TEXAS INSTRUMENTS

TUSB3410

USB To Serial Port Controller

Data Manual

June 2002

MSP USB

SLLS519B

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 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.

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.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Contents

List of Illustrations

List of Tables

1 Introduction

1.1 Controller Description

The TUSB3410 provides bridging between a USB port and an enhanced UART serial port. The TUSB3410 contains all the necessary logic to communicate with the host computer using the USB bus. It contains an 8052 microcontroller unit (MCU) with 16K bytes of RAM that can be loaded from the host or from external on-board memory via an I^2C bus. It also contains 10K bytes of ROM that allow the MCU to configure the USB port at boot time. The ROM code also contains an I^2C boot loader. All the device functions such as the USB command decoding, UART setup, and error reporting are managed by the internal MCU firmware under the auspices of the PC host.

The TUSB3410 can be used to build an interface between a legacy serial peripheral device and a PC with USB ports, such as a legacy-free PC. Once configured, data flows from the host to the TUSB3410 via USB OUT commands and then out from the TUSB3410 on the SOUT line. Conversely, data flows into the TUSB3410 on the SIN line and then into the host via USB IN commands.

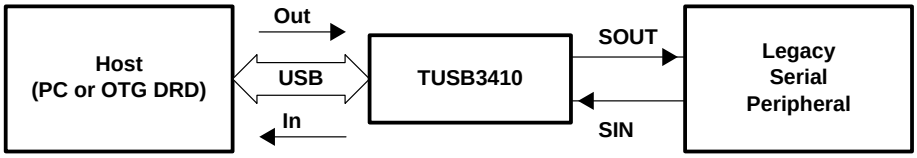


Figure 1–1. Data Flow

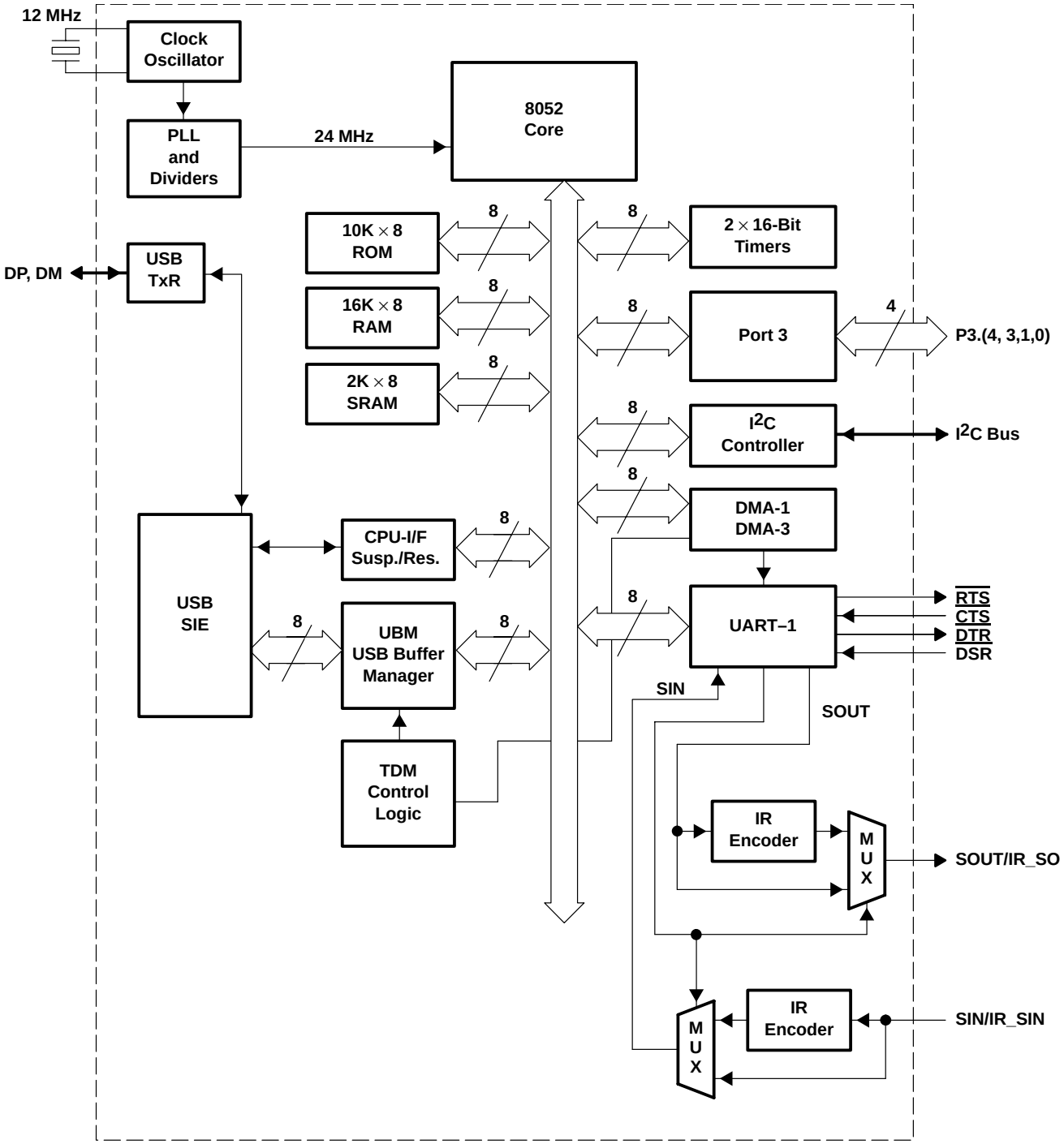


Figure 1–2. USB-to-Serial (Single Channel) Controller Block Diagram

2 Main Features

2.1 USB Features

- Fully compliant with USB 2.0 full speed Specifications
- Supports 12-Mbps USB data rate (full speed)
- Supports USB suspend, resume, and remote wakeup operations
- Supports two power source modes:
  - Bus-powered mode
  - Self-powered mode
- Can support a total of 3-input and 3-output (interrupt, bulk) endpoints

2.2 General Features

- Integrated 8052 microcontroller with
  - 256 × 8 RAM for internal data
  - 10K × 8 ROM (with USB and I^2C boot loader)
  - 16K × 8 RAM for code space loadable from host or I^2C port
  - 2K × 8 Shared RAM used for data buffers and endpoint descriptor blocks (EDB)
  - Four GPIO pins from 8052 port 3
  - Master I^2C controller for EEPROM device access
  - MCU operates at 24 MHz providing 2 MIPS operation
  - 128-ms Watchdog Timer
- Built-in two-channel DMA controller for USB/UART bulk I/O
- Operates from a 12-MHz crystal
- Supports USB suspend and resume
- Supports remote wake-up
- Available in 32-pin LQFP
- 3.3-V operation with 1.8-V core operating voltage provided by on-chip 1.8-V voltage regulator

2.3 Enhanced UART Features

- Software/hardware flow control:
  - Programmable Xon/Xoff characters
  - Programmable Auto-$\overline{\text{RTS}}$/$\overline{\text{DTR}}$ and Auto-$\overline{\text{CTS}}$/$\overline{\text{DSR}}$
- Automatic RS485-bus transceiver control, with and without echo
- Selectable IrDA mode for up to 115.2 kbps transfer
- Software selectable baud rate from 50 to 921.6 k baud
- Programmable serial-interface characteristics
  - 5-, 6-, 7-, or 8-Bit characters
  - Even, odd, or no parity-bit generation and detection
  - 1-, 1.5-, or 2-Stop bit generation
- Line break generation and detection

- Internal test and loop-back capabilities
- Modem-control functions ($\overline{\text{CTS}}$, $\overline{\text{RTS}}$, $\overline{\text{DSR}}$, $\overline{\text{DTR}}$, $\overline{\text{RI}}$, and $\overline{\text{DCD}}$)
- Internal diagnostics capability
  - Loopback control for communications link-fault isolation
  - Break, parity, overrun, framing-error simulation

2.4 Pinout Information

VF PACKAGE
(TOP VIEW)

