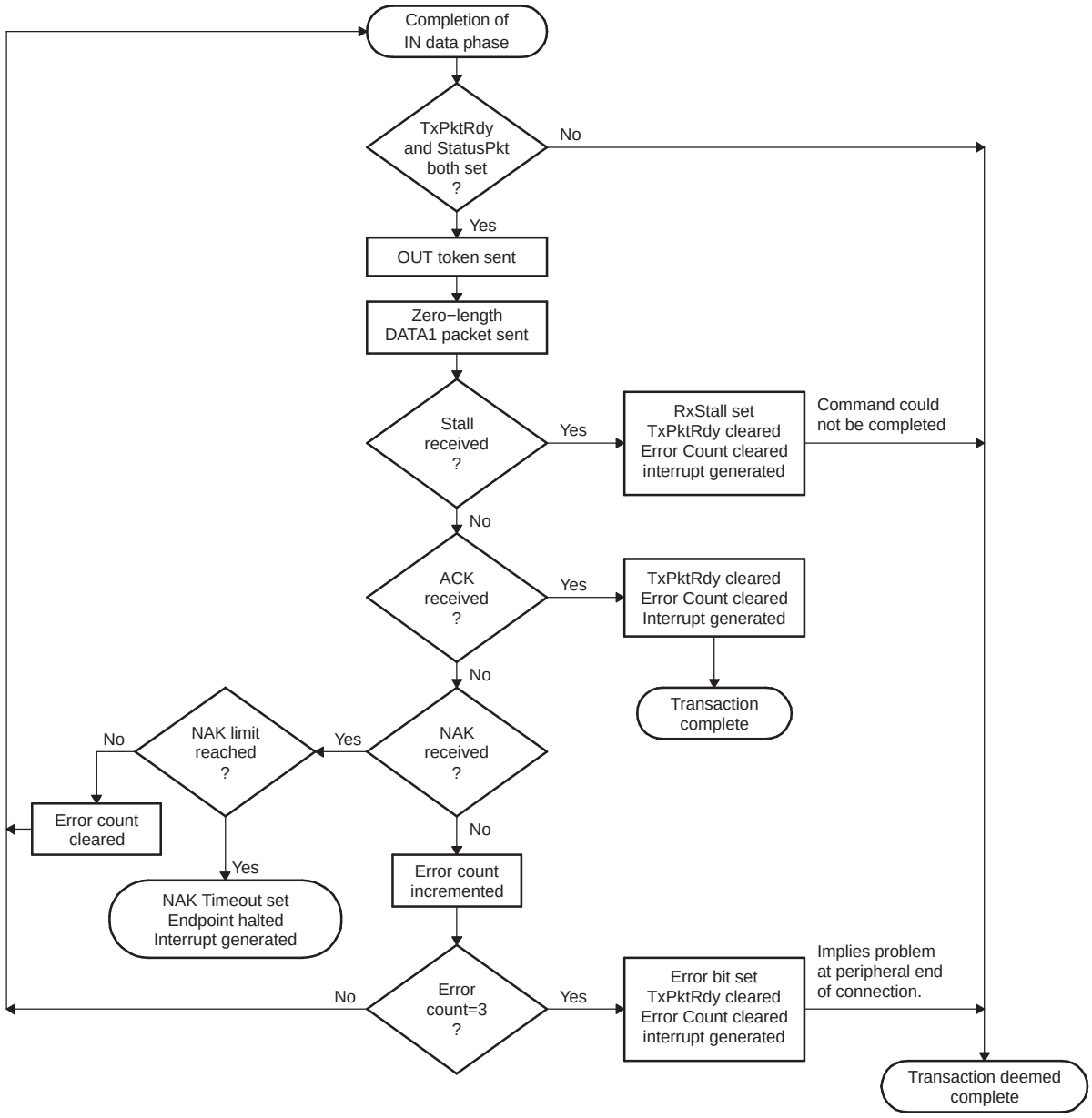
2. Wait while the controller sends the OUT token and a zero-length DATA1 packet.
3. At the end of the attempt to send the data, the controller will generate an Endpoint 0 interrupt. The software should then read HOST_CSR0 to establish whether the RXSTALL bit (bit 2), the ERROR bit (bit 4) or the NAK_TIMEOUT bit (bit 7) has been set.

   If RXSTALL bit is set, it indicates that the target could not complete the command and so has issued a STALL response.

   If ERROR bit is set, it means that the controller has tried to send the STATUS Packet and the following data packet three times without getting any response.

   If NAK_TIMEOUT bit is set, it means that the controller has received a NAK response to each attempt to send the IN token, for longer than the time set in the HOST_NAKLIMIT0 register. The controller can then be directed either to continue trying this transaction (until it times out again) by clearing the NAK_TIMEOUT bit or to abort the transaction by flushing the FIFO before clearing the NAK_TIMEOUT bit.
4. If none of RXSTALL, ERROR or NAK_TIMEOUT bits is set, the STATUS Phase has been correctly ACKed.

Figure 13. Completion of IN Data Phase Flow Chart



3.2.2 Bulk Transactions

3.2.2.1 Host Mode: Bulk IN Transactions

A Bulk IN transaction may be used to transfer non-periodic data from the external USB peripheral to the host.

The following optional features are available for use with an Rx endpoint used in host mode to receive the data:

- Double packet buffering: When enabled, up to two packets can be stored in the FIFO on reception from the host. This allows that one packet can be received while another is being read. Double packet buffering is enabled by setting the DPB bit of RXFIFOSZ register (bit 4).
- DMA: If DMA is enabled for the endpoint, a DMA request will be generated whenever the endpoint has a packet in its FIFO. This feature can be used to allow the DMA controller to unload packets from the FIFO without processor intervention.

  When DMA is enabled, endpoint interrupt will not be generated for completion of packet reception. Endpoint interrupt will be generated only in the error conditions.
- AutoRequest: When the AutoRequest feature is enabled, the REQPKT bit of HOST_RXCSR (bit 5) will be automatically set when the RXPKTRDY bit is cleared.

  This feature is applicable only when DMA is enabled. To enable AutoRequest feature, set the AUTOREQ register for the DMA channel associated for the endpoint.

3.2.2.1.1 Setup

Before initiating any Bulk IN Transactions in Host mode:

- The target function address needs to be set in the RXFUNCADDR register for the selected controller endpoint. (RXFUNCADDR register is available for all endpoints from EP0 to EP4.)
- The HOST_RXTYPE register for the endpoint that is to be used needs to be programmed as follows:
  - Operating speed in the SPEED bit field (bits 7 and 6).
  - Set 10 (binary value) in the PROT field for bulk transfer.
  - Endpoint Number of the target device in RENDPN field. This is the endpoint number contained in the Rx endpoint descriptor returned by the target device during enumeration.
- The RXMAXP register for the controller endpoint must be written with the maximum packet size (in bytes) for the transfer. This value should be the same as the wMaxPacketSize field of the Standard Endpoint Descriptor for the target endpoint.
- The HOST_RXINTERVAL register needs to be written with the required value for the NAK limit (2 - 215 frames/microframes), or cleared to 0 if the NAK timeout feature is not required.
- The relevant interrupt enable bit in the INTRRXE register should be set (if an interrupt is required for this endpoint).
- The following bits of HOST_RXCSR register should be set as shown below:
  - Set DMAEN (bit 13) to 1 if a DMA request is required for this endpoint.
  - Clear DSINYET (bit 12) to 0 to allow normal PING flow control. This will affect only High Speed transactions.
  - Always clear DMAMODE (bit 11) to 0.
- If DMA is enabled, the AUTOREQ register can be set for generating IN tokens automatically after receiving the data. Set the bit field RXn_AUTOREQ (where n is the endpoint number) with binary value 01 or 11.

When the endpoint is first configured, the endpoint data toggle should be cleared to 0 either by using the DATATOGWREN and DATATOG bits of HOST_RXCSR (bit 10 and bit 9) to toggle the current setting or by setting the CLRDATATOG bit of HOST_RXCSR (bit 7). This will ensure that the data toggle (which is handled automatically by the controller) starts in the correct state. Also if there are any data packets in the FIFO (indicated by the RXPKTRDY bit (bit 0 of HOST_RXCSR) being set), they should be flushed by setting the FLUSHFIFO bit of HOST_RXCSR (bit 4).

NOTE: It may be necessary to set this bit twice in succession if double buffering is enabled.

3.2.2.1.2 Operation

When Bulk data is required from the USB peripheral device, the software should set the REQPKT bit in the corresponding HOST_RXCSR register (bit 5). The controller will then send an IN token to the selected peripheral endpoint and waits for data to be returned.

If data is correctly received, RXPKTRDY bit of HOST_RXCSR (bit 0) is set. If the USB peripheral device responds with a STALL, RXSTALL bit (bit 6 of HOST_RXCSR) is set. If a NAK is received, the controller tries again and continues to try until either the transaction is successful or the POLINTVL_NAKLIMIT set in the HOST_RXINTERVAL register is reached. If no response at all is received, two further attempts are made before the controller reports an error by setting the ERROR bit of HOST_RXCSR (bit 2).

The controller then generates the appropriate endpoint interrupt, whereupon the software should read the corresponding HOST_RXCSR register to determine whether the RXPKTRDY, RXSTALL, ERROR or DATAERR_NAKTIMEOUT bit is set and act accordingly. If the DATAERR_NAKTIMEOUT bit is set, the controller can be directed either to continue trying this transaction (until it times out again) by clearing the DATAERR_NAKTIMEOUT bit or to abort the transaction by clearing REQPKT bit before clearing the DATAERR_NAKTIMEOUT bit.

The packets received should not exceed the size specified in the RXMAXP register (as this should be the value set in the wMaxPacketSize field of the endpoint descriptor sent to the host).

In the general case, the application software will need to read each packet from the FIFO individually. If large blocks of data are being transferred, the overhead of calling an interrupt service routine to unload each packet can be avoided by using DMA.

3.2.2.1.3 Error Handling

If the target wants to shut down the Bulk IN pipe, it will send a STALL response to the IN token. This will result in the RXSTALL bit of HOST_RXCSR (bit 6) being set.

3.2.2.2 Bulk OUT Transactions

A Bulk OUT transaction may be used to transfer non-periodic data from the host to the USB peripheral.

Following optional features are available for use with a Tx endpoint used in Host mode to transmit this data:

- Double packet buffering: When enabled, up to two packets can be stored in the FIFO awaiting transmission to the peripheral device. Double packet buffering is enabled by setting the DPB bit of TXFIFOSZ register (bit 4).
- DMA: If DMA is enabled for the endpoint, a DMA request will be generated whenever the endpoint is able to accept another packet in its FIFO. This feature can be used to allow the DMA controller to load packets into the FIFO without processor intervention.

  When DMA is enabled and DMAMODE bit in HOST_TXCSR register is set, an endpoint interrupt will not be generated for completion of packet reception. An endpoint interrupt will be generated only in the error conditions.

3.2.2.2.1 Setup

Before initiating any bulk OUT transactions:

- The target function address needs to be set in the TXFUNCADDR register for the selected controller endpoint. (TXFUNCADDR register is available for all endpoints from EP0 to EP4.)
- The HOST_TXTYPE register for the endpoint that is to be used needs to be programmed as follows:
  - Operating speed in the SPEED bit field (bits 7 and 6).
  - Set 10b in the PROT field for bulk transfer.
  - Endpoint Number of the target device in TENDPN field. This is the endpoint number contained in the OUT(Tx) endpoint descriptor returned by the target device during enumeration.
- The TXMAXP register for the controller endpoint must be written with the maximum packet size (in bytes) for the transfer. This value should be the same as the wMaxPacketSize field of the Standard Endpoint Descriptor for the target endpoint.

- The HOST_TXINTERVAL register needs to be written with the required value for the NAK limit (2-215 frames/microframes), or cleared to 0 if the NAK timeout feature is not required.
- The relevant interrupt enable bit in the INTRTXE register should be set (if an interrupt is required for this endpoint).
- The following bits of HOST_TXCSR register should be set as shown below:
  - Set the MODE bit (bit 13) to 1 to ensure the FIFO is enabled (only necessary if the FIFO is shared with an Rx endpoint).
  - Set the DMAEN bit (bit 12) to 1 if a DMA request is required for this endpoint.
  - Clear the FRCDATATOG bit (bit 11) to 0 to allow normal data toggle operations.
  - Set the DMAMODE bit (bit 10) to 1 when DMA is enabled and the endpoint interrupt is not needed for each packet transmission.

When the endpoint is first configured, the endpoint data toggle should be cleared to 0 either by using the DATATOGWREN bit and DATATOG bit of HOST_TXCSR (bit 9 and bit 8) to toggle the current setting or by setting the CLRDATATOG bit of HOST_TXCSR (bit 6). This will ensure that the data toggle (which is handled automatically by the controller) starts in the correct state. Also, if there are any data packets in the FIFO (indicated by the FIFONOTEMPTY bit of HOST_TXCSR register (bit 1) being set), they should be flushed by setting the FLUSHFIFO bit (bit 3 of HOST_TXCSR).

NOTE: It may be necessary to set this bit twice in succession if double buffering is enabled.

3.2.2.2.2 Operation

When Bulk data is required to be sent to the USB peripheral device, the software should write the first packet of the data to the FIFO (or two packets if double-buffered) and set the TXPKTRDY bit in the corresponding HOST_TXCSR register (bit 0). The controller will then send an OUT token to the selected peripheral endpoint, followed by the first data packet from the FIFO.

If data is correctly received by the peripheral device, an ACK should be received whereupon the controller will clear TXPKTRDY bit of HOST_TXCSR (bit 0). If the USB peripheral device responds with a STALL, the RXSTALL bit (bit 5) of HOST_TXCSR is set. If a NAK is received, the controller tries again and continues to try until either the transaction is successful or the NAK limit set in the HOST_TXINTERVAL register is reached. If no response at all is received, two further attempts are made before the controller reports an error by setting ERROR bit in HOST_TXCSR (bit 2).

The controller then generates the appropriate endpoint interrupt, whereupon the software should read the corresponding HOST_TXCSR register to determine whether the RXSTALL (bit 5), ERROR (bit 2) or NAK_TIMEOUT (bit 7) bit is set and act accordingly. If the NAK_TIMEOUT bit is set, the controller can be directed either to continue trying this transaction (until it times out again) by clearing the NAK_TIMEOUT bit or to abort the transaction by flushing the FIFO before clearing the NAK_TIMEOUT bit.

If large blocks of data are being transferred, then the overhead of calling an interrupt service routine to load each packet can be avoided by using DMA.

3.2.2.2.3 Error Handling

If the target wants to shut down the Bulk OUT pipe, it will send a STALL response. This is indicated by the RXSTALL bit of HOST_TXCSR register (bit 5) being set.

3.2.3 Host Mode: Interrupt Transactions

When the controller is operating as the host, interactions with an Interrupt endpoint on the USB peripheral device are handled in very much the same way as the equivalent Bulk transactions (described in previous sections).

The principal difference as far as operational steps are concerned is that PROT field of HOST_RXTYPE and HOST_TXTYPE (bits 5:4) need to be set (binary value) to represent an Interrupt transaction.

The required polling interval also needs to be set in the HOST_RXINTERVAL and HOST_TXINTERVAL registers.

3.2.4 Isochronous Transactions

3.2.4.1 Host Mode: Isochronous IN Transactions

An Isochronous IN transaction is used to transfer periodic data from the USB peripheral to the host.

The following optional features are available for use with an Rx endpoint used in Host mode to receive this data:

- Double packet buffering: When enabled, up to two packets can be stored in the FIFO on reception from the host. This allows that one packet can be received while another is being read. Double packet buffering is enabled by setting the DPB bit of RXFIFOSZ register (bit 4).
- DMA: If DMA is enabled for the endpoint, a DMA request will be generated whenever the endpoint has a packet in its FIFO. This feature can be used to allow the DMA controller to unload packets from the FIFO without processor intervention. However, this feature is not particularly useful with isochronous endpoints because the packets transferred are often not maximum packet size.

  When DMA is enabled, endpoint interrupt will not be generated for completion of packet reception. Endpoint interrupt will be generated only in the error conditions.
- AutoRequest: When the AutoRequest feature is enabled, the REQPKT bit of HOST_RXCSR (bit 5) will be automatically set when the RXPKTRDY bit is cleared.

  This feature is applicable only when DMA is enabled. To enable AutoRequest feature, set the AUTOREQ register for the DMA channel associated for the endpoint.

3.2.4.1.1 Setup

Before initiating an Isochronous IN Transactions in Host mode:

- The target function address needs to be set in the RXFUNCADDR register for the selected controller endpoint (RXFUNCADDR register is available for all endpoints from EP0 to EP4).
- The HOST_RXTYPE register for the endpoint that is to be used needs to be programmed as follows:
  - Operating speed in the SPEED bit field (bits 7 and 6).
  - Set 01 (binary value) in the PROT field for isochronous transfer.
  - Endpoint Number of the target device in RENDPN field. This is the endpoint number contained in the Rx endpoint descriptor returned by the target device during enumeration.
- The RXMAXP register for the controller endpoint must be written with the maximum packet size (in bytes) for the transfer. This value should be the same as the wMaxPacketSize field of the Standard Endpoint Descriptor for the target endpoint.
- The HOST_RXINTERVAL register needs to be written with the required transaction interval (usually one transaction per frame/microframe).
- The relevant interrupt enable bit in the INTRRXE register should be set (if an interrupt is required for this endpoint).
- The following bits of HOST_RXCSR register should be set as shown below:
  - Set the DMAEN bit (bit 13) to 1 if a DMA request is required for this endpoint.
  - Clear the DISNYET it (bit 12) to 0 to allow normal PING flow control. This will only affect High Speed transactions.
  - Always clear the DMAMODE bit (bit 11) to 0.
- If DMA is enabled, AUTOREQ register can be set for generating IN tokens automatically after receiving the data. Set the bit field RX*n*_AUTOREQ (where *n* is the endpoint number) with binary value 01 or 11.

3.2.4.1.2 Operation

The operation starts with the software setting REQPKT bit of HOST_RXCSR (bit 5). This causes the controller to send an IN token to the target.

When a packet is received, an interrupt is generated which the software may use to unload the packet from the FIFO and clear the RXPKTRDY bit in the HOST_RXCSR register (bit 0) in the same way as for a Bulk Rx endpoint. As the interrupt could occur almost any time within a frame(/microframe), the timing of FIFO unload requests will probably be irregular. If the data sink for the endpoint is going to some external hardware, it may be better to minimize the requirement for additional buffering by waiting until the end of each frame before unloading the FIFO. This can be done by using the SOF_PULSE signal from the controller to trigger the unloading of the data packet. The SOF_PULSE is generated once per frame(/microframe). The interrupts may still be used to clear the RXPKTRDY bit in HOST_RXCSR.

3.2.4.1.3 Error Handling

If a CRC or bit-stuff error occurs during the reception of a packet, the packet will still be stored in the FIFO but the DATAERR_NAKTIMEOUT bit of HOST_RXCSR (bit 3) is set to indicate that the data may be corrupt.

3.2.4.2 Host Mode: Isochronous Out Transactions

An Isochronous OUT transaction may be used to transfer periodic data from the host to the USB peripheral.

Following optional features are available for use with a Tx endpoint used in Host mode to transmit this data:

- Double packet buffering: When enabled, up to two packets can be stored in the FIFO awaiting transmission to the peripheral device. Double packet buffering is enabled by setting the DPB bit of TXFIFOSZ register (bit 4).
- DMA: If DMA is enabled for the endpoint, a DMA request will be generated whenever the endpoint is able to accept another packet in its FIFO. This feature can be used to allow the DMA controller to load packets into the FIFO without processor intervention.

  However, this feature is not particularly useful with isochronous endpoints because the packets transferred are often not maximum packet size.

  When DMA is enabled and DMAMODE bit in HOST_TXCSR register is set, endpoint interrupt will not be generated for completion of packet reception. Endpoint interrupt will be generated only in the error conditions.

3.2.4.2.1 Setup

Before initiating any Isochronous OUT transactions:

- The target function address needs to be set in the TXFUNCADDR register for the selected controller endpoint (TXFUNCADDR register is available for all endpoints from EP0 to EP4).
- The HOST_TXTYPE register for the endpoint that is to be used needs to be programmed as follows:
  - Operating speed in the SPEED bit field (bits 7 and 6).
  - Set 01 (binary value) in the PROT field for isochronous transfer.
  - Endpoint Number of the target device in TENDPN field. This is the endpoint number contained in the OUT(Tx) endpoint descriptor returned by the target device during enumeration.
- The TXMAXP register for the controller endpoint must be written with the maximum packet size (in bytes) for the transfer. This value should be the same as the wMaxPacketSize field of the Standard Endpoint Descriptor for the target endpoint.
- The HOST_TXINTERVAL register needs to be written with the required transaction interval (usually one transaction per frame/microframe).
- The relevant interrupt enable bit in the INTRTXE register should be set (if an interrupt is required for this endpoint).
- The following bits of HOST_TXCSR register should be set as shown below:
  - Set the MODE bit (bit 13) to 1 to ensure the FIFO is enabled (only necessary if the FIFO is shared with an Rx endpoint).

- Set the DMAEN bit (bit 12) to 1 if a DMA request is required for this endpoint.
- The FRCDATATOG bit (bit 12) is ignored for isochronous transactions.
- Set the DMAMODE bit (bit 10) to 1 when DMA is enabled and the endpoint interrupt is not needed for each packet transmission.

3.2.4.2.2 Operation

The operation starts when the software writes to the FIFO and sets TXPKTRDY bit of HOST_TXCSR (bit 0). This triggers the controller to send an OUT token followed by the first data packet from the FIFO.

An interrupt is generated whenever a packet is sent and the software may use this interrupt to load the next packet into the FIFO and set the TXPKTRDY bit in the HOST_TXCSR register (bit 0) in the same way as for a Bulk Tx endpoint. As the interrupt could occur almost any time within a frame, depending on when the host has scheduled the transaction, this may result in irregular timing of FIFO load requests. If the data source for the endpoint is coming from some external hardware, it may be more convenient to wait until the end of each frame before loading the FIFO as this will minimize the requirement for additional buffering. This can be done by using the SOF_PULSE signal from the controller to trigger the loading of the next data packet. The SOF_PULSE is generated once per frame(/microframe). The interrupts may still be used to set the TXPKTRDY bit in HOST_TXCSR.

3.3 DMA Operation

The DMA controller sub-module is a common 4 dual-channel DMA controller. It supports 4 TX and 4 RX channels, and each channel attaches to the associated endpoint in the controller. Channel 0 maps to Endpoint 1 up to Channel 3 mapping to Endpoint 4, while endpoint 0 cannot utilize the DMA. The DMA is programmed through parameters stored in the DMA registers, and implements DMAs through the buffer descriptor chains stored in main memory. The DMA utilizes incrementing addressing mode, and can burst up to 64 bytes. When a DMA is initiated, the DMA controller reads the current parameters for that channel from the DMA registers, reads the CPPI buffer descriptor pointed to by the parameters, and performs the DMA using the data buffer in the descriptor. The DMA controller updates and saves the DMA parameters when the data is completed. The software can access the current parameters at any time by accessing the DMA registers.

The DMA controller has a concept of DMA packets which is different from the USB packets. A DMA packet can be (but not necessarily) of the same size as USB packet. Each DMA packet can comprise of one or multiple data buffers. Each DMA channel can process one or multiple chains of these DMA packets.

The controller supports two modes of DMA: transparent mode and RNDIS mode. Transparent mode will interrupt the CPU for every USB data packet, while RNDIS mode can service multiple USB packets with only a single CPU interrupt.

3.3.1 DMA Transmit Operation

For transmit operation, the software has to program the DMA channel with a chain of transmit buffers.

3.3.1.1 Transmit Buffer

A Transmit buffer is a contiguous block of memory used to store data for transmission. Each Tx buffer has a corresponding Tx buffer descriptor. Each Tx buffer can be linked together with other Tx buffers to make a DMA packet or a queue of DMA packets.

Transmit buffers are byte aligned structures located in processor's main memory. Tx buffer size may vary from 1 to 65,535 bytes.

3.3.1.2 CPPI Transmit Buffer Descriptor

Tx buffer descriptors provide information about a single corresponding Tx data buffer. Every Tx buffer has a single Tx buffer descriptor that stores the following information:

- Pointer to the data buffer
- Pointer to the next buffer descriptor in the queue

- Buffer length and offset to the first valid byte of buffer data
- Start of DMA packet (SOP) indicator
- End of DMA packet (EOP) indicator
- Ownership (only valid with SOP)
- End of queue (EOQ) (only valid on EOP)
- Packet Length (only valid with SOP)

Transmit buffer descriptors contain 16 bytes (4 words) and must begin on 16-byte aligned addresses. Transmit buffer descriptors may be linked together to form packets. Buffer descriptor SOP and EOP bits are used to delimit packets. Packets in turn may be linked together to form transmit queue. Each queue consists of a chain of buffer descriptors linked together by Next Descriptor Pointers. The last buffer descriptor in a queue has a zero Next Descriptor Pointer. Each descriptor points to a data buffer yielding a queue of buffers.