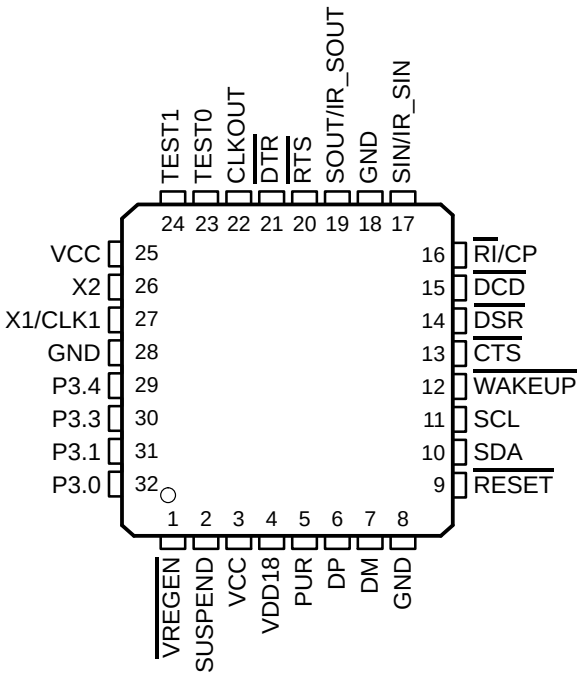


Table 2–1. Terminal Functions

TERMINAL NAME	TERMINAL NO.	I/O	DESCRIPTION
CLKOUT	22	O	Clock output (controlled by CLKOUTEN and CLKSLCT in MODECNFG register (see Section 5.1.5 and Note 1)
$\overline{\text{CTS}}$	13	I	UART: Clear to send (see Note 4)
$\overline{\text{DCD}}$	15	I	UART: Data carrier detect (see Note 4)
DM	7	I/O	Upstream USB port differential data minus
DP	6	I/O	Upstream USB port differential data plus
$\overline{\text{DSR}}$	14	I	UART: Data set ready (see Note 4)
$\overline{\text{DTR}}$	21	O	UART: Data terminal ready (see Note 1)
GND	8, 18, 28	GND	Digital ground
P3.0	32	I/O	Port-3.0 (see Notes 3, 4, 5, and 8)
P3.1	31	I/O	Port-3.1 (see Notes 3, 4, 5, and 8)
P3.3	30	I/O	Port-3.3 (see Notes 3, 4, 5, and 8)
P3.4	29	I/O	Port-3.4 (see Notes 3, 4, 5, and 8)
PUR	5	O	Pull-up resistor connection (see Note 2)
$\overline{\text{RESET}}$	9	I	Controller master reset signal (see Note 4)
$\overline{\text{RI}}$/CP	16	I	UART: Ring indicator (see Note 4)
$\overline{\text{RTS}}$	20	O	UART: Request to send (see Note 1)
SCL	11	O	Master I^2C controller: clock signal (see Note 1)
SDA	10	I/O	Master I^2C controller: data signal (see Notes 1 and 5)
SIN/IR_SIN	17	I	UART: Serial input data / IR Serial data input (see Note 6)
SOUT/IR_SOUT	19	O	UART: Serial output data / IR Serial data output (see Note 7)
SUSPEND	2	O	Suspend condition signal (see Note 3)
TEST0	23	I	Test input (for factory test only) (see Note 5)
TEST1	24	I	Test input (for factory test only) (see Note 5)
VCC	3, 25	PWR	3.3 V
VDD18	4	PWR	1.8-V Supply. An internal voltage regulator generates this supply voltage when terminal $\overline{\text{VREGEN}}$ is asserted. When $\overline{\text{VREGEN}}$ is deasserted, 1.8 V must be supplied externally.
$\overline{\text{VREGEN}}$	1	I	This active-low terminal is used to enable the 3.3-V to 1.8-V voltage regulator in the core.
$\overline{\text{WAKEUP}}$	12	I	Remote wake-up request pin. When low, wakes up system (see Note 5)
X1/CLKI	27	I	12-MHz crystal input or clock input
X2	26	O	12-MHz crystal output

NOTES:
1. 3-state CMOS output (±4-mA drive/sink)
2. 3-state CMOS output (±8-mA drive/sink)
3. 3-state CMOS output (±12-mA drive/sink)
4. TTL-compatible, hysteresis input
5. TTL-compatible, hysteresis input, with internal 100-µA active pullup
6. TTL-compatible input without hysteresis, with internal 100-µA active pullup
7. Normal or IR mode: 3-state CMOS output (±4-mA drive/sink)
8. The MCU treats the outputs as open drain types in that the output can be driven low continuously, but a high output is driven for two clock cycles and then the output is tristated.

3 Detailed Controller Description

3.1 Operating Modes

The TUSB3410 controls its USB interface in response to USB commands, and this action remains the same independent of the serial port mode selected. On the other hand, the serial port can be set up in three modes.

As with any interface device, data movement is the TUSB3410's main function, but typically the initial configuration and error handling consume most of the support code. The following sections describe the various modes the device can be used in and the means of setting up the device.

3.2 USB Interface Configuration

The TUSB3410 contains onboard ROM microcode, which enables the MCU to enumerate the device as a USB peripheral. The ROM microcode can also load application code into internal RAM from either external memory via the I^2C bus or from the host via the USB.

3.2.1 External Memory Case

After reset, the TUSB3410 is disconnected from the USB because the pullup resistor CONT bit is cleared. The TUSB3410 checks the I^2C port for the existence of valid code, if it finds valid code, it uploads the code from the external memory device into the RAM program space. Once loaded, the TUSB3410 connects to the USB by setting the CONT bit and enumeration and configuration are performed. This is the most likely use of the device.

3.2.2 Host Download Case

If the valid code is not found at the I^2C port, the TUSB3410 connects to the USB by setting the CONT bit, and then an enumeration and default configuration are performed. The host can then download additional microcode into RAM to tailor the application. Then, the MCU causes a disconnect and reconnect by using the pullup resistor CONT bit in the USBCTL register, which causes the TUSB3410 to be re-enumerated with a new configuration.

3.3 USB Data Movement

From the USB perspective, the TUSB3410 looks like a USB peripheral device. It uses endpoint 0 as its control endpoint, as do all USB peripherals. It also configures up to three input and three output endpoints, although most applications use one bulk input endpoint for data in, one bulk output endpoint for data out, and one interrupt endpoint for status updates. The USB configuration likely remains the same regardless of the serial port configuration.

Most data is moved from the USB side to the UART side and vice versa using on-chip DMA transfers. Some special cases may use programmed IO under control of the MCU.

3.4 Serial Port Setup

The serial port requires a few control registers to be written to configure its operation. This configuration likely remains the same regardless of the data mode used. These registers include the line control register that controls the serial word format and the divisor registers that control the baud rate.

These registers are usually controlled by the host application.

3.5 Serial Port Data Modes

The serial port can be configured in three different, although similar, data modes. Similar to the USB mode, once configured for a specific application, it is unlikely that the mode would be changed. The different modes affect the timing of the serial input and output or the use of the control signals. However, the basic serial-to-parallel conversion

of the receiver and parallel-to-serial conversion of the transmitter remain the same in all modes. Some features are available in all modes, but are only applicable in certain modes. For instance, software flow control via Xoff/Xon characters can be used in all modes, but would usually only be used in RS-232 or IrDA mode because the RS-485 mode is half-duplex communication. Similarly, hardware flow control via $\overline{\text{RTS}}$/$\overline{\text{CTS}}$ (or $\overline{\text{DTR}}$/$\overline{\text{DSR}}$) handshaking is available in RS-232 or IrDA mode. However, this would probably be used only in RS-232 mode, since in IrDA mode only the SIN and SOUT paths are optically coupled.

3.5.1 RS-232 Data Mode

The default mode is called the RS-232 mode, and is usually used for full duplex communication on SOUT and SIN. In this mode, the modem control outputs ($\overline{\text{RTS}}$ and $\overline{\text{DTR}}$) are used to communicate to a modem or as general outputs. The modem control inputs ($\overline{\text{CTS}}$, $\overline{\text{DSR}}$, $\overline{\text{DCD}}$, and $\overline{\text{RI}}$) are used for modem communication or as general inputs. Alternatively, $\overline{\text{RTS}}$ and $\overline{\text{CTS}}$ (or $\overline{\text{DTR}}$ and $\overline{\text{DSR}}$) can be used to throttle the data flow on SOUT and SIN to prevent receive fifo overruns. Finally, software flow control via Xoff/Xon characters can be used for the same purpose.

This mode represents the most general-purpose applications, and the other modes are subsets of this mode.

3.5.2 RS-485 Data Mode

The RS-485 mode is very similar to the RS-232 mode in that the SOUT and SIN formats remain the same. Since RS-485 is a bus architecture, it is inherently a single duplex communication system. The TUSB3410 in RS-485 mode controls the $\overline{\text{RTS}}$ and $\overline{\text{DTR}}$ signals such that either can be used to enable an RS-485 driver or RS-485 receiver. When in RS-485 mode, the enable signals for transmitting are automatically asserted whenever the DMA is set up for outbound data. The receiver can be left enabled while the driver is enabled to allow an echo if desired, but when receive data is expected, the driver must be disabled. Note that this precludes use of hardware flow control, since this is a half duplex operation, it would not be effective anyhow. Software flow control is supported, but may be of limited value.

The RS-485 mode is enabled by setting the 485E bit in the FCRL register, and a receiver enable (RCVE) bit in the MCR allows the receiver to eavesdrop while in 485 mode.

3.5.3 IrDA Data Mode

The IrDA mode encodes SOUT and decodes SIN in the manner prescribed by the IrDA standard, up to 115.2 kbps. Connection to an external IrDA transceiver is required. Communications is usually full duplex. Generally in an IrDA system only the SOUT and SIN paths are connected, so hardware flow control is usually not an option. Software flow control is supported.

The IrDA mode is enabled by setting the IREN bit in the USB control register.

The IR encoder and decoder circuitry work with the UART to change the serial bit stream into a series of pulses and back again. For every zero bit in the outbound serial stream, the encoder sends a low-to-high-to-low pulse with the duration of 3/16 of a bit frame at the middle of the bit time. For every one bit in the serial stream, the output remains low for the entire bit time.

The decoding process consists of receiving the signal from the IrDA receiver and converting it to a series of zeroes and ones. As the converse to the encoder, the decoder converts a pulse to a zero bit and the lack of a pulse to a one bit.