Four Words of Transmit Buffer Descriptor are described below.

Table 6. Transmit Buffer Descriptor Word 0

Bits	Name	Description
31:0	Next Descriptor Pointer	The 32-bit word aligned memory address of the next buffer descriptor in the Tx queue. This is the mechanism used to reference the next buffer descriptor from the current buffer descriptor. If the value of this pointer is zero then the current buffer is the last buffer in the queue. The software sets the Next Descriptor Pointer.

Table 7. Transmit Buffer Descriptor Word 1

Bits	Name	Description
31:0	Buffer Pointer	The Buffer Pointer is the byte aligned memory address of the buffer associated with the buffer descriptor. The software sets the Buffer Pointer.

Table 8. Transmit Buffer Descriptor Word 2

Bits	Name	Description
31:16	Buffer Offset	The Buffer Offset indicates how many unused bytes are at the start of the buffer (SOP buffers only). A value of zero indicates that there are no unused bytes at the start of the buffer and that valid data begins on the first byte of the buffer. A value of 000Fh (decimal 15) indicates that the first 15 bytes of the buffer are to be ignored by the DMA controller while transmitting and that valid buffer data starts on byte 16 of the buffer. The Buffer Offset is valid only on Start Of Packet buffer descriptors and must be zero otherwise. The software sets the Buffer Offset. The Buffer Offset must be less than the Buffer Length.
15:0	Buffer Length	The Buffer Length field indicates how many valid data bytes are in the buffer. Unused or protocol specific bytes at the beginning of the buffer are not counted in the Buffer Length field. The software sets the Buffer Length.

Table 9. Transmit Buffer Descriptor Word 3

Bits	Name	Value	Description
31	SOP		Start of Packet: SOP Indicates that the descriptor buffer is the first buffer in the packet. The software sets the SOP bit.
		0	Not start of packet buffer
		1	Start of packet buffer
30	EOP		End of Packet: EOP Indicates that the descriptor buffer is the last buffer in the packet. The software sets the EOP bit. It is valid to set both SOP and EOP in the same descriptor.
		0	Not end of packet buffer
		1	End of packet buffer

Table 9. Transmit Buffer Descriptor Word 3 (continued)

Bits	Name	Value	Description
29	Ownership		The Ownership bit indicates ownership of the DMA packet and is valid only on SOP. This bit is set by the software and cleared by the DMA controller when the packet has been transmitted. The software can use this bit to reclaim buffers.
		0	The packet is owned by the host processor
		1	The packet is owned by the DMA controller
28	EOQ		End of Queue: The End of Queue bit is set by the DMA controller to indicate that all packets in the queue have been transmitted and the Tx queue is empty. This bit is valid only on when EOP is set.
		0	The Tx queue has more packets to transfer
		1	The DMA controller took this descriptor buffer as the last buffer descriptor in the last packet in the queue
27:24	Reserved		Reserved
23	Zero Byte		Zero Byte Packet Identifier. This bit is set by the software when a zero byte USB packet needs to be transmitted. This bit tells the DMA controller that no data transfer needs to be done for transmitting this zero byte data buffer. Set the Packet Length to 1 when setting this bit.
22:20	Reserved		Reserved
19	Rx Abort		Receive DMA Transfer abort indicator. This field is only valid on SOP Descriptor. DMA set this bit field to indicate the event that the Rx Packet being received is aborted due to the lack of sufficient buffers in the receive queue and the BUFCNT is not zero. DMA will force the BUFCNT to zero. Rx channel will start up again when the queue is replenished and the BUFCNT is incremented by software.
18-16	Reserved		Reserved
15:0	Packet Length		The length of the DMA packet in bytes. This field is valid only on SOP and is written by the software. If the Packet Length is less than the sum of the buffer lengths, then the packet data will be truncated. A Packet Length greater than the sum of the buffers is a software error. Packet Length must be nonzero. Set to one when setting the Zero Byte bit.

Four different cases are possible for the number of buffers in a DMA packet:

1. Buffer Descriptor contains Start of Packet field and End of Packet field (1 buffer in DMA packet).
2. Buffer Descriptor contains Start of Packet field only (2+ buffers in DMA packet).
3. Buffer Descriptor contains End of Packet field only (2+ buffers in DMA packet).
4. Buffer Descriptor does not contain either of Start of Packet field and End of Packet field (3+ buffers in packet).

3.3.1.3 Transmit DMA State

The DMA controller stores and maintains state information for each transmit channel. The state information is referred to as the Tx DMA State. The Tx DMA State is a combination of control fields and DMA controller specific scratchpad space used to manipulate data structures and transmit DMA packets. The Tx DMA State is stored in registers TCPPIDMASTATEW0, TCPPIDMASTATEW1, TCPPIDMASTATEW2, TCPPIDMASTATEW3, TCPPIDMASTATEW4, TCPPIDMASTATEW5 and TCPPICOMPPTR for each channel.

Each channel has one queue. The queue has one head descriptor pointer and one completion pointer.

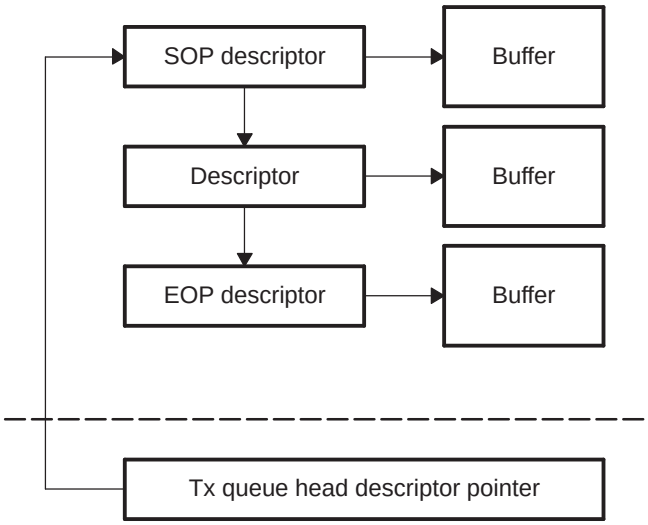
The following information is stored in the Tx DMA State:

- TCPPIDMASTATEW0: Tx Queue Head Descriptor Pointer(s)
- TCPPICOMPPTR: Tx Completion Pointer(s)
- TCPPIDMASTATEW1: Start of Packet Buffer Descriptor Pointer
- TCPPIDMASTATEW2: Current Buffer Descriptor Pointer
- TCPPIDMASTATEW3: Current Buffer Pointer
- TCPPIDMASTATEW5: Remaining DMA Packet Length and Actual DMA Packet Length

3.3.1.4 Transmit Queue

Figure 14 shows a Tx queue. Tx queue provide a logical queue of DMA packets for transmission through a channel. Each channel has one dedicated Tx queues. The queue has one associated Tx Queue Head Descriptor Pointer and one associated Tx Completion Pointer container in the channel Tx DMA state. The Tx queue is linked lists of Tx buffer descriptors that constitute one or more packets queued for transmission. Packets are added to the tail of the list by the software and packets are freed from the head of the list by the DMA controller as each packet transmission is completed.

Figure 14. Tx Queue Flow Chart



3.3.1.5 Operation

- After reset the software must write zeroes to all Tx DMA State registers (TCPPIDMASTATEW0, TCPPIDMASTATEW1, TCPPIDMASTATEW2, TCPPIDMASTATEW3, TCPPIDMASTATEW4, TCPPIDMASTATEW5).
- The software constructs transmit queues in memory (one or more DMA packets for transmission)
- Enable DMA for the endpoint in the PERI_TXCSR or HOST_TXCSR by setting the DMAEN bit.
- Enable the DMA ports by setting TCPPI_ENABLE bit of TCPPICR register.
- Write the head of the queue descriptor pointer to the TCPPIDMASTATEW0 register to start the DMA.
- The USB controller will start transmitting data. Interrupt associated with the DMA channel is asserted, after each DMA packet is transmitted.

For each buffer added to a transmit queue, the software must initialize the Tx buffer descriptor values as follows:

- Write the Next Descriptor Pointer with the 32-bit aligned address of the next descriptor in the queue (zero if last descriptor)
- Write the Buffer Pointer with the byte aligned address of the buffer data
- Write the Buffer Length with the number of bytes in the buffer
- Write the Buffer Offset with the number of bytes in the offset to the data (nonzero with SOP only)
- Set the SOP, EOP, and Ownership bits as appropriate
- Clear the End Of Queue bit

The DMA controller begins Tx DMA packet transmission on a given channel when the host writes the channel's Tx queue head descriptor pointer with the address of the first buffer descriptor in the queue (nonzero value). Each channel has one queue and a head descriptor pointer. The first buffer descriptor for each Tx DMA packet must have the Start of Packet (SOP) bit and the Ownership bit set to one by the software. The last buffer descriptor for each Tx DMA packet must have the End of Packet (EOP) bit set to one by the software. The DMA controller will transmit DMA packets until all queued packets have been transmitted and the queue is empty. When each packet transmission is complete, the DMA controller will

clear the Ownership bit in the DMA packet's SOP buffer descriptor and issue an interrupt to the processor by writing the DMA packet's last buffer descriptor address to the queue's Tx DMA State Completion Pointer (TCPPICOMPPTR register). When the last packet in a queue has been transmitted, the DMA controller sets the End Of Queue bit in the EOP buffer descriptor, clears the Ownership bit in the SOP Descriptor, zeroes the appropriate DMA state head descriptor pointer, and then issues a Tx interrupt to the host by writing address of the last buffer descriptor processed by the DMA controller to the queue's associated Tx completion pointer (TCPPICOMPPTR register).

On interrupt from the port, the software can process the buffer queue, detecting transmitted packets by the status of the Ownership bit in the SOP buffer descriptor. If the Ownership bit is cleared to zero, then the packet has been transmitted and the software may reclaim the buffers associated with the packet. The software continues queue processing until the end of the queue or until a SOP buffer descriptor is read that contains a set Ownership bit indicating that the packet transmission is not complete. The software determines that all packets in the queue have been transmitted when the last packet in the queue has a cleared Ownership bit in the SOP buffer descriptor, the End of Queue bit is set in the last packet EOP buffer descriptor, and the Next Descriptor Pointer of the last packet EOP buffer descriptor is zero.

The software acknowledges an interrupt by writing the address of the last buffer descriptor to the queue's associated Tx Completion Pointer (TCPPICOMPPTR register).

If the software written buffer address value in TCCPICOMPPTR register is different from the buffer address written by the DMA controller after Tx completion, then the interrupt for the Tx Channel remains asserted. If the software-written buffer address value matches with the buffer address written by the DMA controller, the Tx Channel interrupt gets deasserted.

A misqueued packet condition may occur when the software adds a packet to a queue for transmission as the DMA controller finishes transmitting the previous last packet in the queue. The misqueued packet is detected by the software when queue processing detects a cleared Ownership bit in the SOP buffer descriptor, a set End of Queue bit in the EOP buffer descriptor, and a nonzero Next Descriptor Pointer in the EOP buffer descriptor. A misqueued packet means that the DMA controller read the last EOP buffer descriptor before the host added the new last packet to the queue, so the DMA controller determined queue empty just before the last packet was added. The host software corrects the misqueued packet condition by initiating a new packet transfer for the misqueued packet by writing the misqueued packet's SOP buffer descriptor address to the head descriptor pointer in TCCPIDMASTATEW0 register.

3.3.1.6 Transparent Mode and RNDIS Mode Transmit DMA Operation

Transparent Mode DMA operation is the default DMA mode (as described in previous section) where an interrupt is generated whenever a DMA packet is transmitted. In the transparent mode, DMA packet size cannot be greater than USB MaxPktSize and FIFO size for the endpoint. This means, for transmitting say 'n' USB packets, the DMA controller should be programmed with a queue of 'n' DMA packets. Transparent mode must be used whenever USB MaxPktSize for the endpoint is not a multiple of 64 bytes.

RNDIS mode DMA is used to transmit DMA packets which are larger than USB MaxPktSize. This is accomplished by breaking the larger packet into smaller packets, not larger than USB MaxPktSize. This implies that the data to be transmitted will be sent over USB in multiple packets of MaxPktSize and the Tx DMA interrupt for the channel is generated after the transmission of complete DMA packet. This mode of DMA is used for RNDIS type transfers over USB. The protocol defines the end of the complete transfer by sending a short USB packet (smaller than USB MaxPktSize as mentioned in USB specification 2.0). If the DMA packet size is an exact multiple of USB MaxPktSize, the DMA controller sends a zero byte packet at the end of complete transfer to signify the completion of the transfer.

RNDIS Mode DMA is supported only when USB MaxPktSize and the associated FIFO size is an integral multiple of 64 bytes.

RNDIS Mode Setup

The setup of RNDIS mode DMA is similar to the default Transparent Mode as mentioned in the previous section. The following steps need to be taken for setting up RNDIS mode Tx DMA:

- After reset the software must write zeroes to all Tx DMA State registers (TCPPIDMASTATEW0, TCPPIDMASTATEW1, TCPPIDMASTATEW2, TCPPIDMASTATEW3, TCPPIDMASTATEW4, TCPPIDMASTATEW5).
- The software constructs transmit queue in memory (one or more DMA Packets in for transmission).
- Enable DMA for the endpoint in the PERI_TXCSR or HOST_TXCSR by setting the DMAEN bit.
- Enable the DMA ports by setting TCPPI_ENABLE bit of TCPPICR register.
- Set RNDIS bit of CTRLR register for enabling RNDIS mode for all channels or set TX*n*EN bit of RNDISR register for specific DMA channel *n*.
- Write the head of the queue descriptor pointer to the TCPPIDMASTATEW0 register to start the DMA.
- The USB controller will start transmitting data.
- If the DMA packet size is exact multiple of USB MaxPktSize, a zero byte packet is transmitted and interrupt associated with the DMA channel is asserted.
- If the DMA packet size is not exact multiple of USB MaxPktSize, the last USB packet transmitted is a short packet and interrupt associated with the DMA channel is asserted.

Transparent Mode Setup

Transparent DMA configuration is identical to RNDIS DMA configuration with the exception of the followings: Each packet is defined by a single buffer descriptor with SOP and EOP bit fields set. Packet size is not bounded to be a multiple of 64 byte but by max packet size. CTRLR.RNDIS bit field should be cleared to zero.

3.3.1.7 DMA Channel TearDown

The DMA also supports a teardown operation on TX channels. Teardown allows software to terminate the current Tx queue by notifying the DMA. The DMA will stop the current Tx DMA for that channel, and update the channel parameters to remove all Tx descriptors from that queue. Then the software is free to reclaim the buffers without worry of interrupting the DMA in progress.

Teardown of Tx channels is done using the TCPPITDR register. If the READY bit of TCPPITDR register is set, it signifies that the Tx channel can be torn down. The channel number should be written in the CHANNEL field of TCPPITDR register to teardown the channel.

Note that the software must also teardown the core after the DMA. Software should specify that the following steps are all completed before assuming teardown has completed successfully.

1. After tearing down the channel, DMA interrupt should be generated and the TCPPICOMPPTR register will be FFFF FFFCh. This indicates that the DMA has completed the teardown and all the data buffers can be reclaimed.
2. The software must set the FLUSHFIFO bit of the PERI_TXCSR or HOST_TXCSR register for the endpoint to be torn down.

After both the DMA and the core are torn down, the DMA channel and endpoint can be restarted cleanly. A failure to perform either of these steps could result in data loss or spurious data upon restart.

3.3.2 DMA Receive Operation

For receive operation, the software has to program the DMA channel with a chain of receive buffers.

3.3.2.1 Receive Buffer

A receive buffer is a contiguous block of memory used to store received data. Each Rx buffer has a corresponding Rx buffer descriptor. Each Rx buffer can be linked together with other Rx buffers to make a DMA packet or a queue of DMA packets.

Receive buffers are byte aligned structures located in processor's main memory. Rx buffer size may vary from 1 to 65,535 bytes.

3.3.2.2 CPPI Receive Buffer Descriptor

Rx buffer descriptors provide information about a single corresponding Rx data buffer. Every Rx buffer has a single Rx buffer descriptor that stores the following information:

- Pointer to the data buffer
- Pointer to the next buffer descriptor in the queue
- Buffer length and offset to the first valid byte of buffer data
- Start of DMA packet (SOP) indicator
- End of DMA packet (EOP) indicator
- Ownership (only valid with SOP)
- End of queue (EOQ) (only valid on EOP)
- Packet Length (only valid with SOP)

Receive buffer descriptors contain 16 bytes (4 words) and must begin on 16-byte aligned addresses. Receive buffer descriptors may be linked together to form packets. Buffer descriptor SOP and EOP bits are used to delimit packets. Packets in turn may be linked together to form receive queue. Each queue consists of a chain of buffer descriptors linked together by Next Descriptor Pointers. The last buffer descriptor in a queue has a zero Next Descriptor Pointer. Each descriptor points to a data buffer yielding a queue of buffers.

Four Words of Receive Buffer Descriptor are described in Table 10 through Table 13.

Table 10. Receive Buffer Descriptor Word 0

Bits	Name	Description
31:0	Next Descriptor Pointer	The 32-bit word aligned memory address of the next buffer descriptor in the Rx queue. This is the mechanism used to reference the next buffer descriptor from the current buffer descriptor. If the value of this pointer is zero then the current buffer is the last buffer in the queue. The software sets the Next Descriptor Pointer.

Table 11. Receive Buffer Descriptor Word 1

Bits	Name	Description
31:0	Buffer Pointer	The Buffer Pointer is the byte aligned memory address of the buffer associated with the buffer descriptor. The software sets the Buffer Pointer.

Table 12. Receive Buffer Descriptor Word 2

Bits	Name	Description
31:16	Buffer Offset	The Buffer Offset indicates how many unused bytes are at the start of the buffer (SOP buffers only). A value of zero indicates that there are no unused bytes at the start of the buffer and that valid data begins on the first byte of the buffer. A value of 000Fh (decimal 15) indicates that the first 15 bytes of the buffer are to be ignored by the DMA controller while transmitting and that valid buffer data starts on byte 16 of the buffer. The software sets the buffer offset to zero on buffer initialization and the DMA controller overwrites the zero value on SOP packets with the Rx DMA State buffer offset value.
15:0	Buffer Length	The Buffer Length field indicates how many valid data bytes are in the buffer. Unused or protocol specific bytes at the beginning of the buffer are not counted in the Buffer Length field. The software sets the buffer length on buffer initialization. The DMA controller will overwrite the software initialized value on an EOP buffer when the number of received data bytes is less than the host initiated value. The DMA controller will overwrite the host initialized value on SOP when the Buffer Offset is greater than zero, or the Packet Length is less than the buffer length.

Table 13. Receive Buffer Descriptor Word 3

Bit	Field	Value	Description
31	SOP		Start of Packet: SOP Indicates that the descriptor buffer is the first buffer in the packet. Software should clear the SOP bit when setting up the descriptor.
		0	Not start of packet buffer
		1	Start of packet buffer
30	EOP		End of Packet: EOP Indicates that the descriptor buffer is the last buffer in the packet. Software should clear the EOP bit when setting up the descriptor.
		0	Not end of packet buffer
		1	End of packet buffer
29	Ownership		The Ownership bit indicates ownership of the DMA packet and is valid only on SOP. This bit is set by the software and cleared by the DMA controller when the packet has been transmitted. The software can use this bit to reclaim buffers.
		0	The packet is owned by the host processor.
		1	The packet is owned by the DMA controller.
28	EOQ		End of Queue: The End of Queue bit is set by the DMA controller to indicate that all packets in the queue have been transmitted and the Tx queue is empty. This bit is valid only on when EOP is set. Software should clear this bit when setting up the descriptor.
		0	The Tx queue has more packets to transfer.
		1	The DMA controller took this descriptor buffer as the last buffer descriptor in the last packet in the queue.
27:24	Reserved		Reserved
23	Zero Byte		Zero Byte Packet Identifier. This bit is set by the DMA controller when a zero byte USB packet has been received. Software should ignore the Packet Length when this bit is set.
22:16	Reserved		Reserved
15:0	Packet Length		The length of the DMA packet in bytes. This field is valid only on SOP and is written by the DMA controller. If the Packet Length is less than the sum of the buffer lengths, then the packet data will be truncated. Software should ignore the Packet Length when Zero Byte is set.

Four different cases are possible for the number of buffers in a DMA packet:

1. Buffer Descriptor contains Start of Packet field and End of Packet field (1 buffer in DMA packet)
2. Buffer Descriptor contains Start of Packet field only (2+ buffers in DMA packet)
3. Buffer Descriptor contains End of Packet field only (2+ buffers in DMA packet)
4. Buffer Descriptor does not contain either of Start of Packet field and End of Packet field (3+ buffers in packet)

3.3.2.3 Receive DMA State

The DMA controller stores and maintains state information for each receive channel. The state information is referred to as the Rx DMA State. The Rx DMA State is a combination of control fields and DMA controller specific scratchpad space used to manipulate data structures and receive DMA packets. The Rx DMA State is stored in registers RCPPIDMASTATEW0, RCPPIDMASTATEW1, RCPPIDMASTATEW2, RCPPIDMASTATEW3, RCPPIDMASTATEW4, RCPPIDMASTATEW5, RCPPIDMASTATEW6 and RCPPICOMPPTR for each channel.

Each channel has one receive queue that has a head descriptor pointer and one completion pointer.

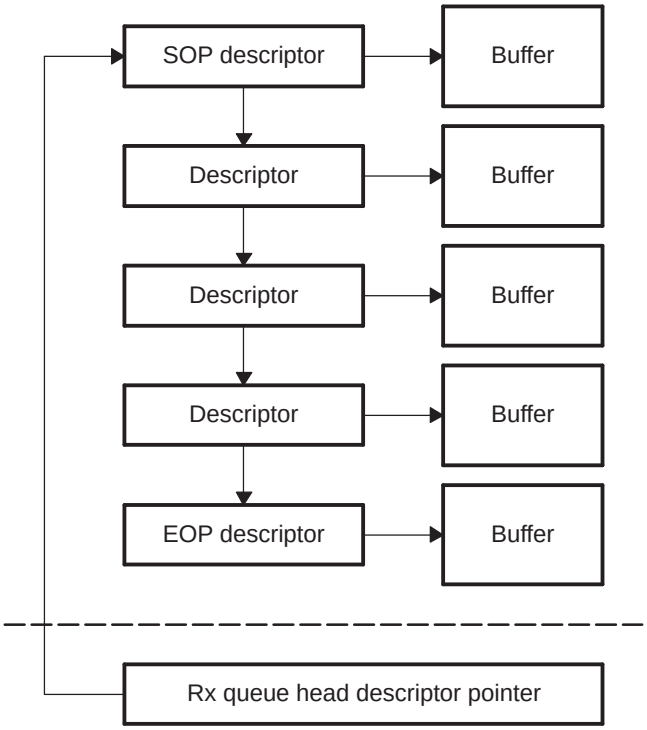
The following information is stored in the Tx DMA State:

- RCPPIDMASTATEW0: Rx Queue Head Descriptor Pointer
- RCPPICOMPPTR: Rx Completion Pointer
- RCPPIDMASTATEW2: Start of Packet Buffer Descriptor Pointer
- RCPPIDMASTATEW3: Current Buffer Descriptor Pointer
- RCPPIDMASTATEW4: Current Buffer Pointer
- RCPPIDMASTATEW5: Remaining DMA Packet Length and Actual DMA Packet Length

3.3.2.4 Receive Queue

Figure 15 shows an Rx Queue. Rx queue provide a logical queue of processor memory space for DMA packets to be received from DMA controller channel. Each channel has single Rx queue. There are no multiple queue as in transmit channels. Each queue has one associated Rx Queue Head Descriptor Pointer and one associated Rx Completion Pointer contained in the channel Rx DMA State. The Rx queue are linked lists of Rx buffer descriptors that constitute processor memory space for one or more packets to be received. Packet space is added to the tail of the list by the software and received packets are freed from the list by the DMA controller as each packet is received.

Figure 15. Rx Queue Flow Chart



3.3.2.5 Operation

- After reset the software must write zeroes to all Rx DMA State registers (RCPPIDMASTATEW0, RCPPIDMASTATEW1, RCPPIDMASTATEW2, RCPPIDMASTATEW3, RCPPIDMASTATEW4, RCPPIDMASTATEW5 and RCPPIDMASTATEW6).
- The software constructs receive queue in memory.
- Enable DMA for the endpoint in the PERI_RXCSR or HOST_RXCSR by setting the DMAEN bit.
- Enable the DMA ports by setting RCPPI_ENABLE bit of RCPPICR register.
- Set the value in RBUFCNT*n* register (where n is the channel number) for the number of buffers available in the Rx queue. The hardware requires at least 3 available buffers to start the DMA. A new transfer will not be started if the buffer count is below 3. The value in RBUFCNT*n* decrements as DMA controller consumes the buffers for reception.
- Write the head of the queue descriptor pointer to the RCPPIDMASTATEW1 register to start the DMA.
- The USB controller will send IN token and wait for the data on the bus. Once data is received, DMA controller will transfer the data in the Rx queue from the endpoint FIFO. Once a complete DMA packet is received, interrupt associated with the DMA channel is asserted.