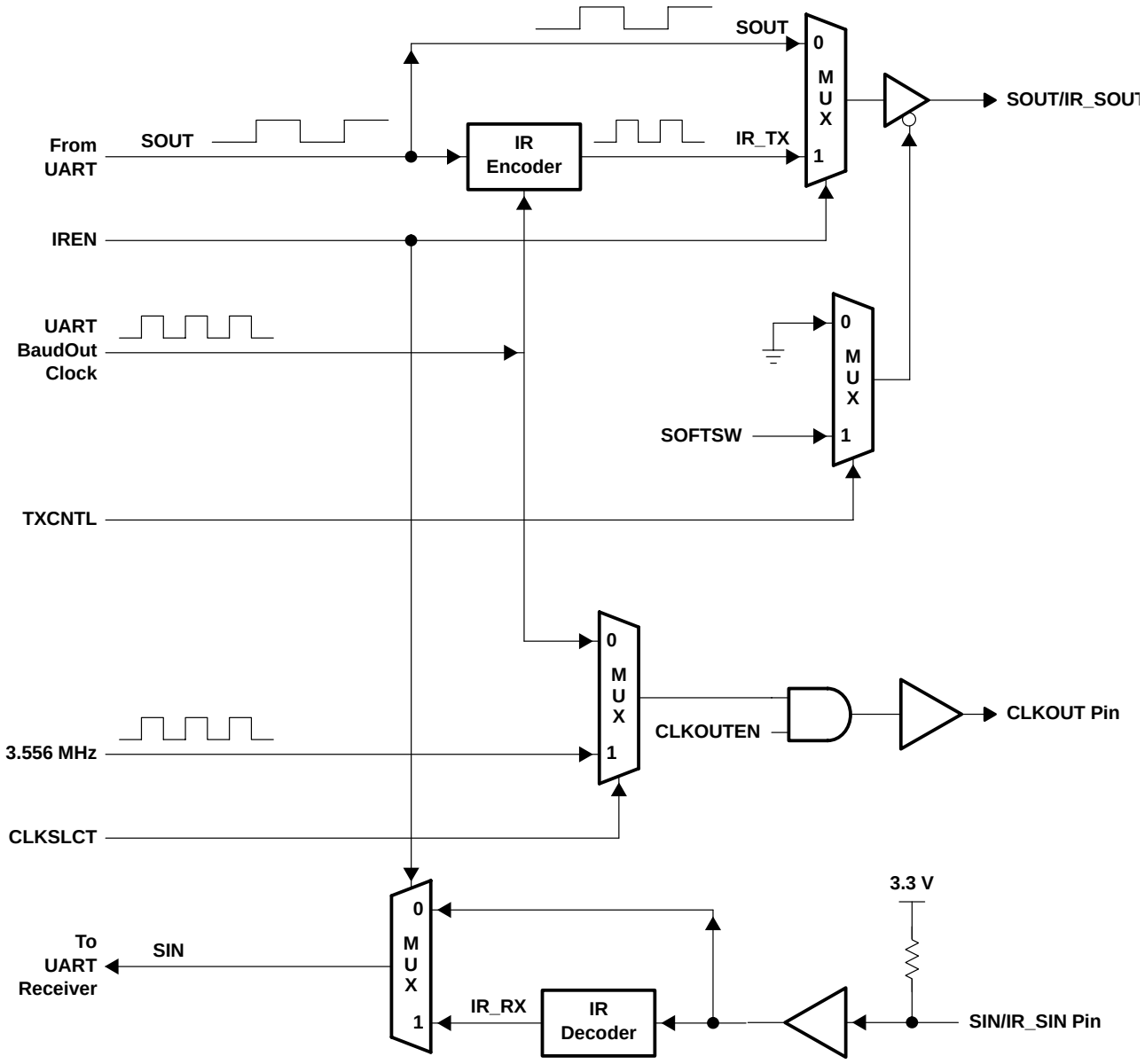


Figure 3–1. RS-232 and IR Mode Select

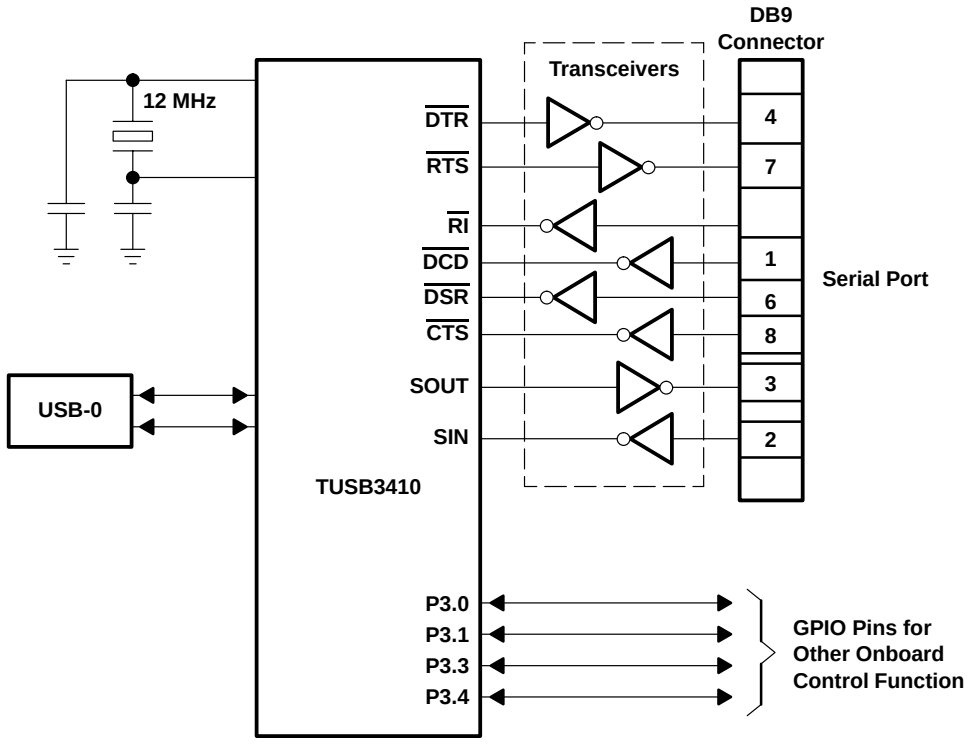


Figure 3–2. USB-to-Serial Implementation (RS-232)

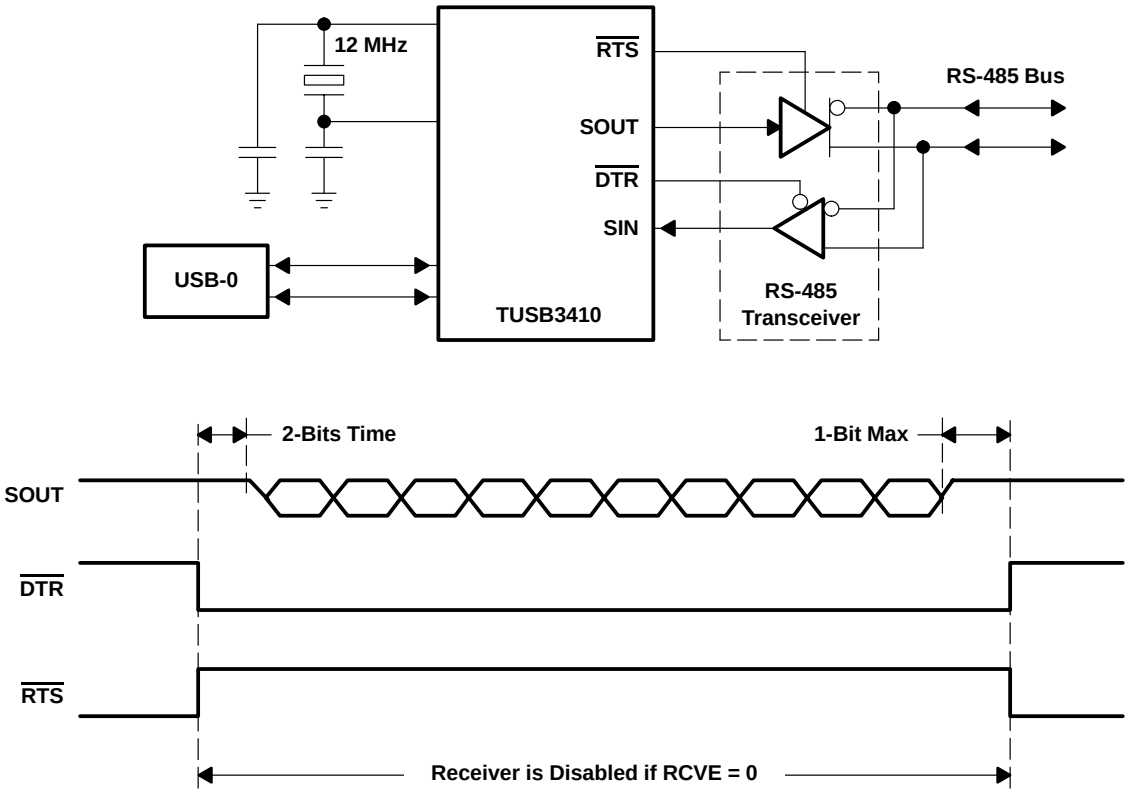


Figure 3–3. RS-485 Bus Implementation

4 MCU Memory Map (Internal Operation)

Figure 4–1 illustrates the MCU memory map under boot and normal operation. For more information regarding the integrated 8052, see the *TUSBxxxx Microcontroller Reference Guide* (SLLU044).

NOTE:

The internal 256 bytes of RAM are not shown, since they are assumed to be in the standard 8052 location (0000 to 00FF). The shaded areas represent the internal ROM/RAM.

When SDW bit = 0 (boot mode): The 10K ROM is mapped to address (0x0000–0x27FF) and is duplicated in location (0x8000–0xA7FF) in code space. The internal 16K RAM is mapped to address range (0x0000–0x3FFF) in data space. Buffers, MMR, and I/O are mapped to address range (0xF800–0xFFFF) in data space.

When SDW bit = 1 (normal mode): The 10K ROM is mapped to (0x8000–0xA7FF) in code space. The internal 166K RAM is mapped to address range (0x0000–0x3FFFF) in code space. Buffers, MMR, and I/O are mapped to address range (0xF800–0xFFFF) in data space.

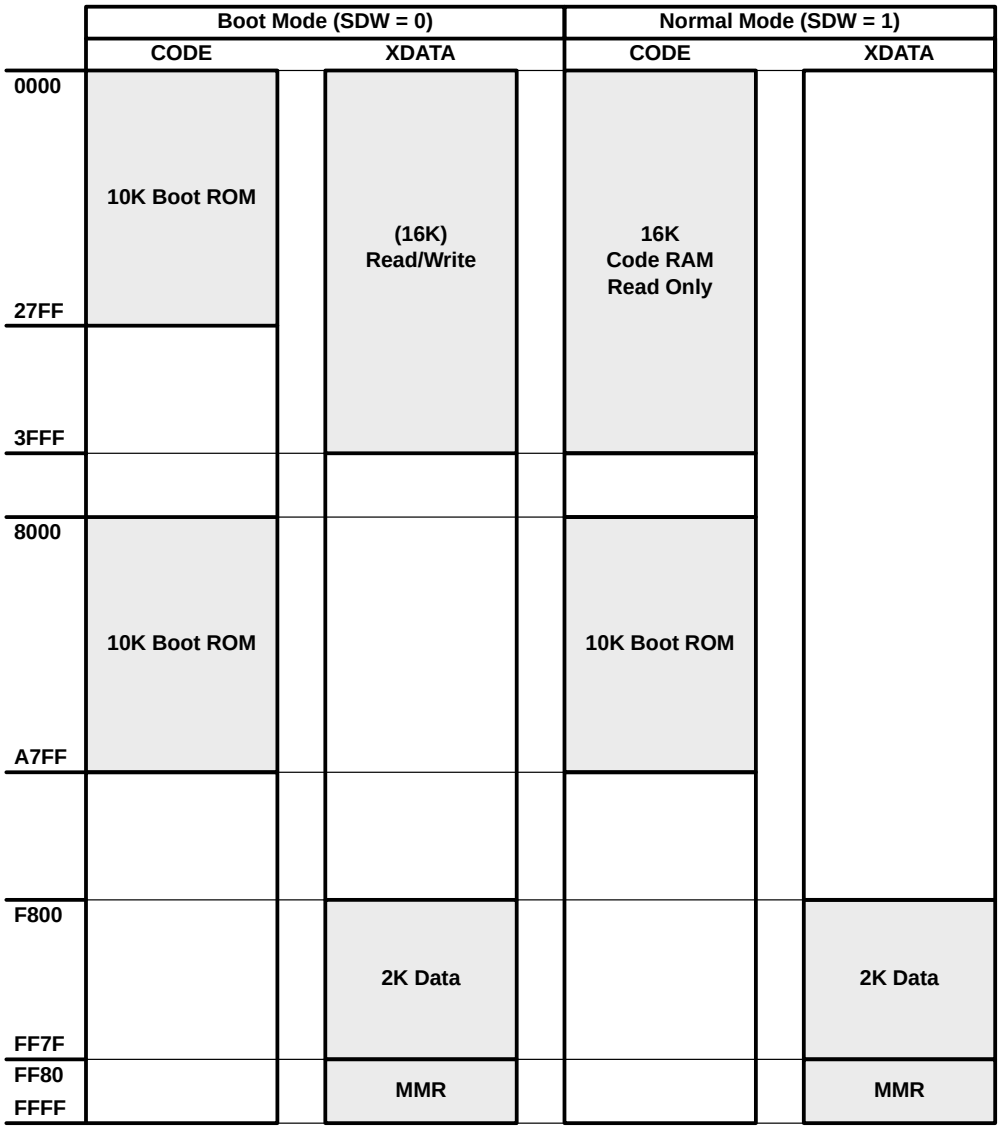


Figure 4–1. MCU Memory Map

4.1 Miscellaneous Registers

4.1.1 ROMS: ROM Shadow Configuration Register (Addr:FF90)

This register is used by the MCU to switch from boot mode to normal operation mode (boot mode is set on power-on reset only). In addition, this register provides the device revision number and the ROM/RAM configuration.

7	6	5	4	3	2	1	0
ROA	**S1**	**S0**	**R3**	**R2**	**R1**	**R0**	**SDW**
R/O	R/O	R/O	R/O	R/O	R/O	R/O	R/W

BIT	NAME	RESET	FUNCTION
0	SDW	0	This bit enables/disables boot ROM. (Shadow the ROM). SDW = 0 When clear, the MCU executes from the 10K boot-ROM space. The boot ROM appears in two locations: 0000 and 8000h. The 16K RAM is mapped to XDATA space; therefore, read/write operation is possible. This bit is set by the MCU after the RAM load is completed. MCU cannot clear this bit; it is cleared on power-up reset or watchdog time-out reset. SDW = 1 When set by the MCU, the 10K boot-ROM maps to location 8000h, and the 16K RAM is mapped to code space, starting at location 0000h. At this point, the MCU executes from RAM, and the write operation is disabled (no write operation is possible in code space).
4–1	R[3:0]	No effect	These bits reflect the device revision number.
6–5	S[1:0]	No effect	Code space size. These bits define the ROM or RAM code-space size (ROA bit defines ROM or RAM). These bits are permanently set and are not affected by reset (see Table 4–1). 00 = 4K bytes code space size 01 = 8K bytes code space size 10 = 16K bytes code space size 11 = 32K bytes code space size
7	ROA	No effect	ROM or RAM version. This bit indicates whether the code space is RAM or ROM based. This bit is permanently set and is not affected by reset (see Table 4–1). ROA = 0 Code space is ROM ROA = 1 Code space is RAM

Table 4–1. ROM/RAM Size Definition Table

ROMS REGISTER			BOOT ROM	RAM CODE	ROM CODE
ROA	S1	S0			
0	0	0	None	None	4K
0	0	1	None	None	8K
0	1	0	None	None	16K (reserved)
1	1	1	None	None	32K (reserved)
1	0	0	10K	4K	None
1	0	1	10K	8K	None
1	1	0	10K	16K	None
1	1	1	10K	32K (reserved)	None

4.1.2 Boot Operation (MCU Firmware Loading)

Since the code space is in RAM (with the exception of the boot ROM), the TUSB3410 firmware must be loaded from an external source. Two sources are available for booting: one from an external serial E^2PROM connected to the I^2C bus and the other from the host via the USB. On device reset, the SDW bit (in ROMS register) and CONT bit (in USBCTL: USB control register) are cleared. This configures the memory space to boot mode (see Memory Map) and keeps the device disconnected from the host. The first instruction is fetched from location 0000h (which is in the 10K ROM). The 16K RAM is mapped to XDATA space (location 0000h). The MCU executes a read from an external E^2PROM and tests whether it contains the code (by testing for boot signature). If it contains the code, the MCU reads from E^2PROM and writes to the 16K RAM in XDATA space. If it does not contain the code, the MCU proceeds to boot from the USB.

Once the code is loaded, the MCU sets SDW = 1. This switches the memory map to normal mode; i.e. the 16K RAM is mapped to code space, and the MCU starts executing from location 0000h. Once the switch is done, the MCU sets CONT = 1 (in the USBCTL register). This connects the device to the USB and results in normal USB device enumeration.

4.1.3 WDCSR: Watchdog Timer, Control, and Status Register (Addr:FF93)

A watchdog timer (WDT) with 1-ms clock is provided. If this register is not accessed for a period of 128 ms, the WDT counter resets the MCU. (see Figure 5–1). The watchdog timer is enabled by default and can be disabled by writing a pattern of 101010 into the WDD[5:0] bits.

7	6	5	4	3	2	1	0
ROA	**S1**	**S0**	**R3**	**R2**	**R1**	**R0**	**SDW**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	W/O

BIT	NAME	RESET	FUNCTION
0	WDT	0	MCU must write a 1 to this bit to prevent the WDT from resetting the MCU. If MCU does not write a 1 in a period of 128 ms, the WDT resets the device. Writing a 0 has no effect on the WDT. (WDT is a 7-bit counter using a 1-ms CLK). This bit is read as 0.
5–1	WDD[5:1]	00000	These bits are used to disable the watchdog timer. For the timer to be disabled these bits must be set to 10101 and WDD[0] must also be set to 0. If any other pattern is present, the watchdog timer is in operation.
6	WDR	0	Watchdog reset indication bit. This bit indicates if the reset occurred due to power-on reset or watchdog timer reset.
			WDR = 0 A power-up reset occurred
			WDR = 1 A USB reset or watchdog time-out reset occurred. To clear this bit, the MCU must write a 1. Writing a 0 has no effect.
7	WDD[0]	1	This bit is one of the disable bits for the watchdog timer. This bit must be cleared in order for the watchdog timer to be disabled.

4.2 Buffers + I/O RAM Map

The address range from F800 to FFFF (2K bytes) is reserved for data buffers, setup packet, endpoint descriptors block (EDB), and all I/O. There are 128 locations reserved for MMR (memory mapped registers). Table 4–2 represents the XDATA space allocation and access restriction for the DMA, UBM, and MCU.