The software enables packet reception on a given channel by writing the address of the first buffer descriptor in the queue (nonzero value) to the channel's head descriptor pointer (RCCPIDMASTATEW1) in the channel's Rx DMA state. When packet reception begins on a given channel, the DMA controller fills each Rx buffer with data in order starting with the first buffer and proceeding through the Rx queue. If the Buffer Offset in the Rx DMA State is nonzero, then the controller will begin writing data after the offset number of bytes in the SOP buffer. The DMA controller performs the following operations at the end of each packet reception:

- Overwrite the buffer length in the packet's EOP buffer descriptor with the number of bytes actually received in the packet's last buffer. The software initialized value is the buffer size. The overwritten value will be less than or equal to the software initialized value.
- Set the EOP bit in the packet's EOP buffer descriptor.
- Set the EOQ bit in the packet's EOP buffer descriptor if the current packet is the last packet in the queue.
- Overwrite the packet's SOP buffer descriptor Buffer Offset with the Rx DMA state value (the software initialized the buffer descriptor Buffer Offset value to zero). All non SOP buffer descriptors must have a zero Buffer Offset initialized by the host.
- Overwrite the packet's SOP buffer descriptor buffer length with the number of valid data bytes in the buffer. If the buffer is filled up, the buffer length will be the buffer size minus buffer offset.
- Set the SOP bit in the packet's SOP buffer descriptor.
- Write the SOP buffer descriptor Packet Length field.
- Clear the Ownership bit in the packet's SOP buffer descriptor.
- Issue an Rx DMA interrupt to the host processor by writing the address of the packet's last buffer descriptor to the queue's Rx DMA State Completion Pointer (RCPPICOMPPTR register).

  On interrupt the software processes the Rx buffer queue detecting received packets by the status of the Ownership bit in each packet's SOP buffer descriptor. If the Ownership bit is cleared then the packet has been completely received and is available to be processed by the software. The software may continue Rx queue processing until the end of the queue or until a buffer descriptor is read that contains a set Ownership bit indicating that the next packet's reception is not complete. The software determines that the Rx queue is empty when the last packet in the queue has a cleared Ownership bit in the SOP buffer descriptor, a set End of Queue bit in the EOP buffer descriptor, and the Next Descriptor Pointer in the EOP buffer descriptor is zero.

  The software acknowledges an interrupt by writing the address of the last buffer descriptor to the queue's associated Rx Completion Pointer (RCPPICOMPPTR register).

  If the software written buffer address value in RCCPICOMPPTR register is different from the buffer address written by the DMA controller after Rx completion, then the interrupt for the Rx Channel remains asserted. If the software written buffer address value matches with the buffer address written by the DMA controller, the Rx Channel interrupt gets deasserted.

  A misqueued buffer may occur when the software adds buffers to a queue as the DMA controller finishes the reception of the previous last packet in the queue. The misqueued buffer is detected by the software when queue processing detects a cleared Ownership bit in the SOP buffer descriptor, a set End of Queue bit in the EOP buffer descriptor, and a nonzero Next Descriptor Pointer in the EOP buffer descriptor. A misqueued buffer means that the DMA controller read the last EOP buffer descriptor before the software added buffer descriptor(s) to the queue, so the DMA controller determined queue empty just before the software added more buffer descriptor(s). Receive overrun condition may occur in the misqueued buffer case. If a new packet reception is begun during the time that the DMA controller has determined the end of queue condition, then the received packet will overrun (start of packet overrun). If the misqueued buffer occurs during the middle of a packet reception then middle of packet overrun may occur. If the misqueued buffer occurs after the last packet has completed, and is corrected before the next packet reception begins, then overrun will not occur. The software acts on the misqueued buffer condition by writing the added buffer descriptor address to the appropriate Rx DMA State Head Descriptor Pointer in RCPPIDMASTATEW1 register.

3.3.2.6 Receive Abort Handling

The DMA controller sets 'Rx Abort' bit used to identify Rx packets which were aborted due to lack of buffers. Software must take care to inspect any Rx SOP packet for this bit and ignore all the buffers in that packet as the packet is incomplete. Also, for aborted packets, the packet length may not match the data size in the buffers.

3.3.2.7 RNDIS Mode and Transparent Mode Receive DMA Operation

Transparent Mode DMA operation is the default DMA mode (as described in previous section) where an interrupt is generated whenever a DMA packet is received. In the transparent mode, DMA packet size cannot be greater than USB MaxPktSize for the endpoint. This means, for receiving say *n* USB packets, the DMA controller should be programmed with a queue of minimum *n* DMA packets. RBUFCNT*n* register also needs to be written with value *n*. The number *n* should be greater than 3 as the RBUFCNT*n* register should have at least 3 packets for the Rx DMA to receive packets from the endpoint FIFO.

RNDIS mode DMA is used to receive DMA packets which are larger than USB MaxPktSize. This is accomplished by breaking the larger packet into smaller packets, not larger than USB MaxPktSize. This implies that multiple USB packets of MaxPktSize will be received and transferred together as a single large DMA packet and the DMA interrupt is generated only at the end of the complete reception of DMA packet. This mode of DMA is used for RNDIS type transfers over USB. The protocol defines the end of the complete transfer by receiving a short USB packet (smaller than USB MaxPktSize as mentioned in USB specification 2.0). If the DMA packet size is an exact multiple of USB MaxPktSize, the DMA controller waits for a zero byte packet at the end of complete transfer to signify the completion of the transfer.

RNDIS Mode DMA is supported only when USB MaxPktSize is an integral multiple of 64 bytes.

RNDIS Mode Setup

The setup of RNDIS mode DMA is similar to the default Transparent Mode as mentioned in the previous section. The following steps need to be taken for setting up RNDIS mode Rx DMA:

- RNDIS mode requires that the associated MaxPkt Size and FIFO size must be integral multiples of 64 bytes.
- After reset the software must write zeroes to all Rx DMA State registers (RCPPIDMASTATEW0, RCPPIDMASTATEW1, RCPPIDMASTATEW2, RCPPIDMASTATEW3, RCPPIDMASTATEW4, RCPPIDMASTATEW5 and RCPPIDMASTATEW6).
- The software constructs receive queue in memory.
- Enable DMA for the endpoint in the PERI_RXCSR or HOST_RXCSR by setting the DMAEN bit.
- Enable the DMA ports by setting RCPPI_ENABLE bit of RCPPICR register.
- Set RNDIS bit of CTRLR register for enabling RNDIS mode for all channels or set TX*n*EN bit of RNDISR register for specific DMA channel *n*.
- Set the value in RBUFCNT*n* register (where n is the channel number) for the number of buffers available in the Rx queue. The minimum value should be 3 for the Rx DMA to start operation. The value in RBUFCNT*n* decrements as DMA controller consumes the buffers for reception.
- Write the head of the queue descriptor pointer to the RCPPIDMASTATEW1 register to start the DMA.
- The USB controller will send IN token and wait for the data on the bus. Once data is received, DMA controller will transfer the data in the Rx queue from the endpoint FIFO. Once a complete DMA packet is received, the interrupt associated with the DMA channel is asserted.
- If the DMA packet size is exact multiple of USB MaxPktSize, a zero byte packet is expected and interrupt associated with the DMA channel is asserted when that zero byte packet is received.
- If the DMA packet size is not exact multiple of USB MaxPktSize, the last USB packet received is a short packet and interrupt associated with the DMA channel is asserted.

An additional feature of automatically generating IN tokens is available. This feature is used in USB Host mode operation of the USB controller. This feature is functional in RNDIS mode DMA only. To automatically generate the IN tokens while receiving data, set the field RXn_AUTOREQ (where n is the channel number) of AUTOREQ register with binary 01. In this case, IN tokens will be generated and sent to the target USB peripheral device after every successfully received packet. No IN token will be generated after the End Of DMA Packet is reached. Rx DMA interrupt is generated after the complete reception.

If RXn_AUTOREQ (where n is the channel number) of AUTOREQ register is set with binary 11, IN tokens will be generated and sent to the target USB peripheral device even after the End Of DMA Packet is reached. This feature is useful to keep data reception operational across multiple DMA packets in a Rx queue. The host processor does not have to restart sending IN tokens for every DMA packet in the Rx queue.

Transparent Mode Setup

Transparent receive DMA configuration is identical to RNDIS DMA configuration with the exception of the following:

- Each packet is defined by a single buffer descriptor with SOP and EOP bit fields set.
- Packet size is not bounded to be a multiple of 64 byte but by max packet size.
- CTRLR.RNDIS bit field should be cleared to zero.
- RXn:AUTOREQ register field is programmed with the value of 0, which is programmed for no Auto Request.

3.3.2.8 DMA Teardown Procedure

In order to Teardown a TX channel, the endpoint FIFO must be flushed after the DMA Teardown completes. Teardown is not complete until the following steps are successfully completed.

- Write the TX Teardown register in the CPPI DMA with the channel to teardown. The DMA will interrupt after the teardown is complete and the TX Completion Pointer will be FFFF FFFCh. This indicates that the DMA has completed the teardown and all of the associated CPPI buffers can be reclaimed.
- Call the flush_tx_fifo routine (code provided in Example 4) once if the FIFO is set up for single-buffer or twice for double-buffer.

3.4 Interrupt Handling

Table 14 lists the interrupts generated by the USB controller.

Table 14. Interrupts Generated by the USB Controller

Interrupt	Description
Tx Endpoint [4:0]	Tx endpoint ready or error condition. For endpoints 4 to 0. (Rx and Tx for endpoint 0)
Rx Endpoint [4:1]	Rx endpoint ready or error condition. For endpoints 4 to 1. (Endpoint 0 has interrupt status in Tx interrupt)
USB Core[8:0]	Interrupts for 9 USB conditions
DMA Tx Completion [3:0]	Tx DMA completion interrupt for channel 3 to 0
DMA Rx Completion [3:0]	Rx DMA completion interrupt for channel 3 to 0

Whenever any of these interrupt conditions are generated, the host processor is interrupted. The software needs to read the different interrupt status registers (discussed in later section) to determine the source of the interrupt.

The nine USB interrupt conditions are listed in Table 15.

Table 15. USB Interrupt Conditions

Interrupt	Description
USB[8]	DRVVBUS level change
USB[7]	VBus voltage < VBus Valid Threshold (VBus error)
USB[6]	SRP detected
USB[5]	Device Disconnected (Valid in Host Mode)
USB[4]	Device Connected (Valid in Host Mode)
USB[3]	SOF started
USB[2]	Reset Signaling detected (In Peripheral Mode) Babble detected (In Host Mode)
USB[1]	Resume signaling detected
USB[0]	Suspend Signaling detected

3.4.1 USB Core Interrupts

There are two methods available for software to access USB core interrupts, selectable by the UINT bit of CTRLR. The UINT bit cleared to 0 selects the PDR 2.0 compliant register set (INTSRCR, INTSETR, INTCLRR, INTMSKR, INTMSKSETR, INTMSKCLRR, INTMASKEDR). This is the default, and should be used for most systems. The DRVVBUS level change interrupt is only available in the PDR compliant register. UINT set to one selects direct access to the USB core interrupt registers (INTRUSB, INTRUSBE, INTRTX, INTRRX). Software should select a single method for interrupts and use its corresponding registers exclusively.

Interrupt status can be determined using the INTSRCR (interrupt source) register. This register is non-masked. To clear the interrupt source, set the corresponding interrupt bit in INTCLRR register. For debugging purposes, interrupt can be set manually through INTSETR register.

The interrupt controller provides the option of masking the interrupts. A mask can be set using INTMSKSETR register and can be cleared by setting the corresponding bit in the INTMSKCLRR register. The mask can be read from INTMSKR register. The masked interrupt status is determined using the INTMASKEDR register.

Software should write all zeros to the End Of Interrupt Register (EOIR) to acknowledge the completion of the USB core interrupt.

NOTE: If the EOIR is not written, the interrupt output to the CPU will not be pulsed again for the next interrupt.

3.4.2 DMA Interrupts

Interrupt status for the DMA interrupts is determined by TCCPIRAWSR and RCPPIRAWSR registers. These are the raw interrupt status registers for DMA interrupts.

Tx DMA interrupts mask is set using TCPPIENSETR register and cleared using TCPPIIENCLRR register. The masked status is read using TCPPIMSKSR register.

Rx DMA interrupts mask is set using RCPPIENSETR register and cleared using RCPPIIENCLRR register. The masked status is read using RCPPIMSKSR register.

Like USB core interrupts, the CPPIEOIR register needs to be written by the host processor software to acknowledge the completion of the interrupt.

Upon receipt of a DMA interrupt, software should check TCPPIRAWSR/RCPPIRAWSR to determine which DMA channel(s) to service. Check the CPPI buffer descriptor ownership field for the completed channel for error conditions and add or update buffer descriptors as needed. Write the RCPPICOMPPTR or TCPPICOMPPTR completion pointer with the address of the buffer descriptor serviced in order to clear the interrupt. Finally, write the DMA End Of Interrupt register CPPIEOIR with all zeros to enable future (or current unserviced) interrupts to pulse the interrupt output to the CPU.

When using DMA with a TX endpoint, set the TXCSR register DMAMODE bit to one in order to receive only error (not packet completion) interrupts. For DMA with an RX endpoint, the DMAMODE bit in RXCSR should be cleared to 0.

3.5 Test Modes

The controller supports the four USB 2.0 test modes defined for High-speed functions. It also supports an additional “FIFO access” test mode that can be used to test the operation of the CPU interface, the DMA controller (if configured) and the RAM block.

The test modes are entered by writing to the TestMode register (offset address 0x40F). At test mode is usually requested by the host sending a SET_FEATURE request to Endpoint 0. When the software receives the request, it should wait until the Endpoint 0 transfer has completed (when it receives the Endpoint 0 interrupt indicating the status phase has completed) then write to the TestMode register.

NOTE: These test modes have no purpose in normal operation.

3.5.1 TEST_SE0_NAK

To enter the Test_SE0_NAK test mode, the software should set the Test_SE0-NAK bit by writing 0x01 to the TestMode register. The controller will then go into a mode in which it responds to any valid IN token with a NAK.

3.5.2 TEST_J

To enter the Test_J test mode, the software should set the Test_J bit by writing 0x02 to the TestMode register. The controller will then go into a mode in which it transmits a continuous J on the bus.

3.5.3 TEST_K

To enter the Test_K test mode, the software should set the Test_K bit by writing 0x04 to the TestMode register. The controller will go into a mode in which it transmits a continuous K on the bus.

3.5.4 TEST_PACKET

To execute the Test_Packet, the software should:

1. Start a session (if the core is being used in Host mode).
2. Write the standard test packet (shown below) to the Endpoint 0 FIFO.
3. Write 0x8 to the TestMode register to enter Test_Packet test mode.
4. Set the TxPktRdy bit in the CSR0 register (D1).

The 53 by test packet to load is as follows (all bytes in hex). The test packet only has to be loaded once; the controller will keep re-sending the test packet without any further intervention from the software.

00	00	00	00	00	00	00	00
00	AA	AA	AA	AA	AA	AA	AA
AA	EE	EE	EE	EE	EE	EE	EE
EE	FE	FF	FF	FF	FF	FF	FF
FF	FF	FF	FF	FF	7F	BF	DF
EF	F7	FB	FD	FC	7E	BF	DF
EF	F7	FB	FD	7E			

This data sequence is defined in Universal Serial Bus Specification Revision 2.0, Section 7.1.20. The controller will add the DATAA0 PID to the head of the data sequence and the CRC to the end.

3.5.5 FIFO_ACCESS

The FIFO Access test mode allows the user to test the operation of CPU Interface, the DMA controller (if configured) and the RAM block by loading a packet of up to 64 bytes into the Endpoint 0 FIFO and then reading it back out again. Endpoint 0 is used because it is a bi-directional endpoint that uses the same area of RAM for its Tx and Rx FIFOs.

NOTE: The core does not need to be connected to the USB bus to run this test. If it is connected, then no session should be in progress when the test is run.

The test procedure is as follows:

1. Load a packet of up to 64 bytes into the Endpoint 0 Tx FIFO.
2. Set CSR0.TxPktRdy.
3. Write 0x40 to the Testmode register.
4. Unload the packet from the Endpoint Rx FIFO, again.
5. Set CSR0.ServicedRxPktRdy

Writing 0x40 to the Testmode register causes the following sequence of events:

1. The Endpoint 0 CPU pointer (which records the number of bytes to be transmitted) is copied to the Endpoint 0 USB pointer (which records the number of bytes received).
2. The Endpoint 0 CPU pointer is reset.
3. CSR0.TxPktRdy is cleared.
4. CSR0.RxPktRdy is set.
5. An Endpoint 0 interrupt is generated (if enabled).,

The effect of these steps is to make the Endpoint 0 controller act as if the packet loaded into the Tx FIFO has flushed and the same packet received over the USB. The data that was loaded in the Tx FIFO can now be read out of the Rx FIFO.

3.5.6 FORCE_HOST

The Force Host test mode enables the user to instruct the core to operate in Host mode, regardless of whether it is actually connected to any peripheral, i.e., the state of the CID input and the LINESTATE and HOSTDISCON signals are ignored. (While in this mode, the state of the HOSTDISCON signal can be read from bit 7 of the DevCtl register.)

This mode, which is selected by setting bit 7 within the Testmode register, allows implementation of the USB Test_Force_Enable (7.1.20). It can also be used for debugging PHY problems in hardware.

While the Force_Host bit remains set, the core will enter Host mode when the Session bit is set and remain in Host mode until the Session bit is cleared even if a connected device is disconnected during the session. The operating speed while in this mode is determined for the sitting of the Force_HS and Force_FS bits of the Testmode register.

3.6 Reset Considerations

The USB controller has two reset sources: hardware reset and the soft reset (RESET bit in CTRLR register).

3.6.1 Software Reset Considerations

When the RESET bit in CTRLR is set, all the USB controller registers and DMA operations are reset. The bit is cleared automatically.

A software reset on the ARM or DSP CPUs does not affect the register values and operation of the USB controller.

3.6.2 Hardware Reset Considerations

When a hardware reset is asserted, all the registers are set to their default values.

3.6.3 USB Reset Considerations

When operating in peripheral mode, a USB reset received from the host causes some internal registers to be reset. The USB controller setup operations (for example FIFO sizing) in peripheral mode should be performed after receiving the USB reset, and again on each subsequent USB reset. There are some conditions where multiple USB reset interrupts may be received in rapid succession. Good interrupt handling practices must be observed to assure that the setup is performed (again) after the final USB reset interrupt.

Software must teardown any TX DMA channel use upon receipt of a USB reset. See Section 3.3.2.8.

3.7 Interrupt Support

The USB peripheral presents a single interrupt to the ARM interrupt controller (AINTC). For information on the mapping of interrupts, refer to the device-specific data manual.

3.8 EDMA Event Support

The USB is an internal bus master peripheral and therefore does not utilize any EDMA events. The registers support only individual access. Bursting data to or from the USB register space through EDMA is not supported.

3.9 Power Management

The USB controller can be placed in reduced power modes to conserve power during periods of low activity. The power management of the peripheral is controlled by the processor Power and Sleep Controller (PSC). The PSC acts as a master controller for power management for all of the peripherals on the device. For detailed information on power management procedures using the PSC, see the *TMS320DM644x DMSoC ARM Subsystem Reference Guide* (SPRUE14) .

4 Registers

Table 16 lists the memory-mapped registers for the universal serial bus (USB). See the device-specific data manual for the memory address of these registers. The base address is 01C6 4000h.

NOTE: In some cases, a single register address can have different names or meanings depending on the mode (host/peripheral) or the setting of the index register. The meaning of some bit fields varies with the mode.

Table 16. Universal Serial Bus (USB) Registers

Offset	Acronym	Register Description	Section
4h	CTRLR	Control Register	Section 4.1
8h	STATR	Status Register	Section 4.2
10h	RNDISR	RNDIS Register	Section 4.3
14h	AUTOREQ	Autorequest Register	Section 4.4
20h	INTSRCR	USB Interrupt Source Register	Section 4.5
24h	INTSETR	USB Interrupt Source Set Register	Section 4.6
28h	INTCLRR	USB Interrupt Source Clear Register	Section 4.7
2Ch	INTMSKR	USB Interrupt Mask Register	Section 4.8
30h	INTMSKSETR	USB Interrupt Mask Set Register	Section 4.9
34h	INTMSKCLRR	USB Interrupt Mask Clear Register	Section 4.10
38h	INTMASKEDR	USB Interrupt Source Masked Register	Section 4.11
3Ch	EOIR	USB End of Interrupt Register	Section 4.12
80h	TCPPICR	Transmit CPPI Control Register	Section 4.13
84h	TCPPITDR	Transmit CPPI Teardown Register	Section 4.14
88h	CPPIEOIR	CPPI DMA End of Interrupt Register	Section 4.15
90h	TCPPIMSKSR	Transmit CPPI Masked Status Register	Section 4.16
94h	TCPPIRAWSR	Transmit CPPI Raw Status Register	Section 4.17
98h	TCPPIIENSETR	Transmit CPPI Interrupt Enable Set Register	Section 4.18
9Ch	TCPPIIENCLRR	Transmit CPPI Interrupt Enable Clear Register	Section 4.19
C0h	RCPPICR	Receive CPPI Control Register	Section 4.20
D0h	RCPPIMSKSR	Receive CPPI Masked Status Register	Section 4.21
D4h	RCPPIRAWSR	Receive CPPI Raw Status Register	Section 4.22
D8h	RCPPIENSETR	Receive CPPI Interrupt Enable Set Register	Section 4.23
DCh	RCPPIIENCLRR	Receive CPPI Interrupt Enable Clear Register	Section 4.24
E0h	RBUFCNT0	Receive Buffer Count 0 Register	Section 4.25
E4h	RBUFCNT1	Receive Buffer Count 1 Register	Section 4.26
E8h	RBUFCNT2	Receive Buffer Count 2 Register	Section 4.27
ECh	RBUFCNT3	Receive Buffer Count 3 Register	Section 4.28
		Transmit/Receive CPPI Channel 0 State Block	
100h	TCPPIDMASTATEW0	Transmit CPPI DMA State Word 0	Section 4.29
104h	TCPPIDMASTATEW1	Transmit CPPI DMA State Word 1	Section 4.30
108h	TCPPIDMASTATEW2	Transmit CPPI DMA State Word 2	Section 4.31
10Ch	TCPPIDMASTATEW3	Transmit CPPI DMA State Word 3	Section 4.32
110h	TCPPIDMASTATEW4	Transmit CPPI DMA State Word 4	Section 4.33
114h	TCPPIDMASTATEW5	Transmit CPPI DMA State Word 5	Section 4.34
11Ch	TCPPICOMPPTR	Transmit CPPI Completion Pointer	Section 4.35
120h	RCPPIDMASTATEW0	Receive CPPI DMA State Word 0	Section 4.36
124h	RCPPIDMASTATEW1	Receive CPPI DMA State Word 1	Section 4.37
128h	RCPPIDMASTATEW2	Receive CPPI DMA State Word 2	Section 4.38

Table 16. Universal Serial Bus (USB) Registers (continued)