Table 4–2. XDATA Space

DESCRIPTION	ADDRESS RANGE	UBM ACCESS	DMA ACCESS	MCU ACCESS
Internal MMRs (Memory Mapped Registers)	FFFF ↑ FF80	No (Only EDB-0)	No (only Data reg. and EDB-0)	Yes
EDB (Endpoint Descriptors Block)	FF7F ↑ FF08	Only for EDB update	Only for EDB update	Yes
Setup Packet	FF07 ↑ FF00	Yes	No	Yes
Input Endpoint-0 Buffer	FEFF ↑ FEF8	Yes	Yes	Yes
Output Endpoint-0 Buffer	FEF7 ↑ FEF0	Yes	Yes	Yes
Data Buffers	FEEF ↑ F800	Yes	Yes	Yes

Table 4–3. Memory Mapped Registers Summary (XDATA Range = FF80 → FFFF)

ADDRESS	REGISTER	DESCRIPTION
FFFF	FUNADR	Function address register
FFFE	USBSTA	USB status register
FFFD	USBMSK	USB interrupt mask register
FFFC	USBCTL	USB control register
FFFB	MODECNFG	Mode configuration register
FFFA	DEVVIDH	Device custom VID high byte register
FFF9	DEVVIDL	Device custom VID low byte register
FFF8	DEVPIDH	Device custom PID high byte register
FFF7	DEVPIDL	Device custom PID low byte register
FFF6	DEVREVH	Device custom revision number high byte register
FFF5	DEVREVL	Device custom revision number low byte register
↑	RESERVED	
FFF3	I2CADR	I^2C-port address register
FFF2	I2CDATI	I^2C-port data input register
FFF1	I2CDATO	I^2C-port data output register
FFF0	I2CSTA	I^2C-port status register
FFEF	SERNUM7	Serial number byte 7 register
FFEE	SERNUM6	Serial number byte 6 register
FFED	SERNUM5	Serial umber byte 5 register
FFEC	SERNUM4	Serial number byte 4 register
FFEB	SERNUM3	Serial number byte 3 register
FFEA	SERNUM2	Serial number byte 2 register
FFE9	SERNUM1	Serial number byte 1 register
FFE8	SERNUM0	Serial number byte 0 register
↑	RESERVED	
FFE5	DMACSR3	DMA–3: Control and status register
FFE4	DMACDR3	DMA–3: Channel definition register
↑	RESERVED	
FFE1	DMACSR1	DMA–1: Control and status register
FFE0	DMACDR1	DMA–1: Channel definition register
↑	RESERVED	
FFAB	MASK	UART: Interrupt mask register
FFAA	XOFF	UART: Xoff register
FFA9	XON	UART: Xon register
FFA8	DLH	UART: Divisor high-byte register
FFA7	DLL	UART: Divisor low-byte register
FFA6	MSR	UART: Modem status register
FFA5	LSR	UART: Line status register
FFA4	MCR	UART: Modem control register
FFA3	FCRL	UART: Flow control register
FFA2	LCR	UART: Line control registers
FFA1	TDR	UART: Transmitter data registers
FFA0	RDR	UART: Receiver data registers
FF9E	PUR_3	GPIO: Pullup register for port 3

Table 4–3. Memory Mapped Registers Summary (XDATA Range = FF80 → FFFF) (Continued)

ADDRESS	REGISTER	DESCRIPTION
↑	RESERVED	
FF93	WDCSR	Watchdog timer control and status register
FF92	VECINT	Vector interrupt register
↑	RESERVED	
FF90	ROMS	ROM shadow configuration register
↑	RESERVED	
FF83	OEPBCNT_0	Output endpoint_0: Byte count register
FF82	OEPCNFG_0	Output endpoint_0: Configuration register
FF81	IEPBCNT_0	Input endpoint_0: Byte count register
FF80	IEPCNFG_0	Input endpoint_0: Configuration register

Table 4–4. EDB Memory Locations

ADDRESS	REGISTER	DESCRIPTION
↑	RESERVED	
FF58	IEPCNF_3	Input endpoint_3: Configuration
FF50	IEPCNF_2	Input endpoint_2: Configuration
FF48	IEPCNF_1	Input endpoint_1: Configuration
FF47 ↑ FF20	RESERVED	
FF18	OEPCNF_3	Output endpoint_3: Configuration
FF10	OEPCNF_2	Output endpoint_2: Configuration
FF08	OEPCNF_1	Output endpoint_1: Configuration
FF07 ↑ FF00	(8 bytes)	Setup packet block
FEFF ↑ FEF8	(8 bytes)	Input endpoint-0 buffer
FEF7 ↑ FEF0	(8 bytes)	Output endpoint-0 buffer
FEEF ↑ F800	TOPBUFF STABUFF	Top of buffer space Buffer space Start of buffer space

4.3 Endpoint Descriptor Block (EDB–1 to EDB–3)

Data transfers between the USB, the MCU, and external devices that are defined by an endpoint descriptor Block (EDB). Three input- and three output-EDBs are provided. With the exception of EDB–0 (I/O endpoint–0), all EDBs are located in SRAM as per Table 4–3. Each EDB contains information describing the X- and Y-buffers. In addition, each EDB provides general status information.

Table 4–5 illustrates the EDB entries for EDB–1 to EDB–3. EDB–0 registers are described separately.

Table 4–5. EDB Entries in RAM (n = 1 to 3)

OFFSET	ENTRY NAME	DESCRIPTION
07	EPSIZXY_n	I/O Endpoint_n: X/Y-buffer size
06	EPBCTY_n	I/O Endpoint_n: Y-byte count
05	EPBBAY_n	I/O Endpoint_n: Y-buffer base address
04	SPARE	Not used
03	SPARE	Not used
02	EPBCTX_n	I/O Endpoint_n: X-byte count
01	EPBBAX_n	I/O Endpoint_n: X-buffer base address
00	EPCNF_n	I/O Endpoint_n: Configuration

4.3.1 OEPCNF_n: Output Endpoint Configuration (n = 1 to 3) (Addr:FF08, FF10, FF18)

7	6	5	4	3	2	1	0
UBME	**ISO=0**	**TOGLE**	**DBUF**	**STALL**	**USBIE**	**RSV**	**RSV**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION
1–0	RSV	x	Reserved = 0
2	USBIE	x	USB interrupt enable on transaction completion. Set/cleared by the MCU USBIE = 0 No interrupt USBIE = 1 Interrupt on transaction completion
3	STALL	0	USB stall condition indication. Set/cleared by the MCU
			STALL = 0 No stall Stall = 1 USB stall condition. If set by the MCU, a STALL handshake is initiated and the bit is cleared by the MCU.
4	DBUF	x	Double-buffer enable. Set/cleared by the MCU DBUF = 0 Primary buffer only (X-buffer only) DBUF = 1 Toggle bit selects buffer
5	TOGLE	x	USB toggle bit. This bit reflects the toggle sequence bit of DATA0, DATA1
6	ISO	x	ISO = 0 Nonisochronous transfer. This bit must be cleared by the MCU since only nonisochronous transfer is supported
7	UBME	x	UBM enable/disable bit. Set/cleared by the MCU UBME = 0 UBM cannot use this endpoint UBME = 1 UBM can use this endpoint

4.3.2 OEPBBAX_n: Output Endpoint X-Buffer Base Address (n = 1 to 3)

7	6	5	4	3	2	1	0
A10	**A9**	**A8**	**A7**	**A6**	**A5**	**A4**	**A3**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION
7–0	A[10:3]	x	A[10:3] of X-buffer base address (padded with 3 LSB of zeros for a total of 11 bits). This value is set by the MCU. The UBM or DMA uses this value as the start-address of a given transaction. Note that the UBM or DMA does not change this value at the end of a transaction.

4.3.3 OEPBCTX_n: Output Endpoint X Byte Count (n = 1 to 3)

7	6	5	4	3	2	1	0
NAK	**C6**	**C5**	**C4**	**C3**	**C2**	**C1**	**C0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION
6–0	C[6:0]	x	**X-buffer byte count:** X000.0000b Count = 0 X000.0001b Count = 1 byte : : X011.1111b Count = 63 bytes X100.0000b Count = 64 bytes Any value ≥ 100.0001b may result in unpredictable results.
7	NAK	x	NAK =0 No valid data in buffer. Ready for host OUT NAK = 1 Buffer contains a valid packet from host (gives NAK response to Host OUT request)

4.3.4 OEPBBAY_n: Output Endpoint Y-Buffer Base Address (n = 1 to 3)

7	6	5	4	3	2	1	0
A10	**A9**	**A8**	**A7**	**A6**	**A5**	**A4**	**A3**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION
7–0	A[10:3]	x	A[10:3] of Y-buffer base address (padded with 3 LSB of zeros for a total of 11 bits). This value is set by the MCU. The UBM or DMA uses this value as the start-address of a given transaction. Furthermore, UBM or DMA does not change this value at the end of a transaction.

4.3.5 OEPBCTY_n: Output Endpoint Y-Byte Count (n = 1 to 3)

7	6	5	4	3	2	1	0
NAK	**C6**	**C5**	**C4**	**C3**	**C2**	**C1**	**C0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION
6–0	C[6:0]	x	**Y-byte count:** X000.0000b Count = 0 X000.0001b Count = 1 byte : : X011.1111b Count = 63 bytes X100.0000b Count = 64 bytes Any value ≥ 100.0001b may result in unpredictable results.
7	NAK	x	NAK =0 No valid data in buffer. Ready for host OUT NAK = 1 Buffer contains a valid packet from host (gives NAK response to Host OUT request)

4.3.6 OEPSIZXY_n: Output Endpoint X-/Y-Buffer Size (n =1 to 3)

7	6	5	4	3	2	1	0
RSV	**S6**	**S5**	**S4**	**S3**	**S2**	**S1**	**S0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION
6–0	C[6:0]	x	**X- and Y-buffer size:** 0000.0000b Size = 0 0000.0001b Size = 1 byte : : 0011.1111b Size = 63 bytes 0100.0000b Size = 64 bytes Any value ≥ 100.0001b may result in unpredictable results.
7	RSV	x	Reserved = 0

4.3.7 IEPCNF_n: Input Endpoint Configuration (n = 1 to 3) (Addr:FF48, FF50, FF58)

7	6	5	4	3	2	1	0
UBME	**ISO=0**	**TOGLE**	**DBUF**	**STALL**	**USBIE**	**RSV**	**RSV**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION
1–0	RSV	x	Reserved = 0
2	USBIE	x	USB interrupt enable on transaction completion USBIE = 0 No interrupt USBIE = 1 Interrupt on transaction completion
3	STALL	0	USB stall condition indication. Set by the UBM but can be set/cleared by the MCU STALL = 0 No stall STALL = 1 USB stall condition. If set by the MCU a STALL handshake is initiated and the bit is cleared automatically.
4	DBUF	x	Double buffer enable DBUF = 0 Primary buffer only (X-buffer only) DBUF = 1 Toggle bit selects buffer
5	TOGLE	x	USB toggle bit. This bit reflects the toggle sequence bit of DATA0, DATA1
6	ISO	x	ISO = 0 Nonisochronous transfer. This bit must be cleared by the MCU since only nonisochronous transfer is supported
7	UBME	x	UBM enable/disable bit. Set/cleared by the MCU UBME = 0 UBM cannot use this endpoint UBME = 1 UBM can use this endpoint

4.3.8 IEPBBAX_n: Input Endpoint X-buffer Base Address (n = 1 to 3)

7	6	5	4	3	2	1	0
A10	**A9**	**A8**	**A7**	**A6**	**A5**	**A4**	**A3**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION
7–0	A[10:3]	x	A[10:3] of X-buffer base address (padded with 3 LSB of zeros for a total of 11 bits). This value is set by the MCU. The UBM or DMA uses this value as the start-address of a given transaction, but note that the UBM or DMA does not change this value at the end of a transaction.