Offset	Acronym	Register Description	Section
12Ch	RCPPIDMASTATEW3	Receive CPPI DMA State Word 3	Section 4.39
130h	RCPPIDMASTATEW4	Receive CPPI DMA State Word 4	Section 4.40
134h	RCPPIDMASTATEW5	Receive CPPI DMA State Word 5	Section 4.41
138h	RCPPIDMASTATEW6	Receive CPPI DMA State Word 6	Section 4.42
13Ch	RCPPICOMPPTR	Receive CPPI Completion Pointer	Section 4.43
		Transmit/Receive CPPI Channel 1 State Block	
140h	TCPPIDMASTATEW0	Transmit CPPI DMA State Word 0	Section 4.29
144h	TCPPIDMASTATEW1	Transmit CPPI DMA State Word 1	Section 4.30
148h	TCPPIDMASTATEW2	Transmit CPPI DMA State Word 2	Section 4.31
14Ch	TCPPIDMASTATEW3	Transmit CPPI DMA State Word 3	Section 4.32
150h	TCPPIDMASTATEW4	Transmit CPPI DMA State Word 4	Section 4.33
154h	TCPPIDMASTATEW5	Transmit CPPI DMA State Word 5	Section 4.34
15Ch	TCPPICOMPPTR	Transmit CPPI Completion Pointer	Section 4.35
160h	RCPPIDMASTATEW0	Receive CPPI DMA State Word 0	Section 4.36
164h	RCPPIDMASTATEW1	Receive CPPI DMA State Word 1	Section 4.37
168h	RCPPIDMASTATEW2	Receive CPPI DMA State Word 2	Section 4.38
16Ch	RCPPIDMASTATEW3	Receive CPPI DMA State Word 3	Section 4.39
170h	RCPPIDMASTATEW4	Receive CPPI DMA State Word 4	Section 4.40
174h	RCPPIDMASTATEW5	Receive CPPI DMA State Word 5	Section 4.41
178h	RCPPIDMASTATEW6	Receive CPPI DMA State Word 6	Section 4.42
17Ch	RCPPICOMPPTR	Receive CPPI Completion Pointer	Section 4.43
		Transmit/Receive CPPI Channel 2 State Block	
180h	TCPPIDMASTATEW0	Transmit CPPI DMA State Word 0	Section 4.29
184h	TCPPIDMASTATEW1	Transmit CPPI DMA State Word 1	Section 4.30
188h	TCPPIDMASTATEW2	Transmit CPPI DMA State Word 2	Section 4.31
18Ch	TCPPIDMASTATEW3	Transmit CPPI DMA State Word 3	Section 4.32
190h	TCPPIDMASTATEW4	Transmit CPPI DMA State Word 4	Section 4.33
194h	TCPPIDMASTATEW5	Transmit CPPI DMA State Word 5	Section 4.34
19Ch	TCPPICOMPPTR	Transmit CPPI Completion Pointer	Section 4.35
1A0h	RCPPIDMASTATEW0	Receive CPPI DMA State Word 0	Section 4.36
1A4h	RCPPIDMASTATEW1	Receive CPPI DMA State Word 1	Section 4.37
1A8h	RCPPIDMASTATEW2	Receive CPPI DMA State Word 2	Section 4.38
1ACh	RCPPIDMASTATEW3	Receive CPPI DMA State Word 3	Section 4.39
1B0h	RCPPIDMASTATEW4	Receive CPPI DMA State Word 4	Section 4.40
1B4h	RCPPIDMASTATEW5	Receive CPPI DMA State Word 5	Section 4.41
1B8h	RCPPIDMASTATEW6	Receive CPPI DMA State Word 6	Section 4.42
1BCh	RCPPICOMPPTR	Receive CPPI Completion Pointer	Section 4.43
		Transmit/Receive CPPI Channel 3 State Block	
1C0h	TCPPIDMASTATEW0	Transmit CPPI DMA State Word 0	Section 4.29
1C4h	TCPPIDMASTATEW1	Transmit CPPI DMA State Word 1	Section 4.30
1C8h	TCPPIDMASTATEW2	Transmit CPPI DMA State Word 2	Section 4.31
1CCh	TCPPIDMASTATEW3	Transmit CPPI DMA State Word 3	Section 4.32
1D0h	TCPPIDMASTATEW4	Transmit CPPI DMA State Word 4	Section 4.33
1D4h	TCPPIDMASTATEW5	Transmit CPPI DMA State Word 5	Section 4.34
1DCh	TCPPICOMPPTR	Transmit CPPI Completion Pointer	Section 4.35
1E0h	RCPPIDMASTATEW0	Receive CPPI DMA State Word 0	Section 4.36
1E4h	RCPPIDMASTATEW1	Receive CPPI DMA State Word 1	Section 4.37

Table 16. Universal Serial Bus (USB) Registers (continued)

Offset	Acronym	Register Description	Section
1E8h	RCPPIDMASTATEW2	Receive CPPI DMA State Word 2	Section 4.38
1ECh	RCPPIDMASTATEW3	Receive CPPI DMA State Word 3	Section 4.39
1F0h	RCPPIDMASTATEW4	Receive CPPI DMA State Word 4	Section 4.40
1F4h	RCPPIDMASTATEW5	Receive CPPI DMA State Word 5	Section 4.41
1F8h	RCPPIDMASTATEW6	Receive CPPI DMA State Word 6	Section 4.42
1FCh	RCPPICOMPPTR	Receive CPPI Completion Pointer	Section 4.43
		Common USB Registers	
400h	FADDR	Function Address Register	Section 4.44
401h	POWER	Power Management Register	Section 4.45
402h	INTRTX	Interrupt Register for Endpoint 0 and for Transmit Endpoints 1 to 4	Section 4.46
404h	INTRRX	Interrupt Register for Receive Endpoints 1 to 4	Section 4.47
406h	INTRTXE	Interrupt enable register for INTRTX	Section 4.48
408h	INTRRXE	Interrupt Enable Register for INTRRX	Section 4.49
40Ah	INTRUSB	Interrupt Register for Common USB Interrupts	Section 4.50
40Bh	INTRUSBE	Interrupt Enable Register for INTRUSB	Section 4.51
40Ch	FRAME	Frame Number Register	Section 4.52
40Eh	INDEX	Index Register for Selecting the Endpoint Status and Control Registers	Section 4.53
40Fh	TESTMODE	Register to Enable the USB 2.0 Test Modes	Section 4.54
		Indexed Registers (These registers operate on the endpoint selected by the INDEX register)	
410h	TXMAXP	Maximum Packet Size for Peripheral/Host Transmit Endpoint (Index register set to select Endpoints 1-4)	Section 4.55
412h	PERI_CSR0	Control Status Register for Endpoint 0 in Peripheral Mode. (Index register set to select Endpoint 0)	Section 4.56
	HOST_CSR0	Control Status Register for Endpoint 0 in Host Mode (Index register set to select Endpoint 0)	Section 4.57
	PERI_TXCSR	Control Status Register for Peripheral Transmit Endpoint. (Index register set to select Endpoints 1-4)	Section 4.58
	HOST_TXCSR	Control Status Register for Host Transmit Endpoint (Index register set to select Endpoints 1-4)	Section 4.59
414h	RXMAXP	Maximum Packet Size for Peripheral/Host Receive Endpoint (Index register set to select Endpoints 1-4)	Section 4.60
416h	PERI_RXCSR	Control Status Register for Peripheral Receive Endpoint. (Index register set to select Endpoints 1-4)	Section 4.61
	HOST_RXCSR	Control Status Register for Host Receive Endpoint (Index register set to select Endpoints 1-4)	Section 4.62
418h	COUNT0	Number of Received Bytes in Endpoint 0 FIFO (Index register set to select Endpoint 0)	Section 4.63
	RXCOUNT	Number of Bytes in Host Receive Endpoint FIFO (Index register set to select Endpoints 1- 4)	Section 4.64
41Ah	HOST_TYPE0	Defines the speed of Endpoint 0	Section 4.65
	HOST_TXTYPE	Sets the operating speed, transaction protocol and peripheral endpoint number for the host Transmit endpoint. (Index register set to select Endpoints 1-4)	Section 4.66
41Bh	HOST_NAKLIMIT0	Sets the NAK response timeout on Endpoint 0 (Index register set to select Endpoint 0)	Section 4.67
	HOST_TXINTERVAL	Sets the polling interval for Interrupt/ISOC transactions or the NAK response timeout on Bulk transactions for host Transmit endpoint. (Index register set to select Endpoints 1-4)	Section 4.68
41Ch	HOST_RXTYPE	Sets the operating speed, transaction protocol and peripheral endpoint number for the host Receive endpoint. (Index register set to select Endpoints 1-4)	Section 4.69

Table 16. Universal Serial Bus (USB) Registers (continued)

Offset	Acronym	Register Description	Section
41Dh	HOST_RXINTERVAL	Sets the polling interval for Interrupt/ISOC transactions or the NAK response timeout on Bulk transactions for host Receive endpoint. (Index register set to select Endpoints 1-4)	Section 4.70
41Fh	CONFIGDATA	Returns details of core configuration. (Index register set to select Endpoint 0)	Section 4.71
		FIFOn	
420h	FIFO0	Transmit and Receive FIFO Register for Endpoint 0	Section 4.72
424h	FIFO1	Transmit and Receive FIFO Register for Endpoint 1	Section 4.73
428h	FIFO2	Transmit and Receive FIFO Register for Endpoint 2	Section 4.74
42Ch	FIFO3	Transmit and Receive FIFO Register for Endpoint 3	Section 4.75
430h	FIFO4	Transmit and Receive FIFO Register for Endpoint 4	Section 4.76
		OTG Device Control	
460h	DEVCTL	OTG Device Control Register	Section 4.77
		Dynamic FIFO Control	
462h	TXFIFOSZ	Transmit Endpoint FIFO Size (Index register set to select Endpoints 1-4)	Section 4.78
463h	RXFIFOSZ	Receive Endpoint FIFO Size (Index register set to select Endpoints 1-4)	Section 4.79
464h	TXFIFOADDR	Transmit Endpoint FIFO Address (Index register set to select Endpoints 1-4)	Section 4.80
466h	RXFIFOADDR	Receive Endpoint FIFO Address (Index register set to select Endpoints 1-4)	Section 4.81
		Target Endpoint 0 Control Registers, Valid Only in Host Mode	
480h	TXFUNCADDR	Address of the target function that has to be accessed through the associated Transmit Endpoint.	Section 4.82
482h	TXHUBADDR	Address of the hub that has to be accessed through the associated Transmit Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.83
483h	TXHUBPORT	Port of the hub that has to be accessed through the associated Transmit Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.84
484h	RXFUNCADDR	Address of the target function that has to be accessed through the associated Receive Endpoint.	Section 4.85
486h	RXHUBADDR	Address of the hub that has to be accessed through the associated Receive Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.86
487h	RXHUBPORT	Port of the hub that has to be accessed through the associated Receive Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.87
		Target Endpoint 1 Control Registers, Valid Only in Host Mode	
488h	TXFUNCADDR	Address of the target function that has to be accessed through the associated Transmit Endpoint.	Section 4.82
48Ah	TXHUBADDR	Address of the hub that has to be accessed through the associated Transmit Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.83
48Bh	TXHUBPORT	Port of the hub that has to be accessed through the associated Transmit Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.84
48Ch	RXFUNCADDR	Address of the target function that has to be accessed through the associated Receive Endpoint.	Section 4.85

Table 16. Universal Serial Bus (USB) Registers (continued)

Offset	Acronym	Register Description	Section
48Eh	RXHUBADDR	Address of the hub that has to be accessed through the associated Receive Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.86
48Fh	RXHUBPORT	Port of the hub that has to be accessed through the associated Receive Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.87
		Target Endpoint 2 Control Registers, Valid Only in Host Mode	
490h	TXFUNCADDR	Address of the target function that has to be accessed through the associated Transmit Endpoint.	Section 4.82
492h	TXHUBADDR	Address of the hub that has to be accessed through the associated Transmit Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.83
493h	TXHUBPORT	Port of the hub that has to be accessed through the associated Transmit Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.84
494h	RXFUNCADDR	Address of the target function that has to be accessed through the associated Receive Endpoint.	Section 4.85
496h	RXHUBADDR	Address of the hub that has to be accessed through the associated Receive Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.86
497h	RXHUBPORT	Port of the hub that has to be accessed through the associated Receive Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.87
		Target Endpoint 3 Control Registers, Valid Only in Host Mode	
498h	TXFUNCADDR	Address of the target function that has to be accessed through the associated Transmit Endpoint.	Section 4.82
49Ah	TXHUBADDR	Address of the hub that has to be accessed through the associated Transmit Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.83
49Bh	TXHUBPORT	Port of the hub that has to be accessed through the associated Transmit Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.84
49Ch	RXFUNCADDR	Address of the target function that has to be accessed through the associated Receive Endpoint.	Section 4.85
49Eh	RXHUBADDR	Address of the hub that has to be accessed through the associated Receive Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.86
49Fh	RXHUBPORT	Port of the hub that has to be accessed through the associated Receive Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.87
		Target Endpoint 4 Control Registers, Valid Only in Host Mode	
4A0h	TXFUNCADDR	Address of the target function that has to be accessed through the associated Transmit Endpoint.	Section 4.82
4A2h	TXHUBADDR	Address of the hub that has to be accessed through the associated Transmit Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.83

Table 16. Universal Serial Bus (USB) Registers (continued)

Offset	Acronym	Register Description	Section
4A3h	TXHUBPORT	Port of the hub that has to be accessed through the associated Transmit Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.84
4A4h	RXFUNCADDR	Address of the target function that has to be accessed through the associated Receive Endpoint.	Section 4.85
4A6h	RXHUBADDR	Address of the hub that has to be accessed through the associated Receive Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.86
4A7h	RXHUBPORT	Port of the hub that has to be accessed through the associated Receive Endpoint. This is used only when full speed or low speed device is connected via a USB2.0 high-speed hub.	Section 4.87
		Control and Status Register for Endpoint 0	
502h	PERI_CSR0	Control Status Register for Endpoint 0 in Peripheral Mode	Section 4.56
	HOST_CSR0	Control Status Register for Endpoint 0 in Host Mode	Section 4.57
508h	COUNT0	Number of Received Bytes in Endpoint 0 FIFO	Section 4.63
50Ah	HOST_TYPE0	Defines the Speed of Endpoint 0	Section 4.65
50Bh	HOST_NAKLIMIT0	Sets the NAK Response Timeout on Endpoint 0	Section 4.67
50Fh	CONFIGDATA	Returns details of core configuration.	Section 4.71
		Control and Status Register for Endpoint 1	
510h	TXMAXP	Maximum Packet Size for Peripheral/Host Transmit Endpoint	Section 4.55
512h	PERI_TXCSR	Control Status Register for Peripheral Transmit Endpoint (peripheral mode)	Section 4.58
	HOST_TXCSR	Control Status Register for Host Transmit Endpoint (host mode)	Section 4.59
514h	RXMAXP	Maximum Packet Size for Peripheral/Host Receive Endpoint	Section 4.60
516h	PERI_RXCSR	Control Status Register for Peripheral Receive Endpoint (peripheral mode)	Section 4.61
	HOST_RXCSR	Control Status Register for Host Receive Endpoint (host mode)	Section 4.62
518h	RXCOUNT	Number of Bytes in Host Receive endpoint FIFO	Section 4.64
51Ah	HOST_TXTYPE	Sets the operating speed, transaction protocol and peripheral endpoint number for the host Transmit endpoint.	Section 4.66
51Bh	HOST_TXINTERVAL	Sets the polling interval for Interrupt/ISOC transactions or the NAK response timeout on Bulk transactions for host Transmit endpoint.	Section 4.68
51Ch	HOST_RXTYPE	Sets the operating speed, transaction protocol and peripheral endpoint number for the host Receive endpoint.	Section 4.69
51Dh	HOST_RXINTERVAL	Sets the polling interval for Interrupt/ISOC transactions or the NAK response timeout on Bulk transactions for host Receive endpoint.	Section 4.70
		Control and Status Register for Endpoint 2	
520h	TXMAXP	Maximum Packet Size for Peripheral/Host Transmit Endpoint	Section 4.55
522h	PERI_TXCSR	Control Status Register for Peripheral Transmit Endpoint (peripheral mode)	Section 4.58
	HOST_TXCSR	Control Status Register for Host Transmit Endpoint (host mode)	Section 4.59
524h	RXMAXP	Maximum Packet Size for Peripheral/Host Receive Endpoint	Section 4.60
526h	PERI_RXCSR	Control Status Register for Peripheral Receive Endpoint (peripheral mode)	Section 4.61
	HOST_RXCSR	Control Status Register for Host Receive Endpoint (host mode)	Section 4.62
528h	RXCOUNT	Number of Bytes in Host Receive endpoint FIFO	Section 4.64

Table 16. Universal Serial Bus (USB) Registers (continued)

Offset	Acronym	Register Description	Section
52Ah	HOST_TXTYPE	Sets the operating speed, transaction protocol and peripheral endpoint number for the host Transmit endpoint.	Section 4.66
52Bh	HOST_TXINTERVAL	Sets the polling interval for Interrupt/ISOC transactions or the NAK response timeout on Bulk transactions for host Transmit endpoint.	Section 4.68
52Ch	HOST_RXTYPE	Sets the operating speed, transaction protocol and peripheral endpoint number for the host Receive endpoint.	Section 4.69
52Dh	HOST_RXINTERVAL	Sets the polling interval for Interrupt/ISOC transactions or the NAK response timeout on Bulk transactions for host Receive endpoint.	Section 4.70
		Control and Status Register for Endpoint 3	
530h	TXMAXP	Maximum Packet Size for Peripheral/Host Transmit Endpoint	Section 4.55
532h	PERI_TXCSR	Control Status Register for Peripheral Transmit Endpoint (peripheral mode)	Section 4.58
	HOST_TXCSR	Control Status Register for Host Transmit Endpoint (host mode)	Section 4.59
534h	RXMAXP	Maximum Packet Size for Peripheral/Host Receive Endpoint	Section 4.60
536h	PERI_RXCSR	Control Status Register for Peripheral Receive Endpoint (peripheral mode)	Section 4.61
	HOST_RXCSR	Control Status Register for Host Receive Endpoint (host mode)	Section 4.62
538h	RXCOUNT	Number of Bytes in Host Receive endpoint FIFO	Section 4.64
53Ah	HOST_TXTYPE	Sets the operating speed, transaction protocol and peripheral endpoint number for the host Transmit endpoint.	Section 4.66
53Bh	HOST_TXINTERVAL	Sets the polling interval for Interrupt/ISOC transactions or the NAK response timeout on Bulk transactions for host Transmit endpoint.	Section 4.68
53Ch	HOST_RXTYPE	Sets the operating speed, transaction protocol and peripheral endpoint number for the host Receive endpoint.	Section 4.69
53Dh	HOST_RXINTERVAL	Sets the polling interval for Interrupt/ISOC transactions or the NAK response timeout on Bulk transactions for host Receive endpoint.	Section 4.70
		Control and Status Register for Endpoint 4	
540h	TXMAXP	Maximum Packet Size for Peripheral/Host Transmit Endpoint	Section 4.55
542h	PERI_TXCSR	Control Status Register for Peripheral Transmit Endpoint (peripheral mode)	Section 4.58
	HOST_TXCSR	Control Status Register for Host Transmit Endpoint (host mode)	Section 4.59
544h	RXMAXP	Maximum Packet Size for Peripheral/Host Receive Endpoint	Section 4.60
546h	PERI_RXCSR	Control Status Register for Peripheral Receive Endpoint (peripheral mode)	Section 4.61
	HOST_RXCSR	Control Status Register for Host Receive Endpoint (host mode)	Section 4.62
548h	RXCOUNT	Number of Bytes in Host Receive endpoint FIFO	Section 4.64
54Ah	HOST_TXTYPE	Sets the operating speed, transaction protocol and peripheral endpoint number for the host Transmit endpoint.	Section 4.66
54Bh	HOST_TXINTERVAL	Sets the polling interval for Interrupt/ISOC transactions or the NAK response timeout on Bulk transactions for host Transmit endpoint.	Section 4.68
54Ch	HOST_RXTYPE	Sets the operating speed, transaction protocol and peripheral endpoint number for the host Receive endpoint.	Section 4.69
54Dh	HOST_RXINTERVAL	Sets the polling interval for Interrupt/ISOC transactions or the NAK response timeout on Bulk transactions for host Receive endpoint.	Section 4.70

4.1 Control Register (CTRLR)

The Control Register (CTRLR) is shown in Figure 16 and described in Table 17.

Figure 16. Control Register (CTRLR)

31	16
Reserved	
R-0	

15	5	4	3	2	1	0
Reserved		RNDIS	UINT	Reserved	CLKFACK	RESET
R-0		R/W-0	R/W-0	R-0	R/W-0	W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 17. Control Register (CTRLR) Field Descriptions

Bit	Field	Value	Description
31-5	Reserved	0	Reserved
4	RNDIS		Global RNDIS mode enable for all endpoints.
		0	Global RNDIS mode is disabled.
		1	Global RNDIS mode is enabled.
3	UINT		USB non-PDR interrupt handler enable.
		0	PDR interrupt handler is enabled.
		1	PDR interrupt handler is disabled.
2	Reserved	0	Reserved
1	CLKFACK		Clock stop fast ACK enable.
		0	Clock stop fast ACK is disabled.
		1	Clock stop fast ACK is enabled.
0	RESET		Soft reset.
		0	No effect.
		1	Writing a 1 starts a module reset. The USB controller will clear this bit when it completes reset.

4.2 Status Register (STATR)

The Status Register (STATR) is shown in Figure 17 and described in Table 18.

Figure 17. Status Register (STATR)

31	16
Reserved	
R-0	

15	1	0
Reserved		DRVVBUS
R-0		R-0

LEGEND: R = Read only; -*n* = value after reset

Table 18. Status Register (STATR) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	DRVVBUS		Current DRVVBUS value.
		0	DRVVBUS value is logic 0
		1	DRVVBUS value is logic 1

4.3 RNDIS Register (RNDISR)

The RNDIS Register (RNDISR) is shown in Figure 18 and described in Table 19.

Figure 18. RNDIS Register (RNDISR)

31	20	19	18	17	16
Reserved		RX4EN	RX3EN	RX2EN	RX1EN
R-0		R/W-0	R/W-0	R/W-0	R/W-0

15	4	3	2	1	0
Reserved		TX4EN	TX3EN	TX2EN	TX1EN
R-0		R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 19. RNDIS Register (RNDISR) Field Descriptions

Bit	Field	Value	Description
31-20	Reserved	0	Reserved
19	RX4EN	0-1	Receive Endpoint 4 RNDIS mode enable.
18	RX3EN	0-1	Receive Endpoint 3 RNDIS mode enable.
17	RX2EN	0-1	Receive Endpoint 2 RNDIS mode enable.
16	RX1EN	0-1	Receive Endpoint 1 RNDIS mode enable.
15-4	Reserved	0	Reserved
3	TX4EN	0-1	Transmit Endpoint 4 RNDIS mode enable.
2	TX3EN	0-1	Transmit Endpoint 3 RNDIS mode enable.
1	TX2EN	0-1	Transmit Endpoint 2 RNDIS mode enable.
0	TX1EN	0-1	Transmit Endpoint 1 RNDIS mode enable.

4.4 Auto Request Register (AUTOREQ)

The Auto Request Register (AUTOREQ) is shown in Figure 19 and described in Table 20.

Figure 19. Auto Request Register (AUTOREQ)

31	16
Reserved	
R-0	

15	8	7	6	5	4	3	2	1	0
Reserved		Rx4		Rx3		Rx2		Rx1	
R-0		R/W-0		R/W-0		R/W-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; *-n* = value after reset

Table 20. Auto Request Register (AUTOREQ) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7-6	Rx4	0-3h	RX endpoint 4 Auto Req enable.
		0	No auto req
		1h	Auto req on all but EOP
		2h	Reserved
		3h	Auto req always
5-4	Rx3	0-3h	RX endpoint 3 Auto Req enable.
		0	No auto req
		1h	Auto req on all but EOP
		2h	Reserved
		3h	Auto req always
3-2	Rx2	0-3h	RX endpoint 2 Auto Req enable.
		0	No auto req
		1h	Auto req on all but EOP
		2h	Reserved
		3h	Auto req always
1-0	Rx1	0-3h	RX endpoint 1 Auto Req enable.
		0	No auto req
		1h	Auto req on all but EOP
		2h	Reserved
		3h	Auto req always

4.5 USB Interrupt Source Register (INTSRCR)

The USB Interrupt Source Register (INTSRCR) is shown in Figure 20 and described in Table 21.

Figure 20. USB Interrupt Source Register (INTSRCR)

31	25	24	16
Reserved		USB	
R-0		R-0	

15	13	12	9	8	5	4	0
Reserved		RX		Reserved		TX	
R-0		R-0		R-0		R-0	

LEGEND: R = Read only; -*n* = value after reset

Table 21. USB Interrupt Source Register (INTSRCR) Field Descriptions

Bit	Field	Value	Description
31-25	Reserved	0	Reserved
24-16	USB	0-1FFh	USB interrupt sources Generated by the USB core (not the DMA)
15-13	Reserved	0	Reserved
12-9	RX	0-Fh	Receive endpoint interrupt sources Generated by the USB core (not the DMA)
8-5	Reserved	0	Reserved
4-0	TX	0-1Fh	Transmit endpoint interrupt sources Generated by the USB core (not the DMA)

4.6 USB Interrupt Source Set Register (INTSETR)

The USB Interrupt Source Set Register (INTSETR) is shown in Figure 21 and described in Table 22.

Figure 21. USB Interrupt Source Set Register (INTSETR)

31	25	24	16
Reserved		USB	
R-0		W-0	

15	13	12	9	8	5	4	0
Reserved		RX		Reserved		TX	
R-0		W-0		R-0		W-0	

LEGEND: R = Read only; W = Write only; -*n* = value after reset

Table 22. USB Interrupt Source Set Register (INTSETR) Field Descriptions

Bit	Field	Value	Description
31-25	Reserved	0	Reserved
24-16	USB	0-1FFh	Write a 1 to set equivalent USB interrupt source Allows the USB interrupt sources to be manually triggered
15-13	Reserved	0	Reserved
12-9	RX	0-Fh	Write a 1 to set equivalent Receive endpoint interrupt source Allows the USB interrupt sources to be manually triggered
8-5	Reserved	0	Reserved
4-0	TX	0-1Fh	Write a 1 to set equivalent Transmit endpoint interrupt source Allows the USB interrupt sources to be manually triggered

4.7 USB Interrupt Source Clear Register (INTCLRR)

The USB Interrupt Source Clear Register (INTCLRR) is shown in Figure 22 and described in Table 23.

Figure 22. USB Interrupt Source Clear Register (INTCLRR)

31	25	24	16
Reserved		USB	
R-0		W-0	

15	13	12	9	8	5	4	0
Reserved		RX		Reserved		TX	
R-0		W-0		R-0		W-0	

LEGEND: R = Read only; W = Write only; -*n* = value after reset

Table 23. USB Interrupt Source Clear Register (INTCLRR) Field Descriptions

Bit	Field	Value	Description
31-25	Reserved	0	Reserved
24-16	USB	0-1FFh	Write a 1 to clear equivalent USB interrupt source Allows the CPU to acknowledge an interrupt source and turn it off
15-13	Reserved	0	Reserved
12-9	RX	0-Fh	Write a 1 to clear equivalent Receive endpoint interrupt source Allows the CPU to acknowledge an interrupt source and turn it off
8-5	Reserved	0	Reserved
4-0	TX	0-1Fh	Write a 1 to clear equivalent Transmit endpoint interrupt source Allows the CPU to acknowledge an interrupt source and turn it off

4.8 USB Interrupt Mask Register (INTMSKR)

The USB Interrupt Mask Register (INTMSKR) is shown in Figure 23 and described in Table 24.

Figure 23. USB Interrupt Mask Register (INTMSKR)

31	25	24	16
Reserved		USB	
R-0		R-0	

15	13	12	9	8	5	4	0
Reserved		RX		Reserved		TX	
R-0		R-0		R-0		R-0	

LEGEND: R = Read only; -*n* = value after reset

Table 24. USB Interrupt Mask Register (INTMSKR) Field Descriptions

Bit	Field	Value	Description
31-25	Reserved	0	Reserved
24-16	USB	0-1FFh	USB interrupt source masks Contains the masks of the interrupt sources generated by the USB core (not the DMA). These masks are used to enable or disable interrupt sources appearing as masked interrupts
15-13	Reserved	0	Reserved
12-9	RX	Fh	Receive endpoint interrupt source masks Contains the masks of the interrupt sources generated by the USB core (not the DMA). These masks are used to enable or disable interrupt sources appearing as masked interrupts.
8-5	Reserved	0	Reserved
4-0	TX	0-1Fh	Transmit endpoint interrupt source masks Contains the masks of the interrupt sources generated by the USB core (not the DMA). These masks are used to enable or disable interrupt sources appearing as masked interrupts.

4.9 USB Interrupt Mask Set Register (INTMSKSETR)

The USB Interrupt Mask Set Register (INTMSKSETR) is shown in Figure 24 and described in Table 25.

Figure 24. USB Interrupt Mask Set Register (INTMSKSETR)

31	25	24	16
Reserved		USB	
R-0		W-0	

15	13	12	9	8	5	4	0
Reserved		RX		Reserved		TX	
R-0		W-0		R-0		W-0	

LEGEND: R = Read only; W = Write only; -*n* = value after reset

Table 25. USB Interrupt Mask Set Register (INTMSKSETR) Field Descriptions

Bit	Field	Value	Description
31-25	Reserved	0	Reserved
24-16	USB	0-1FFh	Write a 1 to set equivalent USB interrupt mask Allows the USB interrupt masks to be individually enabled
15-13	Reserved	0	Reserved
12-9	RX	0-Fh	Write a 1 to set equivalent Receive endpoint interrupt mask Allows the USB interrupt masks to be individually enabled
8-5	Reserved	0	Reserved
4-0	TX	0-1Fh	Write a 1 to set equivalent Transmit endpoint interrupt mask Allows the USB interrupt masks to be individually enabled

4.10 USB Interrupt Mask Clear Register (INTMSKCLRR)

The USB Interrupt Mask Clear Register (INTMSKCLRR) is shown in Figure 25 and described in Table 26.

Figure 25. USB Interrupt Mask Clear Register (INTMSKCLRR)

31	25	24	16
Reserved		USB	
R-0		W-0	

15	13	12	9	8	5	4	0
Reserved		RX		Reserved		TX	
R-0		W-0		R-0		W-0	

LEGEND: R = Read only; W = Write only; -*n* = value after reset

Table 26. USB Interrupt Mask Clear Register (INTMSKCLRR) Field Descriptions

Bit	Field	Value	Description
31-25	Reserved	0	Reserved
24-16	USB	0-1FFh	Write a 1 to clear equivalent USB interrupt mask Allows the USB interrupt masks to be individually disabled
15-13	Reserved	0	Reserved
12-9	RX	0-Fh	Write a 1 to clear equivalent Receive endpoint interrupt mask Allows the USB interrupt masks to be individually disabled
8-5	Reserved	0	Reserved
4-0	TX	0-1Fh	Write a 1 to clear equivalent Transmit endpoint interrupt mask Allows the USB interrupt masks to be individually disabled

4.11 USB Interrupt Source Masked Register (INTMASKEDR)

The USB Interrupt Source Masked Register (INTMASKEDR) is shown in Figure 26 and described in Table 27.

Figure 26. USB Interrupt Source Masked Register (INTMASKEDR)

31	25	24	16
Reserved		USB	
R-0		R-0	

15	13	12	9	8	5	4	0
Reserved		RX		Reserved		TX	
R-0		R-0		R-0		R-0	

LEGEND: R = Read only; -*n* = value after reset

Table 27. USB Interrupt Source Masked Register (INTMASKEDR) Field Descriptions

Bit	Field	Value	Description
31-25	Reserved	0	Reserved
24-16	USB	0-1FFh	USB interrupt sources masked Contains the status of the interrupt sources generated by the USB core masked by the USB Interrupt Mask Register values
15-13	Reserved	0	Reserved
12-9	RX	0-Fh	Receive endpoint interrupt sources masked Contains the status of the interrupt sources generated by the USB core masked by the USB Interrupt Mask Register values.
8-5	Reserved	0	Reserved
4-0	TX	0-1Fh	Transmit endpoint interrupt sources masked Contains the status of the interrupt sources generated by the USB core masked by the USB Interrupt Mask Register values.

4.12 USB End of Interrupt Register (EOIR)

The USB End of Interrupt Register (EOIR) is shown in Figure 27 and described in Table 28.

Figure 27. USB End of Interrupt Register (EOIR)

31	16
Reserved	
R-0	

15	8	7	0
Reserved		VECTOR	
R-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; *-n* = value after reset

Table 28. USB End of Interrupt Register (EOIR) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7-0	VECTOR	0-FFh	USB End of Interrupt Vector. Allows the CPU to acknowledge completion of non-DMA interrupt by writing ZERO to EOI Vector field.

4.13 Transmit CPPI Control Register (TCPPICR)

The Transmit CPPI Control Register (TCPPICR) is shown in Figure 28 and described in Table 29.

Figure 28. Transmit CPPI Control Register (TCPPICR)

31	16
Reserved	
R-0	

15	1	0
Reserved		TCPPI_ENABLE
R-0		R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 29. Transmit CPPI Control Register (TCPPICR) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	TCPPI_ENABLE		Transmit CPPI Enable Controls if the Transmit CPPI DMA controller is enabled. Be sure to program the CPPI chain and set the DMA state words before enabling the DMA. Failure to initialize the DMA before enabling could result in spurious transfers and memory corruption.
		0	Transmit CPPI DMA is disabled
		1	Transmit CPPI DMA is enabled

4.14 Transmit CPPI Teardown Register (TCPPITDR)

The Transmit CPPI Teardown Register (TCPPITDR) is shown in Figure 29 and described in Table 30.

Figure 29. Transmit CPPI Teardown Register (TCPPITDR)

31	30	16
READY	Reserved	
R-1	R-0	

15	2	1	0
Reserved		CHANNEL	
R-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 30. Transmit CPPI Teardown Register (TCPPITDR) Field Descriptions

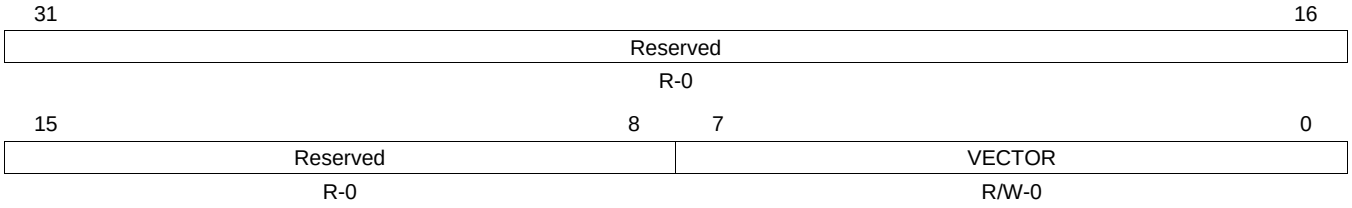
Bit	Field	Value	Description
31	READY	0-1	Indicates if the Teardown register can be written
30-2	Reserved	0	Reserved
1-0	CHANNEL	0-3h	Teardown Channel Indicates the channel that is to be torn down

4.15 CPPI DMA End of Interrupt Register (CPPIEOIR)

NOTE: This register was previously named TCPPIEOIR, and that name will continue to exist in the CSL for backward compatibility.

The CPPI DMA End of Interrupt Register (CPPIEOIR) is shown in Figure 30 and described in Table 31.

Figure 30. CPPI DMA End of Interrupt Register (CPPIEOIR)



LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 31. CPPI DMA End of Interrupt Register (CPPIEOIR) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7-0	VECTOR	0-FFh	DMA End of Interrupt Vector Allows the CPU to acknowledge completion of DMA interrupt by writing ZERO to the VECTOR field.

4.16 Transmit CPPI Masked Status Register (TCPPIMSKSR)

The Transmit CPPI Masked Status Register (TCPPIMSKSR) is shown in Figure 31 and described in Table 32.

Figure 31. Transmit CPPI Masked Status Register (TCPPIMSKSR)

31	16
Reserved	
R-0	

15	4	3	0
Reserved		MASKED COMP_PENDING	
R-0		R-0	

LEGEND: R = Read only; -*n* = value after reset

Table 32. Transmit CPPI Masked Status Register (TCPPIMSKSR) Field Descriptions

Bit	Field	Value	Description
31-4	Reserved	0	Reserved
3-0	MASKED COMP_PENDING	0-Fh	Masked High Priority Transmit Completion Pending Indicators for channels 3 to 0 Raw Transmit high priority completion indicators bitwise anded with Transmit high priority completion mask bits

4.17 Transmit CPPI Raw Status Register (TCPPIRAWSR)

The Transmit CPPI Raw Status Register (TCPPIRAWSR) is shown in Figure 32 and described in Table 33.

Figure 32. Transmit CPPI Raw Status Register (TCPPIRAWSR)

31	16
Reserved	
R-0	

15	4	3	0
Reserved		COMP_PENDING	
R-0		R-0	

LEGEND: R = Read only; -*n* = value after reset

Table 33. Transmit CPPI Raw Status Register (TCPPIRAWSR) Field Descriptions

Bit	Field	Value	Description
31-4	Reserved	0	Reserved
3-0	COMP_PENDING	0-Fh	Raw High Priority Transmit Completion Pending Indicators for channels 3 to 0 Active high flags which indicate that a packet has completed transmission on the high priority queue

4.18 Transmit CPPI Interrupt Enable Set Register (TCPPIIENSETR)

The Transmit CPPI Interrupt Enable Set Register (TCPPIIENSETR) is shown in Figure 33 and described in Table 34.

Figure 33. Transmit CPPI Interrupt Enable Set Register (TCPPIIENSETR)

31	16
Reserved	
R-0	

15	4	3	0
Reserved		COMP_PENDING_INTR_EN	
R-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 34. Transmit CPPI Interrupt Enable Set Register (TCPPIIENSETR) Field Descriptions

Bit	Field	Value	Description
31-4	Reserved	0	Reserved
3-0	COMP_PENDING_INTR_EN	0-Fh	Transmit CPPI High Priority Interrupt Enables These are active high interrupt enables corresponding to the Transmit CPPI High Priority Completion Pending status bits.

4.19 Transmit CPPI Interrupt Enable Clear Register (TCPPIIENCLRR)

The Transmit CPPI Interrupt Enable Clear Register (TCPPIIENCLRR) is shown in Figure 34 and described in Table 35.

Figure 34. Transmit CPPI Interrupt Enable Clear Register (TCPPIIENCLRR)

31	16
Reserved	
R-0	

15	4	3	0
Reserved		COMP_PENDING_INTR_EN	
R-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 35. Transmit CPPI Interrupt Enable Clear Register (TCPPIIENCLRR) Field Descriptions

Bit	Field	Value	Description
31-4	Reserved	0	Reserved
3-0	COMP_PENDING_INTR_EN	0-Fh	Writing a 1 to any of the bits in the Transmit CPPI Interrupt Enable Clear Register will result in clearing of the corresponding bit in the Transmit CPPI High Priority Interrupt Enable Register.

4.20 Receive CPPI Control Register (RCPPICR)

The Receive CPPI Control Register (RCPPICR) is shown in Figure 35 and described in Table 36.

Figure 35. Receive CPPI Control Register (RCPPICR)

31	16
Reserved	
R-0	

15	1	0
Reserved		RCPPI_ENABLE
R-0		R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 36. Receive CPPI Control Register (RCPPICR) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	RCPPI_ENABLE		Receive CPPI Enable Controls if the Receive CPPI DMA controller is enabled. Be sure to program the CPPI chain and set the DMA state words before enabling the DMA. Failure to initialize the DMA before enabling could result in spurious transfers and memory corruption.
		0	Receive CPPI DMA is disabled.
		1	Receive CPPI DMA is enabled.

4.21 Receive CPPI Masked Status Register (RCPPIMSKSR)

The Receive CPPI Masked Status Register (RCPPIMSKSR) is shown in Figure 36 and described in Table 37.

Figure 36. Receive CPPI Masked Status Register (RCPPIMSKSR)

31	16
Reserved	
R-0	

15	4	3	0
Reserved		MASKED_COMP_PENDING	
R-0		R-0	

LEGEND: R = Read only; -*n* = value after reset

Table 37. Receive CPPI Masked Status Register (RCPPIMSKSR) Field Descriptions

Bit	Field	Value	Description
31-4	Reserved	0	Reserved
3-0	MASKED_COMP_PENDING	0-Fh	Masked Receive Completion Pending Indicators for channels 3 to 0 Raw Receive completion indicators bitwise ANDed with Receive completion mask bits

4.22 Receive CPPI Raw Status Register (RCPPIRAWSR)

The Receive CPPI Raw Status Register (RCPPIRAWSR) is shown in Figure 37 and described in Table 38.

Figure 37. Receive CPPI Raw Status Register (RCPPIRAWSR)

31	16
Reserved	
R-0	

15	4	3	0
Reserved		COMP_PENDING	
R-0		R-0	

LEGEND: R = Read only; -*n* = value after reset

Table 38. Receive CPPI Raw Status Register (RCPPIRAWSR) Field Descriptions

Bit	Field	Value	Description
31-4	Reserved	0	Reserved
3-0	COMP_PENDING	0-Fh	Raw Receive Completion Pending Indicators for channels 3 to 0 Active high flags which indicate that a packet has completed reception

4.23 Receive CPPI Interrupt Enable Set Register (RCPPIENSETR)

The Receive CPPI Interrupt Enable Set Register (RCPPIENSETR) is shown in Figure 38 and described in Table 39.

Figure 38. Receive CPPI Interrupt Enable Set Register (RCPPIENSETR)

31	16
Reserved	
R-0	

15	4	3	0
Reserved		COMP_PENDING_INTR_EN	
R-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

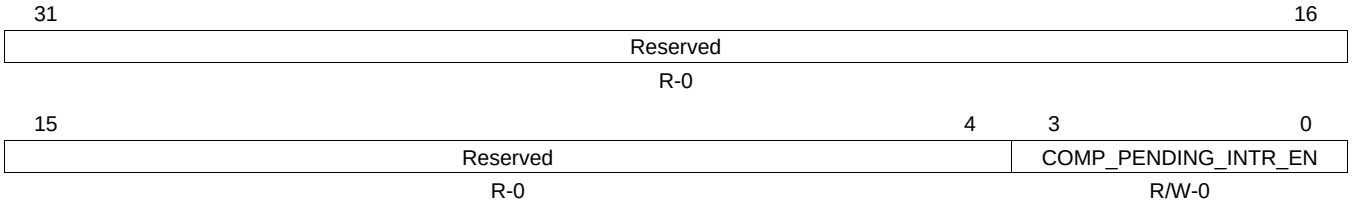
Table 39. Receive CPPI Interrupt Enable Set Register (RCPPIENSETR) Field Descriptions

Bit	Field	Value	Description
31-4	Reserved	0	Reserved
3-0	COMP_PENDING_INTR_EN	0-Fh	Receive CPPI Interrupt Enables These are active high interrupt enables corresponding to the Receive CPPI Completion Pending status bits. Writing a 1 to any of the bits in the Receive CPPI Interrupt Enable Set Register will result in setting of the corresponding bit in the Receive CPPI Interrupt Enable Register.

4.24 Receive CPPI Interrupt Enable Clear Register (RCPPIIENCLRR)

The Receive CPPI Interrupt Enable Clear Register (RCPPIIENCLRR) is shown in Figure 39 and described in Table 40.

Figure 39. Receive CPPI Interrupt Enable Clear Register (RCPPIIENCLRR)



LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

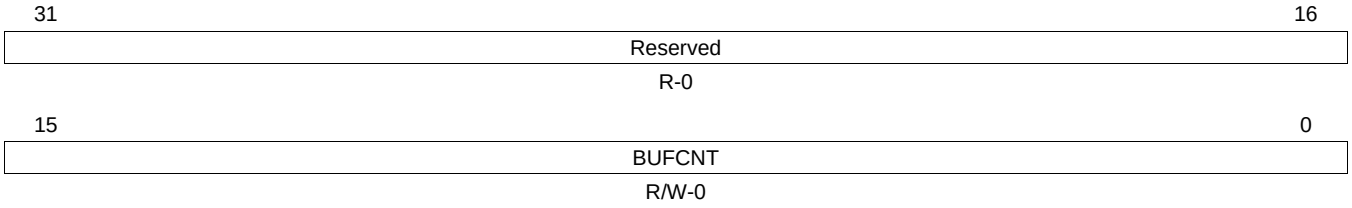
Table 40. Receive CPPI Interrupt Enable Clear Register (RCPPIIENCLRR) Field Descriptions

Bit	Field	Value	Description
31-4	Reserved	0	Reserved
3-0	COMP_PENDING_INTR_EN	0-Fh	Receive CPPI Interrupt Enables These are active high interrupt enables corresponding to the Receive CPPI Completion Pending status bits. Writing a 1 to any of the bits in the Receive CPPI Interrupt Enable Clear Register will result in clearing of the corresponding bit in the Receive CPPI Interrupt Enable Register.

4.25 Receive Buffer Count 0 Register (RBUFCNT0)

The Receive Buffer Count 0 Register (RBUFCNT0) is shown in Figure 40 and described in Table 41.

Figure 40. Receive Buffer Count 0 Register (RBUFCNT0)



LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

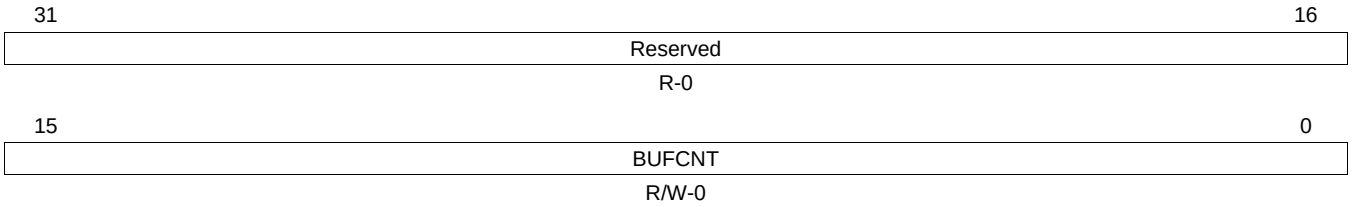
Table 41. Receive Buffer Count 0 Register (RBUFCNT0) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15-0	BUFCNT	0-FFFFh	Receive CPPI Buffer Count The current count of CPPI buffers in Receive channel 0 queue. Writes add to current value (not overwrite). The DMA requires a minimum of 3 RX buffers to operate.

4.26 Receive Buffer Count 1 Register (RBUFCNT1)

The Receive Buffer Count 1 Register (RBUFCNT1) is shown in Figure 41 and described in Table 42.

Figure 41. Receive Buffer Count 1 Register (RBUFCNT1)



LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

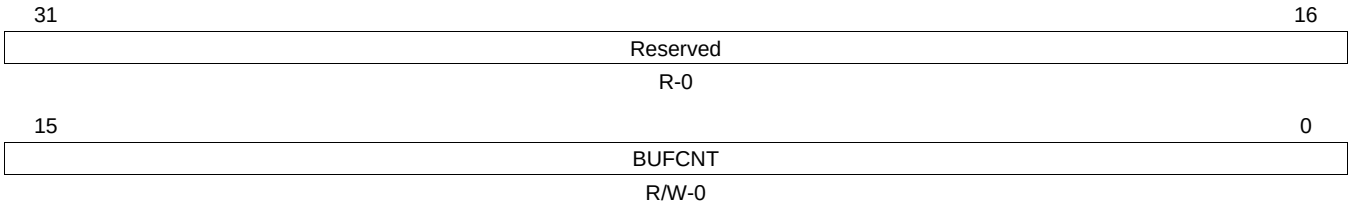
Table 42. Receive Buffer Count 1 Register (RBUFCNT1) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15-0	BUFCNT	0-FFFFh	Receive CPPI Buffer Count The current count of CPPI buffers in Receive channel 1 queue. Writes add to current value (not overwrite). The DMA requires a minimum of 3 RX buffers to operate.

4.27 Receive Buffer Count 2 Register (RBUFCNT2)

The Receive Buffer Count 2 Register (RBUFCNT2) is shown in Figure 42 and described in Table 43.

Figure 42. Receive Buffer Count 2 Register (RBUFCNT2)



LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

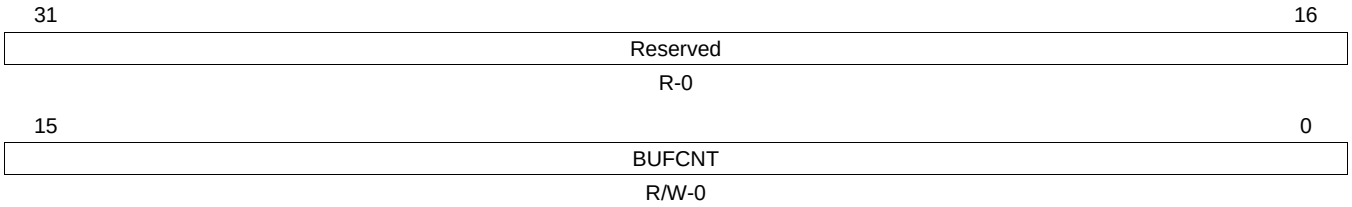
Table 43. Receive Buffer Count 2 Register (RBUFCNT2) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15-0	BUFCNT	0-FFFFh	Receive CPPI Buffer Count The current count of CPPI buffers in Receive channel 2 queue. Writes add to current value (not overwrite). The DMA requires a minimum of 3 RX buffers to operate.

4.28 Receive Buffer Count 3 Register (RBUFCNT3)

The Receive Buffer Count 3 Register (RBUFCNT3) is shown in Figure 43 and described in Table 44.

Figure 43. Receive Buffer Count 3 Register (RBUFCNT3)



LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

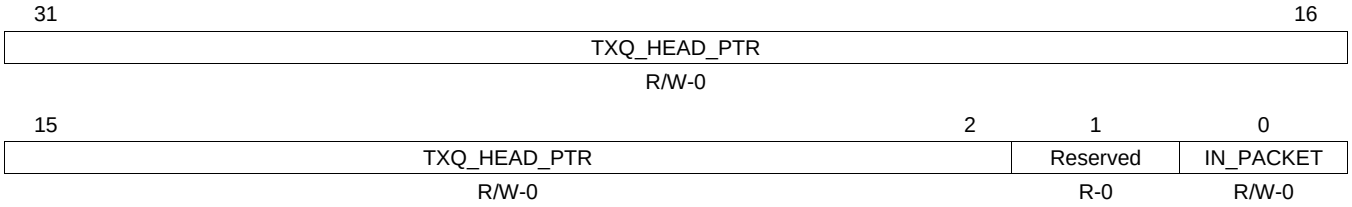
Table 44. Receive Buffer Count 3 Register (RBUFCNT3) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15-0	BUFCNT	0-FFFFh	Receive CPPI Buffer Count 0 The current count of CPPI buffers in Receive channel 3 queue. Writes add to current value (not overwrite). The DMA requires a minimum of 3 RX buffers to operate.

4.29 Transmit CPPI DMA State Word 0 (TCPPIDMASTATEW0)

The Transmit CPPI DMA State Word 0 (TCPPIDMASTATEW0) is shown in Figure 44 and described in Table 45.

Figure 44. Transmit CPPI DMA State Word 0 (TCPPIDMASTATEW0)



LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 45. Transmit CPPI DMA State Word 0 (TCPPIDMASTATEW0) Field Descriptions

Bit	Field	Value	Description
31-2	TXQ_HEAD_PTR	0-3FFF FFFFh	TX Queue Head Pointer 30-bit pointer to 32-bit aligned descriptor at the head of the high priority TX queue
1	Reserved	0	Reserved
0	IN_PACKET		Flag indicating the DMA is in the middle of processing a packet
		0	Not currently in packet
		1	Currently in packet

4.30 Transmit CPPI DMA State Word 1 (TCPPIDMASTATEW1)

The Transmit CPPI DMA State Word 1 (TCPPIDMASTATEW1) is shown in Figure 45 and described in Table 46.