4.3.9 IEPBCTX_n: Input Endpoint X-Byte Count (n = 1 to 3)

7	6	5	4	3	2	1	0
NAK	**C6**	**C5**	**C4**	**C3**	**C2**	**C1**	**C0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION
6–0	C[6:0]	x	**X-Buffer byte count:** X000.0000b Count = 0 X000.0001b Count = 1 byte : : X011.1111b Count = 63 bytes X100.0000b Count = 64 bytes Any value ≥ 100.0001b may result in unpredictable results.
7	NAK	x	NAK = 0 Buffer contains a valid packet for host-IN transaction NAK = 1 Buffer is empty (gives NAK response to host-OUT request)

4.3.10 IEPBBAY_n: Input Endpoint Y-Buffer Base Address (n = 1 to 3)

7	6	5	4	3	2	1	0
A10	**A9**	**A8**	**A7**	**A6**	**A5**	**A4**	**A3**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION
7–0	A[10:3]	x	A[10:3] of Y-buffer base address (padded with 3 LSB of zeros for a total of 11 bits). This value is set by the MCU. The UBM or DMA uses this value as the start-address of a given transaction, but note that the UBM or DMA does not change this value at the end of a transaction.

4.3.11 IEPBCTY_n: Input Endpoint Y-Byte Count (n = 1 to 3)

7	6	5	4	3	2	1	0
NAK	**C6**	**C5**	**C4**	**C3**	**C2**	**C1**	**C0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION
6–0	C[6:0]	x	**Y-Byte count:** X000.0000b Count = 0 X000.0001b Count = 1 byte : : X011.1111b Count = 63 bytes X100.0000b Count = 64 bytes Any value ≥ 100.0001b may result in unpredictable results.
7	NAK	x	NAK =0 Buffer contains a valid packet for host-IN transaction NAK = 1 Buffer is empty (gives NAK response to host-IN request)

4.3.12 IEPSIZXY_n: Output Endpoint X-/Y-Buffer Size (n = 1 to 3)

7	6	5	4	3	2	1	0
RSV	**S6**	**S5**	**S4**	**S3**	**S2**	**S1**	**S0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION
6–0	C[6:0]	x	**X- and Y-buffer size:** 0000.0000b Size = 0 0000.0001b Size = 1 byte : : 0011.1111b Size = 63 bytes 0100.0000b Size = 64 bytes Any value ≥ 100.0001b may result in unpredictable results.
7	RSV	x	Reserved = 0

4.4 Endpoint-0 Descriptor Registers

Unlike registers EDB–1 to EDB–3, which are defined as memory entries in SRAM, endpoint–0 is described by a set of four registers (two for output and two for input). The registers and their respective addresses, used for EDB–0 description, are defined in Table 4–6. EDB–0 has no base-address register, since these addresses are hardwired into FEF8 and FEF0. Note that the bit positions have been preserved to provide consistency with EDB–n (n = 1 to 3).

Table 4–6. Input/Output EDB-0 Registers

ADDRESS	REGISTER NAME	DESCRIPTION	BASE ADDRESS
FF83 FF82	OEPBCNT_0 OEPCNFG_0	Output endpoint_0: Byte count register Output endpoint_0: Configuration register	FEF0
FF81 FF80	IEPBCNT_0 IEPCNFG_0	Output endpoint_0: Byte count register Output endpoint_0: Configuration register	FEF8

4.4.1 IEPCNFG_0: Input Endpoint-0 Configuration Register (Addr:FF80)

7	6	5	4	3	2	1	0
UBME	**RSV**	**TOGLE**	**RSV**	**STALL**	**USBIE**	**RSV**	**RSV**
R/W	R/O	R/O	R/O	R/W	R/W	R/O	R/O

BIT	NAME	RESET	FUNCTION
1–0	RSV	0	Reserved = 0
2	USBIE	0	USB interrupt enable on transaction completion. Set/cleared by the MCU. USBIE = 0 No interrupt USBIE = 1 Interrupt on transaction completion
3	STALL	0	USB stall condition indication. Set/cleared by the MCU STALL = 0 No stall STALL = 1 USB stall condition. If set by the MCU a STALL handshake is initiated and the bit is cleared automatically by the next setup transaction.
4	RSV	0	Double buffer enable DBUF = 0 Primary buffer only (X-buffer only) DBUF = 1 Toggle bit selects buffer
5	TOGLE	0	USB toggle bit. This bit reflects the toggle sequence bit of DATA0, DATA1.
6	RSV	0	Reserved = 0
7	UBME	0	UBM enable/disable bit. Set/cleared by the MCU UBME = 0 UBM cannot use this endpoint UBME = 1 UBM can use this endpoint

4.4.2 IEPBCNT_0: Input Endpoint-0 Byte Count Register (Addr:FF81)

7	6	5	4	3	2	1	0

NAK	RSV	RSV	RSV	C3	C2	C1	C0
R/W	R/O	R/O	R/O	R/W	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION
3–0	C[3:0]	0h	**Byte count:** 0000b Count = 0 : : 1111b Count = 7 1000b Count = 8 1001b to 1111b are reserved. (If used, they default to 8)
6–4	rsv	0	Reserved = 0
7	NAK	1	NAK =0 Buffer contains a valid packet for host-IN transaction NAK = 1 Buffer is empty (gives NAK response to host-IN request)

4.4.3 OEPCNFG_0: Output Endpoint-0 Configuration Register (Addr:FF82)

7	6	5	4	3	2	1	0
UBME	RSV	TOGLE	RSV	STALL	USBIE	RSV	RSV
R/W	R/O	R/O	R/O	R/W	R/W	R/O	R/O

BIT	NAME	RESET	FUNCTION
1–0	RSV	0	Reserved = 0
2	USBIE	0	USB interrupt enable on transaction completion. Set/cleared by the MCU. USBIE = 0 No interrupt USBIE = 1 Interrupt on transaction completion
3	STALL	0	USB stall condition indication. Set/cleared by the MCU STALL = 0 No stall STALL = 1 USB stall condition. If set by the MCU, a STALL handshake is initiated and the bit is cleared automatically.
4	RSV	0	Reserved = 0
5	TOGLE	0	USB \toggle bit. This bit reflects the toggle sequence bit of DATA0, DATA1.
6	RSV	0	Reserved = 0
7	UBME	0	UBM enable/disable bit. Set/cleared by the MCU UBME = 0 UBM cannot use this endpoint UBME = 1 UBM can use this endpoint

4.4.4 OEPBCNT_0: Output Endpoint-0 Byte Count Register (Addr:FF83)

7	6	5	4	3	2	1	0
NAK	RSV	RSV	RSV	C3	C2	C1	C0
R/W	R/O	R/O	R/O	R/O	R/O	R/O	R/O

BIT	NAME	RESET	FUNCTION
3–0	C[3:0]	0h	**Byte count:** 0000b Count = 0 : : 1111b Count = 7 1000b Count = 8 1001b to 1111b are reserved
6–4	rsv	0	Reserved = 0
7	NAK	1	NAK =0 No valid data in buffer. Ready for host OUT NAK = 1 Buffer contains a valid packet from host (gives NAK response to host-IN request).

5 USB

5.1 USB Registers

5.1.1 FUNADR: Function Address Register (Addr:FFFF)

This register contains the device function address.

7	6	5	4	3	2	1	0
RSV	**FA6**	**FA5**	**FA4**	**FA3**	**FA2**	**FA1**	**FA0**
R/O	R/W	R/W	R/W	R/W	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION
6–0	FA[6:0]	0	These bits define the current device address assigned to the function. The MCU writes a value to this register because of the SET-ADDRESS host command.
7	RSV	0	Reserved = 0

5.1.2 USBSTA: USB Status Register (Addr:FFFE)

All bits in this register are set by the hardware and are cleared by the MCU when writing a 1 to the proper bit location (writing a 0 has no effect). In addition, each bit can generate an interrupt if its corresponding mask bit is set (R/C notation indicates read and clear only by the MCU).

7	6	5	4	3	2	1	0
RSTR	**SUSR**	**RESR**	**RSV**	**URRI**	**SETUP**	**WAKEUP**	**STPOW**
R/C	R/C	R/C	R/O	R/C	R/C	R/C	R/C

BIT	NAME	RESET	FUNCTION
0	STPOW	0	SETUP Overwrite bit. Set by hardware when setup packet is received while there is already a packet in the setup buffer. STPOW = 0 MCU can clear this bit by writing a 1 (writing 0 has no effect). STPOW = 1 SETUP overwrite
1	WAKEUP	0	Remote wakeup bit WAKEUP = 0 The MCU can clear this bit by writing a 1 (writing 0 has no effect). WAKEUP = 1 Remote wakeup request from WAKEUP pin
2	SETUP	0	SETUP transaction received bit. As long as SETUP is 1, IN and OUT on endpoint–0 are NAKed, regardless of their real NAK bits value. SETUP = 0 MCU can clear this bit by writing a 1 (writing 0 has no effect). SETUP = 1 SETUP transaction received
3	URRI	0	UART RI status bit – a rising edge causes this bit to be set. URRI = 0 URRI = 0 The MCU can clear this bit by writing a 1 (writing 0 has no effect). URRI = 1 URRI = 1 Ring detected, which is used to wake the chip up (bring it out of suspend).
4	RSV	0	Reserved
5	RESR	0	Function resume request bit RESR = 0 The MCU can clear this bit by writing a 1 (writing 0 has no effect). RESR = 1 Function resume is detected
6	SUSR	0	Function suspended request bit. This bit is set in response to a global or selective suspend condition. FSUSP = 0 The MCU can clear this bit by writing a 1 (writing 0 has no effect). FSUSP = 1 Function suspend is detected
7	RSTR	0	Function reset request bit. This bit is set in response to host initiating a port reset. This bit is not affected by the USB function reset. FRST = 0 The MCU can clear this bit by writing a 1 (writing 0 has no effect). FRST = 1 Function reset is detected

5.1.3 USBMSK: USB Interrupt Mask Register (Addr:FFFD)

7	6	5	4	3	2	1	0
RSTR	**SUSR**	**RESR**	**RSV**	**UR1RI**	**SETUP**	**WAKEUP**	**STPOW**
R/W	R/W	R/W	R/O	R/W	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION
0	STPOW	0	SETUP overwrite interrupt-enable bit STPOW = 0 STPOW interrupt disabled STPOW = 1 STPOW interrupt enabled
1	WAKEUP	0	Remote wakeup interrupt enable bit WAKEUP = 0 WAKEUP interrupt disable WAKEUP = 1 WAKEUP interrupt enable
2	SETUP	0	SETUP interrupt enable bit SETUP = 0 SETUP interrupt disabled SETUP = 1 SETUP interrupt enabled
3	UR1RI	0	UART 1 R1 interrupt enable bit URRI = 0 UR1RI interrupt disable URRI = 1 UR1RI interrupt enable
4	RSV	0	Reserved
5	RESR	0	Function resume interrupt enable bit RESR = 0 Function resume interrupt disabled RESR = 1 Function resume interrupt enabled
6	SUSR	0	Function suspend interrupt enable FSUSP = 0 Function suspend interrupt disabled FSUSP = 1 Function suspend interrupt enabled
7	RSTR	0	Function reset interrupt bit. This bit is not affected by USB function reset. FRST = 0 Function reset interrupt disabled FRST = 1 Function reset interrupt enabled

5.1.4 USBCTL: USB Control Register (Addr:FFFC)

Unlike the rest of the registers, this register is cleared by the power-up reset signal only. The USB reset cannot reset this register (see Figure 5–1**)**.

7	6	5	4	3	2	1	0
CONT			**FRSTE**				
R/W	R/O	R/W	R/W	R/W	R/O	R/W	R/W

BIT	NAME	RESET	
0	DIR	0	As a response to a setup packet, the MCU decodes the request and sets/clears this bit to reflect the data transfer direction. DIR = 0 USB data-OUT transaction (from host to TUSB3410) DIR = 1 USB data-IN transaction (from TUSB3410 to host)
1	SIR	0	SETUP interrupt-status bit. This bit is controlled by the MCU to indicate to the hardware when the SETUP interrupt is being served. SIR = 0 SETUP interrupt is not served. The MCU clears this bit before exiting the SETUP interrupt routine. SIR = 1 SETUP interrupt is in progress. The MCU sets this bit when servicing the SETUP interrupt.
2	RSV	0	Reserved = 0
3	RSV	0	Reserved = 0
4	FRSTE	1	Function reset-connection bit. This bit connects/disconnects the USB function reset to/from the MCU reset. FRSTE = 0 Function reset is not connected to MCU reset FRSTE = 1 Function reset is connected to MCU reset
5	RWUP	0	Device remote wakeup request. This bit is set by the MCU and is cleared automatically. RWUP = 0 Writing a 0 to this bit has no effect RWUP = 1 When MCU writes a 1, a remote-wakeup pulse is generated.
6	IREN	0	IR mode enable. This bit is set and cleared by firmware. IREN = 0 IR encoder/decoder is disabled, UART mode is selected IREN = 1 IR encoder/decoder is enabled, UART mode is deselected
7	CONT	0	Connect/disconnect bit CONT = 0 Upstream port is disconnected. Pullup disabled. CONT = 1 Upstream port is connected. Pullup enabled.

5.1.5 MODECNFG: Mode Configuration Register (Addr:FFFB)

This register is cleared by the power-up reset signal only. The USB reset cannot reset this register.

7	6	5	4	3	2	1	0
RSV	RSV	RSV	RSV	CLKSLCT	CLKOUTEN	SOFTSW	TXCNTL
R/O	R/O	R/O	R/O	R/W	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION
0	TXCNTL	0	Transmit output control: Hardware or firmware switching select for 3-state serial output buffer.
			TXCNTL = 0 Hardware automatic switching is selected TXCNTL = 1 Firmware toggle switching is selected
1	SOFTSW	0	Soft switch: Firmware controllable 3-state output buffer enable for serial output pin.
			SOFTSW = 0 Serial output buffer is enabled SOFTSW = 1 Serial output buffer is disabled
2	CLKOUTEN	0	Clock output enable: Enable/disable the clock output at CLKOUT terminal.
			CLKOUTEN = 0 Clock output is disabled. Device drives low at CLKOUT terminal. CLKOUTEN = 1 Clock output is enabled
3	CLKSLCT	0	Clock output source select: Select between 3.556-MHz fixed clock or UART baud out clock as output clock source.
			CLKSLCT = 0 UART baud out clock is selected as clock output CLKSLCT = 1 Fixed 3.556-MHz free running clock is selected as clock output
4–7	RSV	0	Reserved

Clock Output Control

The CLKOUTEN bit in the Mode Configuration Register (MODECNFG) is used to enable or disable the clock output at the CLKOUT terminal of the TUSB3410. The power up default of CLKOUT is disabled. Firmware can write a 1 to enable the clock output if needed.

The CLKSLCT bit in the MODECNFG register is used to select the output clock source from either a fixed 3.556-MHz free-running clock or the UART BaudOut clock.

5.1.6 Vendor ID/Product ID

USB–IF and Microsoft WHQL certification requires that end equipment makers use their own unique vendor ID and product ID for each product (model). OEMs cannot use silicon vendor's (for instance, TI's default) VID/PID in their end products. A unique VID/PID combination will avoid potential driver conflicts and enable logo certification. See **www.usb.org** for more information.

5.1.7 SERNUM7: Device Serial Number Register (Byte 7) (Addr:FFEF)

Each TUSB3410 chip has a unique 64-bit serial die id number, which is generated during manufacturing. The die id is incremented sequentially, however there is no assurance without skip in the die id number. The device serial number register utilizes (mirrors) this unique 64-bit serial die id number.

After power-up reset, this read-only register (SERNUM7) contains the most significant byte (byte 7) of the complete 64-bit device serial number. The USB reset cannot reset this register.

7	6	5	4	3	2	1	0
D63	D62	D61	D60	D59	D58	D57	D56
R/O	R/O	R/O	R/O	R/O	R/O	R/O	R/O

BIT	NAME	RESET	FUNCTION
7–0	D[7:0]	Device serial number byte 7 value	Device serial number byte 7 value

Procedure to load device serial number value in shared RAM:

- After power-up reset, boot code copies the predefined USB descriptors to shared RAM. As a result, the default serial number hard-coded in the boot code (0x00 hex) is copied to the shared RAM data space.
- Once the boot code finishes copying descriptors, it performs a read to the SERNUM7 to SERNUM0 registers and overwrites the device serial number value stored in the shared RAM with the one found in the SERNUM7 to SERNUM0 registers.
- Once the boot code finishes the read to SERNUM7 – SERNUM0 registers, it then checks if EEPROM is present on the I^2C port. If the EEPROM is present and contains a valid device serial number as part of the USB device descriptor information stored in EEPROM, the boot code overwrites the serial number value stored in shared RAM with the one found in EEPROM. Otherwise, the device serial number value stored in shared RAM stays unchanged from previous step.
- In summary, the serial number value in external EEPROM has the highest priority to be loaded into shared RAM data space. The serial number value stored in shared RAM is used as part of the valid device descriptor information during normal operation.

5.1.8 SERNUM6: Device Serial Number Register (Byte 6) (Addr:FFEE)

The device serial number register utilizes (mirrors) this unique 64-bit serial die id number.

After power-up reset, this read-only register (SERNUM6) contains byte 6 of the complete 64-bit device serial number. The USB reset cannot reset this register.

7	6	5	4	3	2	1	0
D55	**D54**	**D53**	**D52**	**D51**	**D50**	**D49**	**D48**
R/O	R/O	R/O	R/O	R/O	R/O	R/O	R/O

BIT	NAME	RESET	FUNCTION
7–0	D[7:0]	Device serial number byte 6 value	Device serial number byte 6 value

NOTE: See the same procedure described in SERNUM7 register for procedure to load device serial number into the shared RAM.

5.1.9 SERNUM5: Device Serial Number Register (Byte 5) (Addr:FFED)

The device serial number register utilizes (mirrors) this unique 64-bit serial die id number.

After power-up reset, this read-only register (SERNUM5) contains byte 5 of the complete 64-bit device serial number. The USB reset cannot reset this register.

7	6	5	4	3	2	1	0
D47	**D46**	**D45**	**D44**	**D43**	**D42**	**D41**	**D40**
R/O	R/O	R/O	R/O	R/O	R/O	R/O	R/O

BIT	NAME	RESET	FUNCTION
7–0	D[7:0]	Device serial number byte 5 value	Device serial number byte 5 value

NOTE: See the same procedure described in SERNUM7 register for procedure to load device serial number into the shared RAM.