Figure 45. Transmit CPPI DMA State Word 1 (TCPPIDMASTATEW1)

31	16
SOP_DESCRIPTOR_PTR	
R/W-0	

15	2	1	0
SOP_DESCRIPTOR_PTR		Reserved	IN_PACKET
R/W-0		R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 46. Transmit CPPI DMA State Word 1 (TCPPIDMASTATEW1) Field Descriptions

Bit	Field	Value	Description
31-2	SOP_DESCRIPTOR_PTR	0-3FFF FFFFh	Start of Packet Buffer Descriptor Pointer 30-bit pointer to 32-bit aligned descriptor which is the first descriptor for the current packet
1	Reserved	0	Reserved
0	IN_PACKET		Flag indicating the DMA is in the middle of processing a packet
		0	Not currently in packet
		1	Currently in packet

4.31 Transmit CPPI DMA State Word 2 (TCPPIDMASTATEW2)

The Transmit CPPI DMA State Word 2 (TCPPIDMASTATEW2) is shown in Figure 46 and described in Table 47.

Figure 46. Transmit CPPI DMA State Word 2 (TCPPIDMASTATEW2)

31	16
CURR_DESCRIPTOR_PTR	
R/W-0	

15	2	1	0
CURR_DESCRIPTOR_PTR		TRUNCATED_NON_EOP	Reserved
R/W-0		R/W-0	R-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 47. Transmit CPPI DMA State Word 2 (TCPPIDMASTATEW2) Field Descriptions

Bit	Field	Value	Description
31-2	CURR_DESCRIPTOR_PTR	0-3FFF FFFFh	Current Buffer Descriptor Pointer 30-bit pointer to 32-bit aligned descriptor for buffer from which data is currently being transmitted
1	TRUNCATED_NON_EOP		Flag indicating that packet truncation has occurred on the current packet within a buffer which was not designated as the end of packet buff
		0	Truncation within non end of packet buffer has not occurred
		1	Truncation within non end of packet has occurred
0	Reserved	0	Reserved

4.32 Transmit CPPI DMA State Word 3 (TCPPIDMASTATEW3)

The Transmit CPPI DMA State Word 3 (TCPPIDMASTATEW3) is shown in Figure 47 and described in Table 48.

Figure 47. Transmit CPPI DMA State Word 3 (TCPPIDMASTATEW3)

31	0
CURR_BUFFER_PTR	
R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 48. Transmit CPPI DMA State Word 3 (TCPPIDMASTATEW3) Field Descriptions

Bit	Field	Value	Description
31-0	CURR_BUFFER_PTR	0-FFFF FFFFh	Current Buffer Pointer 32-bit absolute byte address in buffer from which data is currently being transmitted

4.33 Transmit CPPI DMA State Word 4 (TCPPIDMASTATEW4)

The Transmit CPPI DMA State Word 4 (TCPPIDMASTATEW4) is shown in Figure 48 and described in Table 49.

Figure 48. Transmit CPPI DMA State Word 4 (TCPPIDMASTATEW4)

31 24	23 18	17	16
DESC_MSG	Reserved	CURR_BUFFER_EOP	CURR_BUFFER_SOP
R/W-0	R-0	R/W-0	R/W-0

15	0
CURR_BUFFER_LENGTH	
R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 49. Transmit CPPI DMA State Word 4 (TCPPIDMASTATEW4) Field Descriptions

Bit	Field	Value	Description
31-24	DESC_MSG	0-FFh	Descriptor Message Byte which is passed from the SOP Tx Descriptor to the SOP Receive descriptor. Used to tag packets.
23-18	Reserved	0	Reserved
17	CURR_BUFFER_EOP	0-1	Current Buffer is EOP Flag indicating whether or not the current buffer that is being transmitted from is the end of packet buffer
16	CURR_BUFFER_SOP	0-1	Current Buffer is SOP Flag indicating whether or not the current buffer that is being transmitted from is the start of packet buffer
15-0	CURR_BUFFER_LENGTH	0-FFFFh	Current Buffer Length Indicates how many valid bytes remain in the current buffer that is being transmitted from

4.34 Transmit CPPI DMA State Word 5 (TCPPIDMASTATEW5)

The Transmit CPPI DMA State Word 5 (TCPPIDMASTATEW5) is shown in Figure 49 and described in Table 50.

Figure 49. Transmit CPPI DMA State Word 5 (TCPPIDMASTATEW5)

31	16
REM_LENGTH	
R/W-0	

15	0
LENGTH	
R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 50. Transmit CPPI DMA State Word 5 (TCPPIDMASTATEW5) Field Descriptions

Bit	Field	Value	Description
31-16	REM_LENGTH	0-FFFFh	Remaining Packet Length Indicates how many bytes remain to be transmitted in the packet before truncation occurs
15-0	LENGTH	0-FFFFh	Packet Length Indicates how many bytes are contained in the packet. This value is copied from the packet-length field in Word 3 of the SOP Descriptor. This value is written into the AAL5 trailer for the packet.

4.35 Transmit CPPI Completion Pointer (TCPPICOMPPTR)

The Transmit CPPI Completion Pointer (TCPPICOMPPTR) is shown in Figure 50 and described in Table 51.

Figure 50. Transmit CPPI Completion Pointer (TCPPICOMPPTR)

31	16
DESC_ADDR	
R/W-0	

15	2	1	0
DESC_ADDR		Reserved	WRBK_MODE
R/W-0		R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 51. Transmit CPPI Completion Pointer (TCPPICOMPPTR) Field Descriptions

Bit	Field	Value	Description
31-2	DESC_ADDR	0-3FFF FFFFh	Descriptor Address This field contains the 30-bit word aligned pointer of the end of packet descriptor that the DMA has last processed
1	Reserved	0	Reserved
0	WRBK_MODE		Writeback/Compare Mode. This bit controls the action that is to be taken when this location is written.
		0	Compare Mode. Indicates that the value that is presented on bits 31:2 of the write data should be compared against the value that is currently contained in bits 31:2 of this location. If the two match, the interrupt bit corresponding to this Tx Queue should be deasserted.
		1	Writeback Mode. Indicates that the value that is presented on bits 31:2 of the write data should be written to this location and the interrupt for this Tx Queue should be asserted. This bit is read as zero.

4.36 Receive CPPI DMA State Word 0 (RCPPIDMASTATEW0)

The Receive CPPI DMA State Word 0 (RCPPIDMASTATEW0) is shown in Figure 51 and described in Table 52.

Figure 51. Receive CPPI DMA State Word 0 (RCPPIDMASTATEW0)

31	16
Reserved	
R-0	

15	8	7	0
Reserved		SOP_BUFFER_OFFSET	
R/W-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; *-n* = value after reset

Table 52. Receive CPPI DMA State Word 0 (RCPPIDMASTATEW0) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0-7FFF FFh	Reserved
7-0	SOP_BUFFER_OFFSET	0-FFh	Start of Packet Buffer Offset Controls how many bytes the Receive DMA will skip at the beginning of the start of packet buffer before beginning to fill the buffer with receive data. This field is programmed by the Host software when the channel is set up to facilitate addition of various protocol encapsulation headers for retransmission of received packets.

4.37 Receive CPPI DMA State Word 1 (RCPPIDMASTATEW1)

The Receive CPPI DMA State Word 1 (RCPPIDMASTATEW1) is shown in Figure 52 and described in Table 53.

Figure 52. Receive CPPI DMA State Word 1 (RCPPIDMASTATEW1)

31	16
RXQ_HEAD_PTR	
R/W-0	

15	2	1	0
RXQ_HEAD_PTR		Reserved	
R/W-0		R-0	

LEGEND: R/W = Read/Write; R = Read only; *-n* = value after reset

Table 53. Receive CPPI DMA State Word 1 (RCPPIDMASTATEW1) Field Descriptions

Bit	Field	Value	Description
31-2	RXQ_HEAD_PTR	0-1FFF FFFFh	Receive Queue Head Pointer 30-bit pointer to 32-bit aligned descriptor at the head of the Receive queue
1-0	Reserved	0	Reserved

4.38 Receive CPPI DMA State Word 2 (RCPPIDMASTATEW2)

The Receive CPPI DMA State Word 2 (RCPPIDMASTATEW2) is shown in Figure 53 and described in Table 54.

Figure 53. Receive CPPI DMA State Word 2 (RCPPIDMASTATEW2)

31	16
SOP_DESCRIPTOR_PTR	
R/W-0	

15	2	1	0
SOP_DESCRIPTOR_PTR		Reserved	IN_PACKET
R/W-0		R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 54. Receive CPPI DMA State Word 2 (RCPPIDMASTATEW2) Field Descriptions

Bit	Field	Value	Description
31-2	SOP_DESCRIPTOR_PTR	0-1FFF FFFFh	Start of Packet Buffer Descriptor Pointer 30-bit pointer to 32-bit aligned descriptor which is the first descriptor for the current packet
1	Reserved	0	Reserved
0	IN_PACKET		Flag indicating the DMA is in the middle of processing a packet 0: Not currently in packet
		0	Not currently in packet
		1	Currently in packet

4.39 Receive CPPI DMA State Word 3 (RCPPIDMASTATEW3)

The Receive CPPI DMA State Word 3 (RCPPIDMASTATEW3) is shown in Figure 54 and described in Table 55.

Figure 54. Receive CPPI DMA State Word 3 (RCPPIDMASTATEW3)

31	16
CURR_DESCRIPTOR_PTR	
R/W-0	

15	1	0
CURR_DESCRIPTOR_PTR		Reserved
R/W-0		R-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

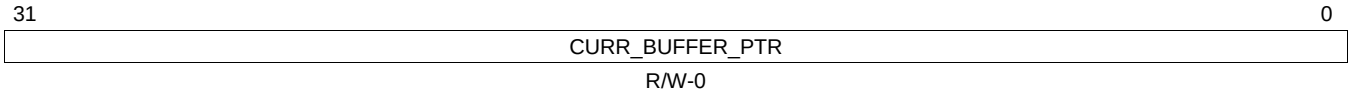
Table 55. Receive CPPI DMA State Word 3 (RCPPIDMASTATEW3) Field Descriptions

Bit	Field	Value	Description
31-2	CURR_DESCRIPTOR_PTR	0-3FFF FFFFh	Current Buffer Descriptor Pointer 30-bit pointer to 32-bit aligned descriptor for buffer into which data is currently being received
1-0	Reserved	0	Reserved

4.40 Receive CPPI DMA State Word 4 (RCPPIDMASTATEW4)

The Receive CPPI DMA State Word 4 (RCPPIDMASTATEW4) is shown in Figure 55 and described in Table 56.

Figure 55. Receive CPPI DMA State Word 4 (RCPPIDMASTATEW4)



LEGEND: R/W = Read/Write; -*n* = value after reset

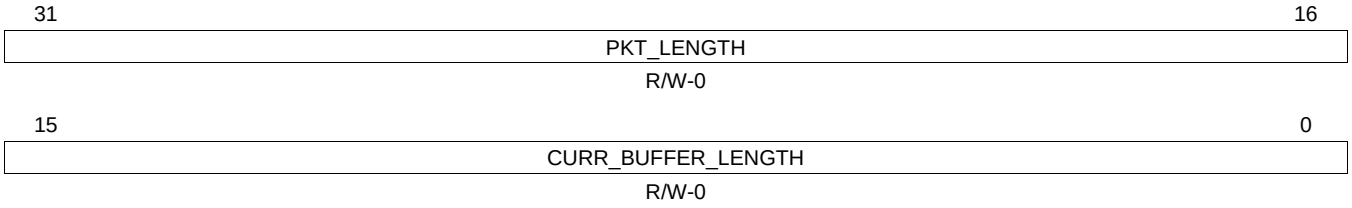
Table 56. Receive CPPI DMA State Word 4 (RCPPIDMASTATEW4) Field Descriptions

Bit	Field	Value	Description
31-0	CURR_BUFFER_PTR	0-FFFF FFFFh	Current Buffer Pointer 32-bit absolute byte address in buffer into which data is currently being received

4.41 Receive CPPI DMA State Word 5 (RCPPIDMASTATEW5)

The Receive CPPI DMA State Word 5 (RCPPIDMASTATEW5) is shown in Figure 56 and described in Table 57.

Figure 56. Receive CPPI DMA State Word 5 (RCPPIDMASTATEW5)



LEGEND: R/W = Read/Write; -*n* = value after reset

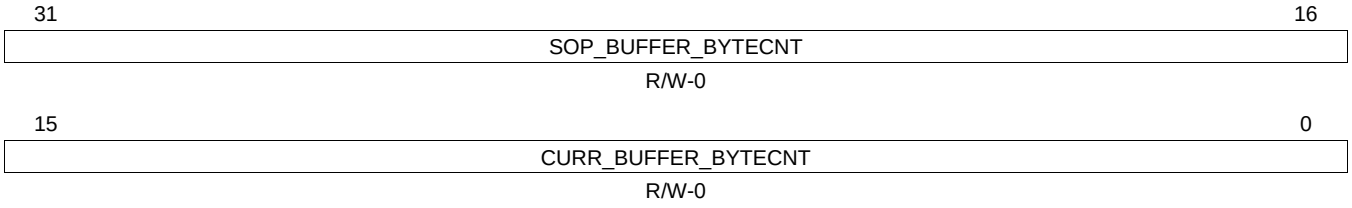
Table 57. Receive CPPI DMA State Word 5 (RCPPIDMASTATEW5) Field Descriptions

Bit	Field	Value	Description
31-16	PKT_LENGTH	0-FFFFh	Packet Length Indicates how many bytes are contained in the packet. This value is written into bits 15:0 of Word 3 of the SOP Descriptor during end of packet processing.
15-0	CURR_BUFFER_LENGTH	0-FFFFh	Current Buffer Length Indicates how many free bytes remain in the current buffer that is being received into

4.42 Receive CPPI DMA State Word 6 (RCPPIDMASTATEW6)

The Receive CPPI DMA State Word 6 (RCPPIDMASTATEW6) is shown in Figure 57 and described in Table 58.

Figure 57. Receive CPPI DMA State Word 6 (RCPPIDMASTATEW6)



LEGEND: R/W = Read/Write; -*n* = value after reset

Table 58. Receive CPPI DMA State Word 6 (RCPPIDMASTATEW6) Field Descriptions

Bit	Field	Value	Description
31-16	SOP_BUFFER_BYTECNT	0-FFFFh	Packet Length Indicates how many bytes are contained in the packet. This value is written into bits 15:0 of Word 3 of the SOP Descriptor during end of packet processing.
15-0	CURR_BUFFER_BYTECNT	0-FFFFh	Current Buffer Length Indicates how many free bytes remain in the current buffer that is being received into

4.43 Receive CPPI Completion Pointer (RCPPICOMPPTR)

The Receive CPPI Completion Pointer (RCPPICOMPPTR) is shown in Figure 58 and described in Table 59.

Figure 58. Receive CPPI Completion Pointer (RCPPICOMPPTR)

31	16
DESC_ADDR	
R/W-0	

15	2	1	0
DESC_ADDR		Reserved	RDBK_MODE
R/W-0		R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 59. Receive CPPI Completion Pointer (RCPPICOMPPTR) Field Descriptions

Bit	Field	Value	Description
31-2	DESC_ADDR	0-7FFF FFFFh	Descriptor Address This field contains the 30-bit word aligned pointer of the end of packet descriptor that the DMA has last processed.
1	Reserved	0	Reserved
0	RDBK_MODE		Readback / Compare Mode
		0	Compare Mode. Indicates that the value that is presented on bits 31:2 of the read data should be compared against the value that is currently contained in bits 31:2 of this location. If the two match, the interrupt bit corresponding to this Receive Queue should be deasserted.
		1	Readback Mode. Indicates that the value that is presented on bits 31:2 of the read data should be read from this location and the interrupt for this Receive Queue should be asserted. This bit is read as zero.

4.44 Function Address Register (FADDR)

The Function Address Register (FADDR) is shown in Figure 59 and described in Table 60.

Figure 59. Function Address Register (FADDR)

7	6	0
Reserved	FUNCADDR	
R-0	R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 60. Function Address Register (FADDR) Field Descriptions

Bit	Field	Value	Description
7	Reserved	0	Reserved
6-0	FUNCADDR	0-7Fh	7-bit address of the peripheral part of the transaction When used in Peripheral mode (DevCtl.D2=0), this register should be written with the address received through a SET_ADDRESS command, which will then be used for decoding the function address in subsequent token packets. When used in Host mode (DevCtl.D2=1), this register should be set to the value sent in a SET_ADDRESS command during device enumeration as the address for the peripheral device.

4.45 Power Management Register (POWER)

The Power Management Register (POWER) is shown in Figure 60 and described in Table 61.

Figure 60. Power Management Register (POWER)

7	6	5	4	3	2	1	0
ISOUPDATE	SOFTCONN	HSEN	HSMODE	RESET	RESUME	SUSPENDM	ENSUSPM
R/W-0	R/W-0	R/W-1	R-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; *-n* = value after reset

Table 61. Power Management Register (POWER) Field Descriptions

Bit	Field	Value	Description
7	ISOUPDATE	0-1	When set, the USB controller will wait for an SOF token from the time TxPktRdy is set before sending the packet. If an IN token is received before an SOF token, then a zero length data packet will be sent. Note: This is only valid in Peripheral Mode. This bit only affects endpoints performing Isochronous transfers.
6	SOFTCONN	0-1	If Soft Connect/Disconnect feature is enabled, then the USB D+/D- lines are enabled when this bit is set and tri-stated when this bit is cleared. Note: This is only valid in Peripheral Mode.
5	HSEN	0-1	When set, the USB controller will negotiate for high-speed mode when the device is reset by the hub. If not set, the device will only operate in full-speed mode.
4	HSMODE	0-1	This bit is set when the USB controller has successfully negotiated for high-speed mode.
3	RESET	0-1	This bit is set when Reset signaling is present on the bus. Note: This bit is Read/Write in Host Mode, but read-only in Peripheral Mode.
2	RESUME	0-1	Set to generate Resume signaling when the controller is in Suspend mode. The bit should be cleared after 10 ms (a maximum of 15 ms) to end Resume signaling. In Host mode, this bit is also automatically set when Resume signaling from the target is detected while the USB controller is suspended.
1	SUSPENDM	0-1	In Host mode, this bit should be set to enter Suspend mode. In Peripheral mode, this bit is set on entry into Suspend mode. It is cleared when the interrupt register is read, or the RESUME bit is set.
0	ENSUSPM	0-1	Set to enable the SUSPENDM output.

4.46 Interrupt Register for Endpoint 0 Plus Transmit Endpoints 1 to 4 (INTRTX)

The Interrupt Register for Endpoint 0 Plus Transmit Endpoints 1 to 4 (INTRTX) is shown in Figure 61 and described in Table 62.

NOTE: Unless the UINT bit in the control register (CTRLR) is set to 1 (non-PDR interrupt mode is enabled), do not read this register directly. Performing a read clears the pending interrupt. Use INTRTX only when in the non-PDR interrupt mode, that is, when handling the interrupt directly from the controller.

Figure 61. Interrupt Register for Endpoint 0 Plus Tx Endpoints 1 to 4 (INTRTX)

15	8
Reserved	
R-0	

7	5	4	3	2	1	0
Reserved		EP4TX	EP3TX	EP2TX	EP1TX	EP0
R-0		R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 62. Interrupt Register for Endpoint 0 Plus Transmit Endpoints 1 to 4 (INTRTX) Field Descriptions

Bit	Field	Value	Description
15-5	Reserved	0	Reserved
4	EP4TX	0-1	Transmit Endpoint 4 interrupt active
3	EP3TX	0-1	Transmit Endpoint 3 interrupt active
2	EP2TX	0-1	Transmit Endpoint 2 interrupt active
1	EP1TX	0-1	Transmit Endpoint 1 interrupt active
0	EP0	0-1	Endpoint 0 interrupt active

4.47 Interrupt Register for Receive Endpoints 1 to 4 (INTRRX)

The Interrupt Register for Receive Endpoints 1 to 4 (INTRRX) is shown in Figure 62 and described in Table 63.

NOTE: Unless the UINT bit in the control register (CTRLR) is set to 1 (non-PDR interrupt mode is enabled), do not read this register directly. Performing a read clears the pending interrupt. Use INTRRX only when in the non-PDR interrupt mode, that is, when handling the interrupt directly from the controller.

Figure 62. Interrupt Register for Receive Endpoints 1 to 4 (INTRRX)

31	16
Reserved	
R-0	

15	5	4	3	2	1	0
Reserved		EP4RX	EP3RX	EP2RX	EP1RX	Reserved
R-0		R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 63. Interrupt Register for Receive Endpoints 1 to 4 (INTRRX) Field Descriptions

Bit	Field	Value	Description
15-5	Reserved	0	Reserved
4	EP4RX	0-1	Receive Endpoint 4 interrupt active
3	EP3RX	0-1	Receive Endpoint 3 interrupt active
2	EP2RX	0-1	Receive Endpoint 2 interrupt active
1	EP1RX	0-1	Receive Endpoint 1 interrupt active
0	Reserved	0	Reserved

4.48 Interrupt Enable Register for INTRTX (INTRTXE)

The Interrupt Enable Register for INTRTX (INTRTXE) is shown in Figure 63 and described in Table 64.

Figure 63. Interrupt Enable Register for INTRTX (INTRTXE)

15						8
Reserved						
R-0						
7	5	4	3	2	1	0
Reserved		EP4TX	EP3TX	EP2TX	EP1TX	EP0
R-0		R/W-1	R/W-1	R/W-1	R/W-1	R/W-1

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 64. Interrupt Enable Register for INTRTX (INTRTXE) Field Descriptions

Bit	Field	Value	Description
15-5	Reserved	0	Reserved
4	EP4TX	0-1	1/0 = Transmit Endpoint 4 interrupt enable/disable
3	EP3TX	0-1	1/0 = Transmit Endpoint 3 interrupt enable/disable
2	EP2TX	0-1	1/0 = Transmit Endpoint 2 interrupt enable/disable
1	EP1TX	0-1	1/0 = Transmit Endpoint 1 interrupt enable/disable
0	EP0	0-1	1/0 = Endpoint 0 interrupt enable/disable

4.49 Interrupt Enable Register for INTRRX (INTRRXE)

The Interrupt Enable Register for INTRRX (INTRRXE) is shown in Figure 64 and described in Table 65.

Figure 64. Interrupt Enable Register for INTRRX (INTRRXE)

15						8
Reserved						
R-0						
7	5	4	3	2	1	0
Reserved		EP4RX	EP3RX	EP2RX	EP1RX	Reserved
R-0		R/W-1	R/W-1	R/W-1	R/W-1	R-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 65. Interrupt Enable Register for INTRRX (INTRRXE) Field Descriptions

Bit	Field	Value	Description
15-5	Reserved	0	Reserved
4	EP4RX	0-1	1/0 = Receive Endpoint 4 interrupt enable/disable
3	EP3RX	0-1	1/0 = Receive Endpoint 3 interrupt enable/disable
2	EP2RX	0-1	1/0 = Receive Endpoint 2 interrupt enable/disable
1	EP1RX	0-1	1/0 = Receive Endpoint 1 interrupt enable/disable
0	Reserved	0	Reserved

4.50 Interrupt Register for Common USB Interrupts (INTRUSB)

The Interrupt Register for Common USB Interrupts (INTRUSB) is shown in Figure 65 and described in Table 66. Reading this register causes all bits to be cleared.

NOTE: Unless the UINT bit in the control register (CTRLR) is set to 1 (non-PDR interrupt mode is enabled), do not read this register directly. Performing a read clears the pending interrupt. Use INTRUSB only when in the non-PDR interrupt mode, that is, when handling the interrupt directly from the controller.

Figure 65. Interrupt Register for Common USB Interrupts (INTRUSB)

7	6	5	4	3	2	1	0
VBUSERR	SESSREQ	DISCON	CONN	SOF	RESET_BABBLE	RESUME	SUSPEND
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 66. Interrupt Register for Common USB Interrupts (INTRUSB) Field Descriptions

Bit	Field	Value	Description
7	VBUSERR	0-1	Set when VBus drops below the VBus valid threshold during a session. Only valid when the USB controller is 'A' device. All active interrupts will be cleared when this register is read.
6	SESSREQ	0-1	Set when session request signaling has been detected. Only valid when USB controller is 'A' device.
5	DISCON	0-1	Set in host mode when a device disconnect is detected. Set in peripheral mode when a session ends.
4	CONN	0-1	Set when a device connection is detected. Only valid in host mode.
3	SOF	0-1	Set when a new frame starts.
2	RESET_BABBLE	0-1	Set in peripheral mode when reset signaling is detected on the bus set in host mode when babble is detected.
1	RESUME	0-1	Set when resume signaling is detected on the bus while the USB controller is in suspend mode.
0	SUSPEND	0-1	Set when suspend signaling is detected on the bus only valid in peripheral mode.

4.51 Interrupt Enable Register for INTRUSB (INTRUSBE)

The Interrupt Enable Register for INTRUSB (INTRUSBE) is shown in Figure 66 and described in Table 67.

Figure 66. Interrupt Enable Register for INTRUSB (INTRUSBE)

7	6	5	4	3	2	1	0
VBUSERR	SESSREQ	DISCON	CONN	SOF	RESET_BABBLE	RESUME	SUSPEND
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-1	R/W-1	R/W-0

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 67. Interrupt Enable Register for INTRUSB (INTRUSBE) Field Descriptions

Bit	Field	Value	Description
7	VBUSERR	0-1	Vbus error interrupt enable
6	SESSREQ	0-1	Session request interrupt enable
5	DISCON	0-1	Disconnect interrupt enable
4	CONN	0-1	Connect interrupt enable
3	SOF	0-1	Start of frame interrupt enable
2	RESET_BABBLE	0-1	Reset interrupt enable
1	RESUME	0-1	Resume interrupt enable
0	SUSPEND	0-1	Suspend interrupt enable

4.52 Frame Number Register (FRAME)

The Frame Number Register (FRAME) is shown in Figure 67 and described in Table 68.

Figure 67. Frame Number Register (FRAME)

15	11	10	0
Reserved		FRAMENUMBER	
R-0		R-0	

LEGEND: R = Read only; -*n* = value after reset

Table 68. Frame Number Register (FRAME) Field Descriptions

Bit	Field	Value	Description
15-11	Reserved	0	Reserved
10-0	FRAMENUMBER	0-7FFh	Last received frame number

4.53 Index Register for Selecting the Endpoint Status and Control Registers (INDEX)

The Index Register for Selecting the Endpoint Status and Control Registers (INDEX) is shown in Figure 68 and described in Table 69.

Figure 68. Index Register for Selecting the Endpoint Status and Control Registers (INDEX)

7	4	3	0
Reserved		EPSEL	
R-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 69. Index Register for Selecting the Endpoint Status and Control Registers (INDEX) Field Descriptions

Bit	Field	Value	Description
7-4	Reserved	0	Reserved
3-0	EPSEL	0-Fh	Each transmit endpoint and each receive endpoint have their own set of control/status registers. EPSEL determines which endpoint control/status registers are accessed. Before accessing an endpoint's control/status registers, the endpoint number should be written to the Index register to ensure that the correct control/status registers appear in the memory map.

4.54 Register to Enable the USB 2.0 Test Modes (TESTMODE)

The Register to Enable the USB 2.0 Test Modes (TESTMODE) is shown in Figure 69 and described in Table 70.

Figure 69. Register to Enable the USB 2.0 Test Modes (TESTMODE)

7	6	5	4	3	2	1	0
FORCE_HOST	FIFO_ACCESS	FORCE_FS	FORCE_HS	TEST_PACKET	TEST_K	TEST_J	TEST_SE0_NAK
R/W-0	W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; W = Write only; -*n* = value after reset

Table 70. Register to Enable the USB 2.0 Test Modes (TESTMODE) Field Descriptions

Bit	Field	Value	Description
7	FORCE_HOST	0-1	Set this bit to forcibly put the USB controller into Host mode when SESSION bit is set, regardless of whether it is connected to any peripheral. The controller remains in Host mode until the Session bit is cleared, even if a device is disconnected. And if the FORCE_HOST but remains set, it will re-enter Host mode next time the SESSION bit is set. The operating speed is determined using the FORCE_HS and FORCE_FS bits.
6	FIFO_ACCESS	0-1	Set this bit to transfer the packet in EP0 Tx FIFO to EP0 Receive FIFO. It is cleared automatically.
5	FORCE_FS	0-1	Set this bit to force the USB controller into full-speed mode when it receives a USB reset.
4	FORCE_HS	0-1	Set this bit to force the USB controller into high-speed mode when it receives a USB reset.
3	TEST_PACKET	0-1	Set this bit to enter the Test_Packet test mode. In this mode, the USB controller repetitively transmits a 53-byte test packet on the bus, the form of which is defined in the Universal Serial Bus Specification Revision 2.0. Note: The test packet has a fixed format and must be loaded into the Endpoint 0 FIFO before the test mode is entered.
2	TEST_K	0-1	Set this bit to enter the Test_K test mode. In this mode, the USB controller transmits a continuous K on the bus.
1	TEST_J	0-1	Set this bit to enter the Test_J test mode. In this mode, the USB controller transmits a continuous J on the bus.
0	TEST_SE0_NAK	0-1	Set this bit to enter the Test_SE0_NAK test mode. In this mode, the USB controller remains in high-speed mode, but responds to any valid IN token with a NAK.

4.55 Maximum Packet Size for Peripheral/Host Transmit Endpoint (TXMAXP)

The Maximum Packet Size for Peripheral/Host Transmit Endpoint (TXMAXP) is shown in Figure 70 and described in Table 71.

Figure 70. Maximum Packet Size for Peripheral/Host Transmit Endpoint (TXMAXP)

15	11	10	0
Reserved		MAXPAYLOAD	
R-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 71. Maximum Packet Size for Peripheral/Host Transmit Endpoint (TXMAXP) Field Descriptions

Bit	Field	Value	Description
15-11	Reserved	0	Reserved
10-0	MAXPAYLOAD	0-FFh	The maximum payload transmitted in a single transaction. The value set can be up to 1024 bytes, but is subject to the constraints placed by the USB Specification on packet sizes for Bulk, Interrupt, and Isochronous transfers in full-speed and high-speed operations. The value written to this register should match the wMaxPacketSize field of the Standard Endpoint Descriptor for the associated endpoint. A mismatch could cause unexpected results.

4.56 Control Status Register for Endpoint 0 in Peripheral Mode (PERI_CSR0)

The Control Status Register for Endpoint 0 in Peripheral Mode (PERI_CSR0) is shown in Figure 71 and described in Table 72.

Figure 71. Control Status Register for Endpoint 0 in Peripheral Mode (PERI_CSR0)

15						9	8
Reserved							FLUSHFIFO
R-0							W-0
7	6	5	4	3	2	1	0
SERV_SETUPEND	SERV_RXPKTRDY	SENDSTALL	SETUPEND	DATAEND	SENTSTALL	TXPKTRDY	RXPKTRDY
W-0	W-0	W-0	R-0	W-0	R/W-0	R/W-0	R-0

LEGEND: R/W = Read/Write; R = Read only; W = Write only; -*n* = value after reset

Table 72. Control Status Register for Endpoint 0 in Peripheral Mode (PERI_CSR0) Field Descriptions

Bit	Field	Value	Description
15-9	Reserved	0	Reserved
8	FLUSHFIFO	0-1	Set this bit to flush the next packet to be transmitted/read from the Endpoint 0 FIFO. The FIFO pointer is reset and the TXPKTRDY/RXPKTRDY bit is cleared. Note: FLUSHFIFO has no effect unless TXPKTRDY/RXPKTRDY is set.
7	SERV_SETUPEND	0-1	Set this bit to clear the SETUPEND bit. It is cleared automatically.
6	SERV_RXPKTRDY	0-1	Set this bit to clear the RXPKTRDY bit. It is cleared automatically.
5	SENDSTALL	0-1	Set this bit to terminate the current transaction. The STALL handshake will be transmitted and then this bit will be cleared automatically.
4	SETUPEND	0-1	This bit will be set when a control transaction ends before the DATAEND bit has been set. An interrupt will be generated, and the FIFO will be flushed at this time. The bit is cleared by the writing a 1 to the SERV_SETUPEND bit.
3	DATAEND	0-1	Set this bit to: 1 - When setting TXPKTRDY for the last data packet 2 - When clearing RXPKTRDY after unloading the last data packet 3 - When setting TXPKTRDY for a zero length data packet. It is cleared automatically.
2	SENTSTALL	0-1	This bit is set when a STALL handshake is transmitted. This bit should be cleared.
1	TXPKTRDY	0-1	Set this bit after loading a data packet into the FIFO. It is cleared automatically when the data packet has been transmitted. An interrupt is generated (if enabled) when the bit is cleared.
0	RXPKTRDY	0-1	This bit is set when a data packet has been received. An interrupt is generated when this bit is set. This bit is cleared by setting the SERV_RXPKTRDY bit.

4.57 Control Status Register for Endpoint 0 in Host Mode (HOST_CSR0)

The Control Status Register for Endpoint 0 in Host Mode (HOST_CSR0) is shown in Figure 72 and described in Table 73.

Figure 72. Control Status Register for Endpoint 0 in Host Mode (HOST_CSR0)

15					11	10	9	8
Reserved						DATATOGWREN	DATATOG	FLUSHFIFO
R-0						W-0	R/W-0	W-0
7	6	5	4	3	2	1	0	
NAK_TIMEOUT	STATUSPKT	REQPKT	ERROR	SETUPPKT	RXSTALL	TXPKTRDY	RXPKTRDY	
W-0	R/W-0	R/W-0	W-0	R/W-0	R/W-0	R/W-0	R/W-0	

LEGEND: R/W = Read/Write; R = Read only; W = Write only; -*n* = value after reset

Table 73. Control Status Register for Endpoint 0 in Host Mode (HOST_CSR0) Field Descriptions

Bit	Field	Value	Description
15-11	Reserved	0	Reserved
10	DATATOGWREN	0-1	Set this bit to enable the DATATOG bit to be written. This bit is automatically cleared once the new value is written to DATATOG.
9	DATATOG	0-1	When read, this bit indicates the current state of the EP0 data toggle. If DATATOGWREN is high, this bit can be written with the required setting of the data toggle. If DATATOGWREN is low, any value written to this bit is ignored.
8	FLUSHFIFO	0-1	Set this bit to flush the next packet to be transmitted/read from the Endpoint 0 FIFO. The FIFO pointer is reset and the TXPKTRDY/RXPKTRDY bit is cleared. Note: FLUSHFIFO has no effect unless TXPKTRDY/RXPKTRDY is set.
7	NAK_TIMEOUT	0-1	This bit will be set when Endpoint 0 is halted following the receipt of NAK responses for longer than the time set by the NAKLIMIT0 register. This bit should be cleared to allow the endpoint to continue.
6	STATUSPKT	0-1	Set this bit at the same time as the TXPKTRDY or REQPKT bit is set, to perform a status stage transaction. Setting this bit ensures that the data toggle is set so that a DATA1 packet is used for the Status Stage transaction.
5	REQPKT	0-1	Set this bit to request an IN transaction. It is cleared when RXPKTRDY is set.
4	ERROR	0-1	This bit will be set when three attempts have been made to perform a transaction with no response from the peripheral. You should clear this bit. An interrupt is generated when this bit is set.
3	SETUPPKT	0-1	Set this bit, at the same time as the TXPKTRDY bit is set, to send a SETUP token instead of an OUT token for the transaction.
2	RXSTALL	0-1	This bit is set when a STALL handshake is received. You should clear this bit.
1	TXPKTRDY	0-1	Set this bit after loading a data packet into the FIFO. It is cleared automatically when the data packet has been transmitted. An interrupt is generated (if enabled) when the bit is cleared.
0	RXPKTRDY	0-1	This bit is set when a data packet has been received. An interrupt is generated when this bit is set. Clear this bit by setting the SERV_RXPKTRDY bit.

4.58 Control Status Register for Peripheral Transmit Endpoint (PERI_TXCSR)

The Control Status Register for Peripheral Transmit Endpoint (PERI_TXCSR) is shown in Figure 73 and described in Table 74.

Figure 73. Control Status Register for Peripheral Transmit Endpoint (PERI_TXCSR)

15	14	13	12	11	10	9 7
Reserved	ISO	MODE	DMAEN	FRCDATATOG	DMAMODE	Reserved
R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0

6	5	4	3	2	1	0
CLRDATATOG	SENTSTALL	SENDSTALL	FLUSHFIFO	UNDERRUN	FIFONOTEMPTY	TXPKTRDY
W-0	R/W-0	R/W-0	W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; W = Write only; -*n* = value after reset

Table 74. Control Status Register for Peripheral Transmit Endpoint (PERI_TXCSR) Field Descriptions

Bit	Field	Value	Description
15	Reserved	0	Reserved
14	ISO	0-1	Set this bit to enable the Tx endpoint for Isochronous transfers, and clear this bit to enable the Tx endpoint for Bulk or Interrupt transfers.
13	MODE	0-1	Set this bit to enable the endpoint direction as Tx, and clear this bit to enable it as Rx. Note: This bit has any effect only where the same endpoint FIFO is used for both Transmit and Receive transactions.
12	DMAEN	0-1	Set this bit to enable the DMA request for the Tx endpoint.
11	FRCDATATOG	0-1	Set this bit to force the endpoint data toggle to switch and the data packet to be cleared from the FIFO, regardless of whether an ACK was received. This can be used by Interrupt Tx endpoints that are used to communicate rate feedback for Isochronous endpoints.
10	DMAMODE	0-1	When using DMA, clear this bit to receive an interrupt for each packet, or set this bit to only receive error interrupts.
9-7	Reserved	0	Reserved
6	CLRDATATOG	0-1	Set this bit to reset the endpoint data toggle to 0.
5	SENTSTALL	0-1	This bit is set automatically when a STALL handshake is transmitted. The FIFO is flushed and the TXPKTRDY bit is cleared. You should clear this bit.
4	SENDSTALL	0-1	Set this bit to issue a STALL handshake to an IN token. Clear this bit to terminate the stall condition. Note: This bit has no effect where the endpoint is being used for Isochronous transfers.
3	FLUSHFIFO	0-1	Set this bit to flush the next packet to be transmitted from the endpoint Tx FIFO. The FIFO pointer is reset and the TXPKTRDY bit is cleared. Note: FlushFIFO has no effect unless TXPKTRDY is set. Also note that, if the FIFO is double-buffered, FlushFIFO may need to be set twice to completely clear the FIFO.
2	UNDERRUN	0-1	This bit is set automatically if an IN token is received when TXPKTRDY is not set. You should clear this bit.
1	FIFONOTEMPTY	0-1	This bit is set when there is at least 1 packet in the Tx FIFO. You should clear this bit.
0	TXPKTRDY	0-1	Set this bit after loading a data packet into the FIFO. It is cleared automatically when a data packet has been transmitted. An interrupt is generated (if enabled) when the bit is cleared.

4.59 Control Status Register for Host Transmit Endpoint (HOST_TXCSR)

The Control Status Register for Host Transmit Endpoint (HOST_TXCSR) is shown in Figure 74 and described in Table 75.

Figure 74. Control Status Register for Host Transmit Endpoint (HOST_TXCSR)

15	14	13	12	11	10	9	8
Reserved		MODE	DMAEN	FRCDATATOG	DMAMODE	DATATOGWREN	DATATOG
R-0		R/W-0	R/W-0	R/W-0	R/W-0	W-0	R/W-0

7	6	5	4	3	2	1	0
NAK_TIMEOUT	CLRDATATOG	RXSTALL	SETUPPKT	FLUSHFIFO	ERROR	FIFONOTEMPTY	TXPKTRDY
R/W-0	W-0	R/W-0	R/W-0	W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; W = Write only; -*n* = value after reset

Table 75. Control Status Register for Host Transmit Endpoint (HOST_TXCSR) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved
13	MODE	0-1	Set this bit to enable the endpoint direction as Tx, and clear this bit to enable it as Rx. Note: This bit has any effect only where the same endpoint FIFO is used for both Transmit and Receive transactions.
12	DMAEN	0-1	Set this bit to enable the DMA request for the Tx endpoint.
11	FRCDATATOG	0-1	Set this bit to force the endpoint data toggle to switch and the data packet to be cleared from the FIFO, regardless of whether an ACK was received. This can be used by Interrupt Tx endpoints that are used to communicate rate feedback for Isochronous endpoints.
10	DMAMODE	0-1	When using DMA, clear this bit to receive an interrupt for each packet, or set this bit to only receive error interrupts.
9	DATATOGWREN	0-1	Set this bit to enable the DATATOG bit to be written. This bit is automatically cleared once the new value is written to DATATOG.
8	DATATOG	0-1	When read, this bit indicates the current state of the Tx EP data toggle. If DATATOGWREN is high, this bit can be written with the required setting of the data toggle. If DATATOGWREN is low, any value written to this bit is ignored.
7	NAK_TIMEOUT	0-1	This bit will be set when the Tx endpoint is halted following the receipt of NAK responses for longer than the time set as the NAKLIMIT by the TXINTERVAL register. It should be cleared to allow the endpoint to continue. Note: This is valid only for Bulk endpoints.
6	CLRDATATOG	0-1	Set this bit to reset the endpoint data toggle to 0.
5	RXSTALL	0-1	This bit is set when a STALL handshake is received. The FIFO is flushed and the TXPKTRDY bit is cleared. You should clear this bit.
4	SETUPPKT	0-1	Set this bit at the same time as TXPKTRDY is set, to send a SETUP token instead of an OUT token for the transaction. Note: Setting this bit also clears the DATATOG bit.
3	FLUSHFIFO	0-1	Set this bit to flush the next packet to be transmitted from the endpoint Tx FIFO. The FIFO pointer is reset and the TXPKTRDY bit is cleared. Note: FlushFIFO has no effect unless TXPKTRDY is set. Also note that, if the FIFO is double-buffered, FLUSHFIFO may need to be set twice to completely clear the FIFO.
2	ERROR	0-1	The USB controller sets this bit when 3 attempts have been made to send a packet and no handshake packet has been received. You should clear this bit. An interrupt is generated when the bit is set. This is valid only when the endpoint is operating in Bulk or Interrupt mode.
1	FIFONOTEMPTY	0-1	The USB controller sets this bit when there is at least 1 packet in the Tx FIFO.
0	TXPKTRDY	0-1	Set this bit after loading a data packet into the FIFO. It is cleared automatically when a data packet has been transmitted. An interrupt is generated (if enabled) when the bit is cleared.

4.60 Maximum Packet Size for Peripheral Host Receive Endpoint (RXMAXP)

The Maximum Packet Size for Peripheral Host Receive Endpoint (RXMAXP) is shown in Figure 75 and described in Table 76.

Figure 75. Maximum Packet Size for Peripheral Host Receive Endpoint (RXMAXP)

15	11	10	0
Reserved		MAXPAYLOAD	
R-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 76. Maximum Packet Size for Peripheral Host Receive Endpoint (RXMAXP) Field Descriptions

Bit	Field	Value	Description
15-11	Reserved	0	Reserved
10-0	MAXPAYLOAD	0-FFh	Defines the maximum amount of data that can be transferred through the selected Receive endpoint in a single frame/microframe (high-speed transfers). The value set can be up to 1024 bytes, but is subject to the constraints placed by the USB Specification on packet sizes for Bulk, Interrupt, and Isochronous transfers in full-speed and high-speed operations. The value written to this register should match the wMaxPacketSize field of the Standard Endpoint Descriptor for the associated endpoint. A mismatch could cause unexpected results.

4.61 Control Status Register for Peripheral Receive Endpoint (PERI_RXCSR)

The Control Status Register for Peripheral Receive Endpoint (PERI_RXCSR) is shown in Figure 76 and described in Table 77.

Figure 76. Control Status Register for Peripheral Receive Endpoint (PERI_RXCSR)

15	14	13	12	11	10		8
Reserved	ISO	DMAEN	DISNYET	DMAMODE	Reserved		
R-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0		
7	6	5	4	3	2	1	0
CLRDATATOG	SENTSTALL	SENDSTALL	FLUSHFIFO	DATAERROR	OVERRUN	FIFOFULL	RXPKTRDY
W-0	R/W-0	R/W-0	W-0	R-0	R/W-0	R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; W = Write only; -*n* = value after reset

Table 77. Control Status Register for Peripheral Receive Endpoint (PERI_RXCSR) Field Descriptions

Bit	Field	Value	Description
15	Reserved	0	Reserved
14	ISO	0-1	Set this bit to enable the Receive endpoint for Isochronous transfers, and clear this bit to enable the Receive endpoint for Bulk/Interrupt transfers.
13	DMAEN	0-1	Set this bit to enable the DMA request for the Receive endpoints.
12	DISNYET	0-1	Set this bit to disable the sending of NYET handshakes. When set, all successfully received Receive packets are ACKed, including at the point at which the FIFO becomes full. Note: This bit only has any effect in high-speed mode, in which mode it should be set for all Interrupt endpoints.
11	DMAMODE	0	This bit should always be cleared to 0.
10-8	Reserved	0	Reserved
7	CLRDATATOG	0-1	Set this bit to reset the endpoint data toggle to 0.
6	SENTSTALL	0-1	This bit is set when a STALL handshake is transmitted. The FIFO is flushed and the TXPKTRDY bit is cleared. You should clear this bit.
5	SENDSTALL	0-1	Set this bit to issue a STALL handshake. Clear this bit to terminate the stall condition. Note: This bit has no effect where the endpoint is being used for Isochronous transfers.
4	FLUSHFIFO	0-1	Set this bit to flush the next packet to be read from the endpoint Receive FIFO. The FIFO pointer is reset and the RXPKTRDY bit is cleared. Note: FLUSHFIFO has no effect unless RXPKTRDY is set. Also note that, if the FIFO is double-buffered, FLUSHFIFO may need to be set twice to completely clear the FIFO.
3	DATAERROR	0-1	This bit is set when RXPKTRDY is set if the data packet has a CRC or bit-stuff error. It is cleared when RXPKTRDY is cleared. Note: This bit is only valid when the endpoint is operating in ISO mode. In Bulk mode, it always returns zero.
2	OVERRUN	0-1	This bit is set if an OUT packet cannot be loaded into the Receive FIFO. You should clear this bit. Note: This bit is only valid when the endpoint is operating in ISO mode. In Bulk mode, it always returns zero.
1	FIFOFULL	0-1	This bit is set when no more packets can be loaded into the Receive FIFO.
0	RXPKTRDY	0-1	This bit is set when a data packet has been received. You should clear this bit when the packet has been unloaded from the Receive FIFO. An interrupt is generated when the bit is set.

4.62 Control Status Register for Host Receive Endpoint (HOST_RXCSR)

The Control Status Register for Host Receive Endpoint (HOST_RXCSR) is shown in Figure 77 and described in Table 78.

Figure 77. Control Status Register for Host Receive Endpoint (HOST_RXCSR)

15	14	13	12	11	10	9	8
Reserved		DMAEN	DISNYET	DMAMODE	DATATOGWREN	DATATOG	Reserved
R-0		R/W-0	R/W-0	R/W-0	W-0	R/W-0	R-0

7	6	5	4	3	2	1	0
CLRDATATOG	RXSTALL	REQPKT	FLUSHFIFO	DATAERR_NAK TIMEOUT	ERROR	FIFOFULL	RXPKTRDY
W-0	R/W-0	R/W-0	W-0	R-0	R/W-0	R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; W = Write only; -*n* = value after reset

Table 78. Control Status Register for Host Receive Endpoint (HOST_RXCSR) Field Descriptions

Bit	Field	Value	Description
15-14	Reserved	0	Reserved
13	DMAEN	0-1	Set this bit to enable the DMA request for the Receive endpoints.
12	DISNYET	0-1	Set this bit to disable the sending of NYET handshakes. When set, all successfully received Receive packets are ACKED including at the point at which the FIFO becomes full. Note: This bit only has any effect in high-speed mode, in which mode it should be set for all Interrupt endpoints.
11	DMAMODE	0	This bit should always be cleared to 0.
10	DATATOGWREN	0-1	Set this bit to enable the DATATOG bit to be written. This bit is automatically cleared once the new value is written to DATATOG.
9	DATATOG	0-1	When read, this bit indicates the current state of the Receive EP data toggle. If DATATOGWREN is high, this bit can be written with the required setting of the data toggle. If DATATOGWREN is low, any value written to this bit is ignored.
8	Reserved	0	Reserved
7	CLRDATATOG	0-1	Set this bit to reset the endpoint data toggle to 0.
6	RXSTALL	0-1	When a STALL handshake is received, this bit is set and an interrupt is generated. You should clear this bit.
5	REQPKT	0-1	Set this bit to request an IN transaction. It is cleared when RXPKTRDY is set.
4	FLUSHFIFO	0-1	Set this bit to flush the next packet to be read from the endpoint Receive FIFO. The FIFO pointer is reset and the RXPKTRDY bit is cleared. Note: FLUSHFIFO has no effect unless RXPKTRDY is set. Also note that, if the FIFO is double-buffered, FLUSHFIFO may need to be set twice to completely clear the FIFO.
3	DATAERR_NAKTIMEOUT	0-1	When operating in ISO mode, this bit is set when RXPKTRDY is set if the data packet has a CRC or bit-stuff error and cleared when RXPKTRDY is cleared. In Bulk mode, this bit will be set when the Receive endpoint is halted following the receipt of NAK responses for longer than the time set as the NAK Limit by the RXINTERVAL register. You should clear this bit to allow the endpoint to continue.
2	ERROR	0-1	The USB controller sets this bit when 3 attempts have been made to receive a packet and no data packet has been received. You should clear this bit. An interrupt is generated when the bit is set. Note: This bit is only valid when the transmit endpoint is operating in Bulk or Interrupt mode. In ISO mode, it always returns zero.
1	FIFOFULL	0-1	This bit is set when no more packets can be loaded into the Receive FIFO.
0	RXPKTRDY	0-1	This bit is set when a data packet has been received. You should clear this bit when the packet has been unloaded from the Receive FIFO. An interrupt is generated when the bit is set.

4.63 Count 0 Register (COUNT0)

The Count 0 Register (COUNT0) is shown in Figure 78 and described in Table 79.

Figure 78. Count 0 Register (COUNT0)

15	7	6	0
Reserved		EP0RXCOUNT	
R-0		R-0	

LEGEND: R = Read only; -*n* = value after reset

Table 79. Count 0 Register (COUNT0) Field Descriptions

Bit	Field	Value	Description
15-7	Reserved	0	Reserved
6-0	EP0RXCOUNT	0-7Fh	Indicates the number of received data bytes in the Endpoint 0 FIFO. The value returned changes as the contents of the FIFO change and is only valid while RXPKTRDY of PERI_CSR0 or HOST_CSR0 is set.

4.64 Receive Count Register (RXCOUNT)

The Receive Count Register (RXCOUNT) is shown in Figure 79 and described in Table 80.

Figure 79. Receive Count Register (RXCOUNT)

15	13	12	0
Reserved		EPRXCOUNT	
R-0		R-0	

LEGEND: R = Read only; -*n* = value after reset

Table 80. Receive Count Register (RXCOUNT) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved
12-0	EPRXCOUNT	0-1FFFh	Holds the number of received data bytes in the packet in the Receive FIFO. The value returned changes as the contents of the FIFO change and is only valid while RXPKTRDY of PERI_RXCSR or HOST_RXCSR is set.

4.65 Type Register (Host mode only) (HOST_TYPE0)

The Type Register (Host mode only) (HOST_TYPE0) is shown in Figure 80 and described in Table 81.

Figure 80. Type Register (Host mode only) (HOST_TYPE0)

7	6	5	0
SPEED		Reserved	
R/W-0		R-0	

LEGEND: R/W = Read/Write; R = Read only; *-n* = value after reset

Table 81. Type Register (Host mode only) (HOST_TYPE0) Field Descriptions

Bit	Field	Value	Description
7-6	SPEED	0-3h	Operating Speed of Target Device
		0	Illegal
		1h	High
		2h	Full
		3h	Low
5-0	Reserved	0	Reserved

4.66 Transmit Type Register (Host mode only) (HOST_TXTYPE)

The Transmit Type Register (Host mode only) (HOST_TXTYPE) is shown in Figure 81 and described in Table 82.

Figure 81. Transmit Type Register (Host mode only) (HOST_TXTYPE)

7	6	5	4	3	0
SPEED		PROT		TENDPN	
R/W-0		R/W-0		R/W-0	

LEGEND: R/W = Read/Write; *-n* = value after reset

Table 82. Transmit Type Register (Host mode only) (HOST_TXTYPE) Field Descriptions

Bit	Field	Value	Description
7-6	SPEED	0-3h	Operating Speed of Target Device
		0	Illegal
		1h	High
		2h	Full
		3h	Low
5-4	PROT	0-3h	Set this to select the required protocol for the transmit endpoint
		0	Control
		1h	Isochronous
		2h	Bulk
		3h	Interrupt
3-0	TENDPN	0-Fh	Set this value to the endpoint number contained in the transmit endpoint descriptor returned to the USB controller during device enumeration.

4.67 NAKLimit0 Register (Host mode only) (HOST_NAKLIMIT0)

The NAKLimit0 Register (Host mode only) (HOST_NAKLIMIT0) is shown in Figure 82 and described in Table 83.

Figure 82. NAKLimit0 Register (Host mode only) (HOST_NAKLIMIT0)

7	5	4	0
Reserved		EP0NAKLIMIT	
R-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 83. NAKLimit0 Register (Host mode only) (HOST_NAKLIMIT0) Field Descriptions

Bit	Field	Value	Description
7-5	Reserved	0	Reserved
4-0	EP0NAKLIMIT	0-1Fh	Sets the number of frames/microframes (high-speed transfers) after which Endpoint 0 should time out on receiving a stream of NAK responses. The number of frames/microframes selected is $2^{(m-1)}$ (where m is the value set in the register, valid values 2-16). If the host receives NAK responses from the target for more frames than the number represented by the Limit set in this register, the endpoint will be halted. Note: A value of 0 or 1 disables the NAK timeout function.

4.68 Transmit Interval Register (Host mode only) (HOST_TXINTERVAL)

The Transmit Interval Register (Host mode only) (HOST_TXINTERVAL) is shown in Figure 83 and described in Table 84.

Figure 83. Transmit Interval Register (Host mode only) (HOST_TXINTERVAL)

7	0
POLINTVL_NAKLIMIT	
R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 84. Transmit Interval Register (Host mode only) (HOST_TXINTERVAL) Field Descriptions

Bit	Field	Value	Description
7-0	POLINTVL_NAKLIMIT	0-FFh	For Interrupt and Isochronous transfers, defines the polling interval for the currently-selected transmit endpoint. For Bulk endpoints, this register sets the number of frames/microframes after which the endpoint should timeout on receiving a stream of NAK responses There is a transmit Interval register for each configured transmit endpoint (except Endpoint 0). In each case the value that is set defines a number of frames/microframes (High Speed transfers), as follows: Transfer Type Speed Valid values (m) Interpretation Interrupt Low Speed or Full Speed 1 - 255 Polling interval is m frames High Speed 1 - 16 Polling interval is $2^{(m-1)}$ microframes Isochronous Full Speed or High Speed 1 - 16 Polling interval is $2^{(m-1)}$ frames/microframes Bulk Full Speed or High Speed 2 - 16 NAK Limit is $2^{(m-1)}$ frames/microframes Note: A value of 0 or 1 disables the NAK timeout function.

4.69 Receive Type Register (Host mode only) (HOST_RXTYPE)

The Receive Type Register (Host mode only) (HOST_RXTYPE) is shown in Figure 84 and described in Table 85.

Figure 84. Receive Type Register (Host mode only) (HOST_RXTYPE)

7	6	5	4	3	0
SPEED		PROT		RENDPN	
R/W-0		R/W-0		R/W-0	

LEGEND: R/W = Read/Write; *-n* = value after reset

Table 85. Receive Type Register (Host mode only) (HOST_RXTYPE) Field Descriptions

Bit	Field	Value	Description
7-6	SPEED	0-3h	Operating Speed of Target Device
		0	Illegal
		1h	High
		2h	Full
		3h	Low
5-4	PROT	0-3h	Set this to select the required protocol for the transmit endpoint
		0	Control
		1h	Isochronous
		2h	Bulk
		3h	Interrupt
3-0	RENDPN	0-Fh	Set this value to the endpoint number contained in the Receive endpoint descriptor returned to the USB controller during device enumeration

4.70 Receive Interval Register (Host mode only) (HOST_RXINTERVAL)

The Receive Interval Register (Host mode only) (HOST_RXINTERVAL) is shown in Figure 85 and described in Table 86.

Figure 85. Receive Interval Register (Host mode only) (HOST_RXINTERVAL)

7	0
POLINTVL_NAKLIMIT	
R/W-0	

LEGEND: R/W = Read/Write; *-n* = value after reset

Table 86. Receive Interval Register (Host mode only) (HOST_RXINTERVAL) Field Descriptions

Bit	Field	Value	Description
7-0	POLINTVL_NAKLIMIT	0-FFh	For Interrupt and Isochronous transfers, defines the polling interval for the currently-selected transmit endpoint For Bulk endpoints, this register sets the number of frames/microframes after which the endpoint should timeout on receiving a stream of NAK responses. There is a transmit Interval register for each configured transmit endpoint (except Endpoint 0). In each case the value that is set defines a number of frames/microframes (High Speed transfers), as follows: Transfer Type Speed Valid values (m) Interpretation Interrupt Low Speed or Full Speed 1 - 255 Polling interval is m frames High Speed 1 - 16 Polling interval is $2^{(-1)}$ microframes Isochronous Full Speed or High Speed 1 - 16 Polling interval is $2^{(-1)}$ frames/microframes Bulk Full Speed or High Speed 2 - 16 NAK Limit is $2^{(-1)}$ frames/microframes Note: A value of 0 or 1 disables the NAK timeout function

4.71 Configuration Data Register (CONFIGDATA)

The configuration data register (CONFIGDATA) is shown in Figure 86 and described in Table 87.

Figure 86. Configuration Data Register (CONFIGDATA)

7	6	5	4	3	2	1	0
MPRXE	MPTXE	BIGENDIAN	HBRXE	HBTXE	DYNFIFO	SOFTCONE	UTMIDATAWIDTH
R-0	R-0	R-0	R-0	R-0	R-1	R-1	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 87. Configuration Data Register (CONFIGDATA) Field Descriptions

Bit	Field	Value	Description
7	MPRXE		Indicates automatic amalgamation of bulk packets.
		0	Automatic amalgamation of bulk packets is not selected.
		1	Automatic amalgamation of bulk packets is selected.
6	MPTXE		Indicates automatic splitting of bulk packets.
		0	Automatic splitting of bulk packets is not selected.
		1	Automatic splitting of bulk packets is selected.
5	BIGENDIAN		Indicates endian ordering.
		0	Little-endian ordering is selected.
		1	Big-endian ordering is selected.
4	HBRXE		Indicates high-bandwidth Rx ISO endpoint support.
		0	High-bandwidth Rx ISO endpoint support is not selected.
		1	High-bandwidth Rx ISO endpoint support is selected.
3	HBTXE		Indicates high-bandwidth Tx ISO endpoint support.
		0	High-bandwidth Tx ISO endpoint support is not selected.
		1	High-bandwidth Tx ISO endpoint support is selected.
2	DYNFIFO		Indicates dynamic FIFO sizing.
		0	Dynamic FIFO sizing option is not selected.
		1	Dynamic FIFO sizing option is selected.
1	SOFTCONE		Indicates soft connect/disconnect.
		0	Soft connect/disconnect option is not selected
		1	Soft connect/disconnect option is selected
0	UTMIDATAWIDTH		Indicates selected UTMI data width.
		0	8 bits
		1	16 bits

4.72 Transmit and Receive FIFO Register for Endpoint 0 (FIFO0)

The Transmit and Receive FIFO Register for Endpoint 0 (FIFO0) is shown in Figure 87 and described in Table 88.

Figure 87. Transmit and Receive FIFO Register for Endpoint 0 (FIFO0)

31	0
DATA	
R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

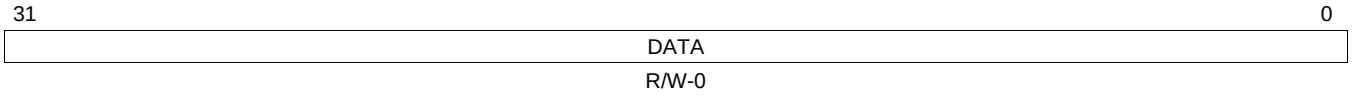
Table 88. Transmit and Receive FIFO Register for Endpoint 0 (FIFO0) Field Descriptions

Bit	Field	Value	Description
31-0	DATA	0-FFFF FFFFh	Writing to these addresses loads data into the Transmit FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Receive FIFO for the corresponding endpoint.

4.73 Transmit and Receive FIFO Register for Endpoint 1 (FIFO1)

The Transmit and Receive FIFO Register for Endpoint 1 (FIFO1) is shown in Figure 88 and described in Table 89.

Figure 88. Transmit and Receive FIFO Register for Endpoint 1 (FIFO1)



LEGEND: R/W = Read/Write; *-n* = value after reset

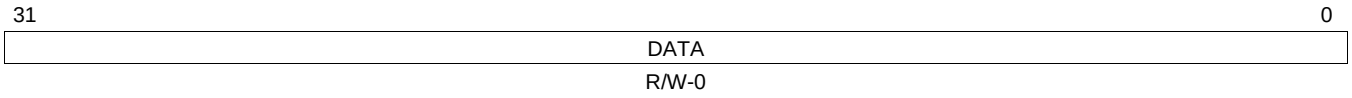
Table 89. Transmit and Receive FIFO Register for Endpoint 1 (FIFO1) Field Descriptions

Bit	Field	Value	Description
31-0	DATA	0-FFFF FFFF	Writing to these addresses loads data into the Transmit FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Receive FIFO for the corresponding endpoint.

4.74 Transmit and Receive FIFO Register for Endpoint 2 (FIFO2)

The Transmit and Receive FIFO Register for Endpoint 2 (FIFO2) is shown in Figure 89 and described in Table 90.

Figure 89. Transmit and Receive FIFO Register for Endpoint 2 (FIFO2)



LEGEND: R/W = Read/Write; *-n* = value after reset

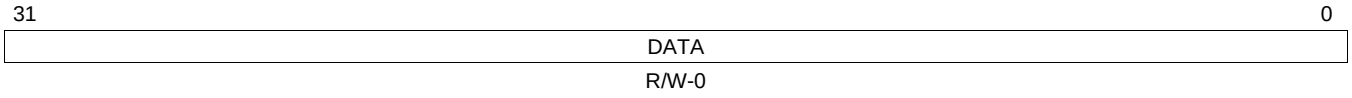
Table 90. Transmit and Receive FIFO Register for Endpoint 2 (FIFO2) Field Descriptions

Bit	Field	Value	Description
31-0	DATA	0-FFFF FFFFh	Writing to these addresses loads data into the Transmit FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Receive FIFO for the corresponding endpoint.

4.75 Transmit and Receive FIFO Register for Endpoint 3 (FIFO3)

The Transmit and Receive FIFO Register for Endpoint 3 (FIFO3) is shown in Figure 90 and described in Table 91.

Figure 90. Transmit and Receive FIFO Register for Endpoint 3 (FIFO3)



LEGEND: R/W = Read/Write; *-n* = value after reset

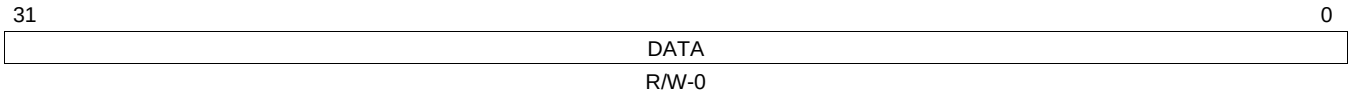
Table 91. Transmit and Receive FIFO Register for Endpoint 3 (FIFO3) Field Descriptions

Bit	Field	Value	Description
31-0	DATA	0-FFFF FFFFh	Writing to these addresses loads data into the Transmit FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Receive FIFO for the corresponding endpoint.

4.76 Transmit and Receive FIFO Register for Endpoint 4 (FIFO4)

The Transmit and Receive FIFO Register for Endpoint 4 (FIFO4) is shown in Figure 91 and described in Table 92.

Figure 91. Transmit and Receive FIFO Register for Endpoint 4 (FIFO4)



LEGEND: R/W = Read/Write; *-n* = value after reset

Table 92. Transmit and Receive FIFO Register for Endpoint 4 (FIFO4) Field Descriptions

Bit	Field	Value	Description
31-0	DATA	0-FFFF FFFFh	Writing to these addresses loads data into the Transmit FIFO for the corresponding endpoint. Reading from these addresses unloads data from the Receive FIFO for the corresponding endpoint.

4.77 OTG Device Control Register (DEVCTL)

The OTG Device Control Register (DEVCTL) is shown in Figure 92 and described in Table 93.

Figure 92. OTG Device Control Register (DEVCTL)

7	6	5	4	3	2	1	0
BDEVICE	FSDEV	LSDEV	VBUS		HOSTMODE	HOSTREQ	SESSION
R-0	R-0	R-0	R-0		R-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; *-n* = value after reset

Table 93. OTG Device Control Register (DEVCTL) Field Descriptions

Bit	Field	Value	Description
7	BDEVICE		This read-only bit indicates whether the USB controller is operating as the 'A' device or the 'B' device.
		0	'A' device
		1	'B' device
			Only valid while a session is in progress.
6	FSDEV	0-1	This read-only bit is set when a full-speed or high-speed device has been detected being connected to the port (high-speed devices are distinguished from full-speed by checking for high-speed chirps when the device is reset). Only valid in Host mode.
5	LSDEV	0-1	This read-only bit is set when a low-speed device has been detected being connected to the port. Only valid in Host mode.
4-3	VBUS	0-3h	These read-only bits encode the current VBus level as follows:
		0	Below Session End
		1h	Above Session End, below AValid
		2h	Above AValid, below VBusValid
		3h	Above VBusValid
2	HOSTMODE	0-1	This read-only bit is set when the USB controller is acting as a Host.
1	HOSTREQ	0-1	When set, the USB controller will initiate the Host Negotiation when Suspend mode is entered. It is cleared when Host Negotiation is completed. ('B' device only)
0	SESSION	0-1	When operating as an 'A' device, you must set or clear this bit start or end a session. When operating as a 'B' device, this bit is set/cleared by the USB controller when a session starts/ends. You must also set this bit to initiate the Session Request Protocol. When the USB controller is in Suspend mode, you may clear the bit to perform a software disconnect. A special software routine is required to perform SRP. Details will be made available in a later document version.

4.78 Transmit Endpoint FIFO Size (TXFIFOSZ)

Section 2.5 describes dynamically setting endpoint FIFO sizes.

The Transmit Endpoint FIFO Size (TXFIFOSZ) is shown in Figure 93 and described in Table 94.

Figure 93. Transmit Endpoint FIFO Size (TXFIFOSZ)

7	5	4	3	0
Reserved		DPB	SZ	
R-0		R/W-0	R-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 94. Transmit Endpoint FIFO Size (TXFIFOSZ) Field Descriptions

Bit	Field	Value	Description
7-5	Reserved	0	Reserved
4	DPB		Double packet buffering enable
		0	Single packet buffering is supported
		1	Double packet buffering is enabled
3-0	SZ	0-Fh	Maximum packet size to be allowed (before any splitting within the FIFO of Bulk packets prior to transmission). If m = SZ[3:0], the FIFO size is calculated as $2^{(m+3)}$ for single packet buffering and $2^{(m+4)}$ for dual packet buffering.

4.79 Receive Endpoint FIFO Size (RXFIFOSZ)

Section 2.5 describes dynamically setting endpoint FIFO sizes.

The Receive Endpoint FIFO Size (RXFIFOSZ) is shown in Figure 94 and described in Table 95.

Figure 94. Receive Endpoint FIFO Size (RXFIFOSZ)

7	5	4	3	0
Reserved		DPB	SZ	
R-0		R/W-0	R-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 95. Receive Endpoint FIFO Size (RXFIFOSZ) Field Descriptions

Bit	Field	Value	Description
7-5	Reserved	0	Reserved
4	DPB		Double packet buffering enable
		0	Single packet buffering is supported
		1	Double packet buffering is enabled
3-0	SZ	0-Fh	Maximum packet size to be allowed (before any splitting within the FIFO of Bulk packets prior to transmission). If m = SZ[3:0], the FIFO size is calculated as $2^{(m+3)}$ for single packet buffering and $2^{(m+4)}$ for dual packet buffering.

4.80 Transmit Endpoint FIFO Address (TXFIFOADDR)

Section 2.5 describes dynamically setting endpoint FIFO sizes.

The Transmit Endpoint FIFO Address (TXFIFOADDR) is shown in Figure 95 and described in Table 96.

Figure 95. Transmit Endpoint FIFO Address (TXFIFOADDR)

15	13	12	0
Reserved		ADDR	
R-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 96. Transmit Endpoint FIFO Address (TXFIFOADDR) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved
12-0	ADDR	0-1FFFh	Start Address of endpoint FIFO in units of 8 bytes If m = ADDR[12:0] then the start address is 8*m

4.81 Receive Endpoint FIFO Address (RXFIFOADDR)

Section 2.5 describes dynamically setting endpoint FIFO sizes.

The Receive Endpoint FIFO Address (RXFIFOADDR) is shown in Figure 96 and described in Table 97.

Figure 96. Receive Endpoint FIFO Address (RXFIFOADDR)

15	13	12	0
Reserved		ADDR	
R-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 97. Receive Endpoint FIFO Address (RXFIFOADDR) Field Descriptions

Bit	Field	Value	Description
15-13	Reserved	0	Reserved
12-0	ADDR	0-1FFFh	Start Address of endpoint FIFO in units of 8 bytes If m = ADDR[12:0], then the start address is 8*m

4.82 Transmit Function Address (TXFUNCADDR)

The Transmit Function Address (TXFUNCADDR) is shown in Figure 97 and described in Table 98.

Figure 97. Transmit Function Address (TXFUNCADDR)

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

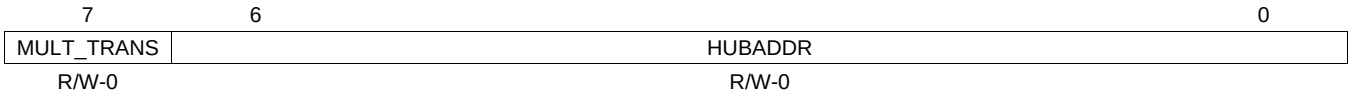
Table 98. Transmit Function Address (TXFUNCADDR) Field Descriptions

Bit	Field	Value	Description
7	Reserved	0	Reserved
6-0	FUNCADDR	0-7Fh	Address of target function

4.83 Transmit Hub Address (TXHUBADDR)

The Transmit Hub Address (TXHUBADDR) is shown in Figure 98 and described in Table 99.

Figure 98. Transmit Hub Address (TXHUBADDR)



LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

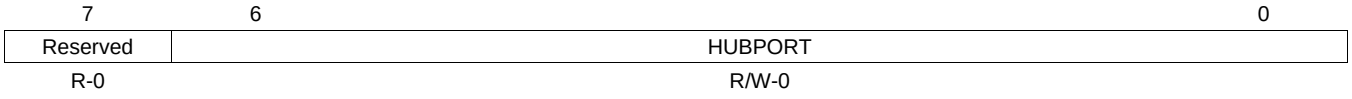
Table 99. Transmit Hub Address (TXHUBADDR) Field Descriptions

Bit	Field	Value	Description
7	MULT_TRANS	0-1	Set to 1 if hub has multiple transaction translators. Cleared to 0 if only single transaction translator is available.
6-0	HUBADDR	0-7Fh	Address of hub

4.84 Transmit Hub Port (TXHUBPORT)

The Transmit Hub Port (TXHUBPORT) is shown in Figure 99 and described in Table 100.

Figure 99. Transmit Hub Port (TXHUBPORT)



LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 100. Transmit Hub Port (TXHUBPORT) Field Descriptions

Bit	Field	Value	Description
7	Reserved	0	Reserved
6-0	HUBPORT	0-7Fh	Port number of the hub

4.85 Receive Function Address (RXFUNCADDR)

The Receive Function Address (RXFUNCADDR) is shown in Figure 100 and described in Table 101.

Figure 100. Receive Function Address (RXFUNCADDR)

7	6	0
Reserved	FUNCADDR	
R-0	R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 101. Receive Function Address (RXFUNCADDR) Field Descriptions

Bit	Field	Value	Description
7	Reserved	0	Reserved
6-0	FUNCADDR	0-7Fh	Address of target function

4.86 Receive Hub Address (RXHUBADDR)

The Receive Hub Address (RXHUBADDR) is shown in Figure 101 and described in Table 102.

Figure 101. Receive Hub Address (RXHUBADDR)

7	6	0
MULT_TRANS	HUBADDR	
R/W-0	R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 102. Receive Hub Address (RXHUBADDR) Field Descriptions

Bit	Field	Value	Description
7	MULT_TRANS	0-1	Set to 1 if hub has multiple transaction translators. Cleared to 0 if only single transaction translator is available.
6-0	HUBADDR	0-7Fh	Address of hub

4.87 Receive Hub Port (RXHUBPORT)

The Receive Hub Port (RXHUBPORT) is shown in Figure 102 and described in Table 103.

Figure 102. Receive Hub Port (RXHUBPORT)

7	6	0
Reserved	HUBPORT	
R-0	R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 103. Receive Hub Port (RXHUBPORT) Field Descriptions

Bit	Field	Value	Description
7	Reserved	0	Reserved
6-0	HUBPORT	0-7Fh	Port number of hub

Appendix A Revision History

Table 104 lists the changes made since the previous version of this document.

Table 104. Document Revision History

Reference	Additions/Modifications/Deletions
Section 2.4	Changed subsection.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DLP® Products	www.dlp.com	Communications and Telecom	www.ti.com/communications
DSP	dsp.ti.com	Computers and Peripherals	www.ti.com/computers
Clocks and Timers	www.ti.com/clocks	Consumer Electronics	www.ti.com/consumer-apps
Interface	interface.ti.com	Energy	www.ti.com/energy
Logic	logic.ti.com	Industrial	www.ti.com/industrial
Power Mgmt	power.ti.com	Medical	www.ti.com/medical
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Space, Avionics & Defense	www.ti.com/space-avionics-defense
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video and Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless-apps

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265