5.1.10 SERNUM4: Device Serial Number Register (Byte 4) (Addr:FFEC)

The device serial number register utilizes (mirrors) this unique 64-bit serial die id number.

After power-up reset, this read-only register (SERNUM4) contains byte 4 of the complete 64-bit device serial number. The USB reset cannot reset this register.

7	6	5	4	3	2	1	0
D39	**D38**	**D37**	**D36**	**D35**	**D34**	**D33**	**D32**
R/O	R/O	R/O	R/O	R/O	R/O	R/O	R/O

BIT	NAME	RESET	FUNCTION
7–0	D[7:0]	Device serial number byte 4 value	Device serial number byte 4 value

NOTE: See the same procedure described in SERNUM7 register for procedure to load device serial number into the shared RAM.

5.1.11 SERNUM3: Device Serial Number Register (Byte 3) (Addr:FFEB)

The device serial number register utilizes (mirrors) this unique 64-bit serial die id number.

After power-up reset, this read-only register (SERNUM3) contains byte 3 of the complete 64-bit device serial number. The USB reset cannot reset this register.

7	6	5	4	3	2	1	0
D31	**D30**	**D29**	**D28**	**D27**	**D26**	**D25**	**D24**
R/O	R/O	R/O	R/O	R/O	R/O	R/O	R/O

BIT	NAME	RESET	FUNCTION
7–0	D[7:0]	Device serial number byte 3 value	Device serial number byte 3 value

NOTE: See the same procedure described in SERNUM7 register for procedure to load device serial number into the shared RAM.

5.1.12 SERNUM2: Device Serial Number Register (Byte 2) (Addr:FFEA)

The device serial number register utilizes (mirrors) this unique 64-bit serial die id number.

After power-up reset, this read-only register (SERNUM2) contains byte 2 of the complete 64-bit device serial number. The USB reset cannot reset this register.

7	6	5	4	3	2	1	0
D23	**D22**	**D21**	**D20**	**D19**	**D18**	**D17**	**D16**
R/O	R/O	R/O	R/O	R/O	R/O	R/O	R/O

BIT	NAME	RESET	FUNCTION
7–0	D[7:0]	0	Device serial number byte 2 value

NOTE: See the same procedure described in SERNUM7 register for procedure to load device serial number into the shared RAM.

5.1.13 SERNUM1: Device Serial Number Register (Byte 1) (Addr:FFE9)

The device serial number register utilizes (mirrors) this unique 64-bit serial die id number.

After power-up reset, this read-only register (SERNUM1) contains byte 1 of the complete 64-bit device serial number. The USB reset cannot reset this register.

7	6	5	4	3	2	1	0
D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
R/O	R/O	R/O	R/O	R/O	R/O	R/O	R/O

BIT	NAME	RESET	FUNCTION
7–0	D[7:0]	Device serial number byte 1 value	Device serial number byte 1 value

NOTE: See the same procedure described in SERNUM7 register for procedure to load device serial number into the shared RAM.

5.1.14 SERNUM0: Device Serial Number Register (Byte 0) (Addr:FFE8)

The device serial number register utilizes (mirrors) this unique 64-bit serial die id number.

After power-up reset, this read-only register (SERNUM0) contains byte 0 of the complete 64-bit device serial number. The USB reset cannot reset this register.

7	6	5	4	3	2	1	0
D7	D6	D5	D4	D3	D2	D1	D0
R/O	R/O	R/O	R/O	R/O	R/O	R/O	R/O

BIT	NAME	RESET	FUNCTION
7–0	D[7:0]	Device serial number byte 0 value	Device serial number byte 0 value

NOTE: See the same procedure described in SERNUM7 register for procedure to load device serial number into the shared RAM.

5.1.15 Function Reset And Power-Up Reset Interconnect

Figure 5–1 represents the logical connection of the USB-function reset ($\overline{\text{USBR}}$) and power-up reset ($\overline{\text{RESET}}$) pins. The internal $\overline{\text{RESET}}$ signal is generated from the $\overline{\text{RESET}}$ pin ($\overline{\text{PURS}}$ signal) or from the USB reset ($\overline{\text{USBR}}$ signal). The $\overline{\text{USBR}}$ can be enabled or disabled by the FRSTE bit in the USBCTL register (on power up, FRSTE = 0). The internal $\overline{\text{RESET}}$ is used to reset all registers and logic, with the exception of the USBCTL and GLOBCTL registers which are cleared by the $\overline{\text{PURS}}$ signal only.

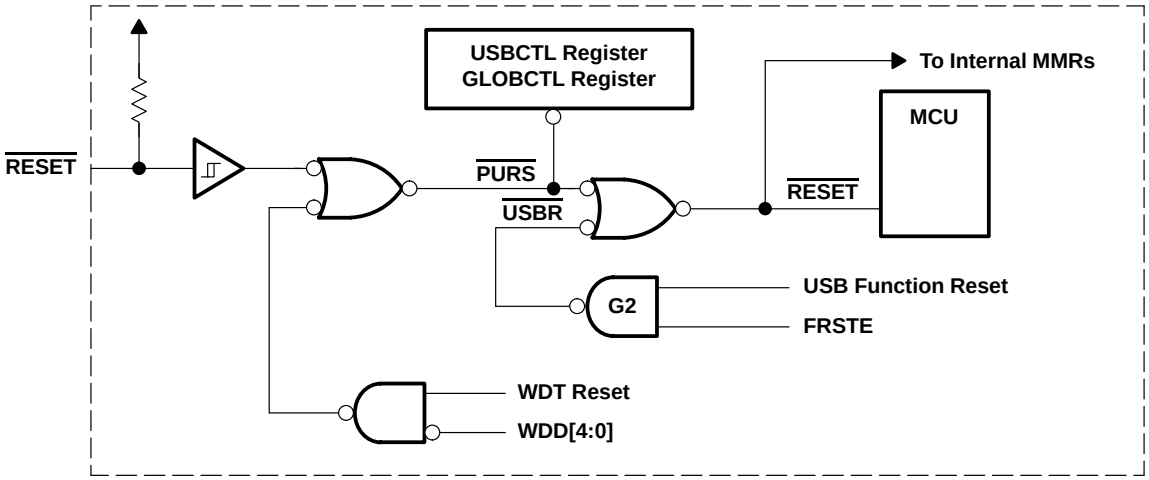


Figure 5–1. Reset Diagram

5.1.16 Pullup Resistor Connect/Disconnect

The TUSB3410 enumeration can be activated by the MCU (there is no need to disconnect the cable physically). Figure 5–2 represents the implementation of the TUSB3410 connect and disconnect from a USB up-stream port. When CONT = 1 in the USBCTL register, the CMOS driver sources V_{DD} to the pullup resistor (PUR pin) presenting a normal connect condition to the USB hub (high speed). When CONT = 0, the PUR pin is driven low. In this state, the 1.5-kΩ resistor is connected to GND, resulting in the device disconnection state. The PUR driver is a CMOS driver that can provide (V_{DD} – 0.1 V) minimum at 8-mA source current.

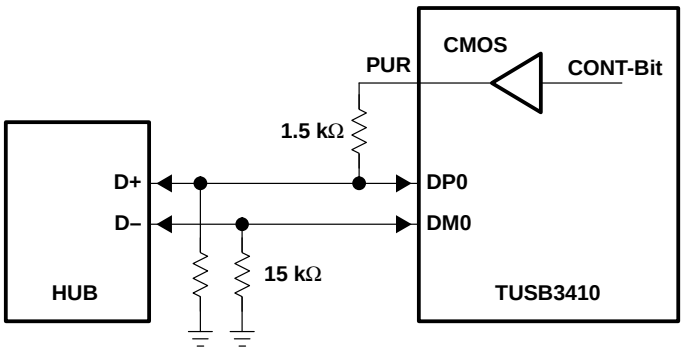


Figure 5–2. Pullup Resistor Connect/Disconnect Circuit

6 DMA Controller

Table 6–1 outlines the DMA channels and their associated transfer directions. Two channels are provided for data transfer between the host and the UART.

Table 6–1. DMA Controller Registers

DMA CHANNEL	TRANSFER DIRECTION	COMMENTS
DMA–1	Host to UART	DMA writes to UART TDR register
DMA–3	UART to host	DMA reads from UART RDR register

6.1 DMA Controller Registers

Each DMA channel can point to one of three EDBs (EDB[3:1]) and transfer data to/from the UART channel. The DMA can move data from a given out-point buffer (defined by EDB) to the destination port. Similarly, the DMA can move data from a port to a given input-endpoint buffer. Two modes of DMA transfers are supported: burst and continuous.

- **Burst (CNT = 0) Mode**

  The DMA stops at the end of a block-data transfer (or if an error condition occurred) and interrupts the MCU. It is the responsibility of the MCU to update the X/Y bit and the NAK bit in the EDB.

- **Continuous (CNT = 1) Mode**

  At the end of a block transfer the DMA updates the byte count and NAK bit in the EDB when receiving. In addition, it uses the X/Y bit to switch automatically, without interrupting the MCU (the X/Y bit toggle is performed by the UBM). The DMA stops only when a time-out or error condition occurs. When the DMA is transmitting (from the X/Y buffer) it continues alternating between X/Y buffers until it detects a byte count smaller than the buffer size (buffer size is typically 64 bytes). At that point it completes the transfer and stops.

6.1.1 DMACDR1: DMA Channel Definition Register (UART Transmit Channel) (Addr:FFE0)

These registers are used to define the EDB number that the DMA uses for data transfer to the UARTS. In addition, these registers define the data transfer direction and selects X or Y as the transaction buffer.

7	6	5	4	3	2	1	0
EN	**INE**	**CNT**	**XY**	**T/R**	**E2**	**E1**	**E0**
R/W	R/W	R/W	R/W	R/O	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION	
2–0	E[2:0]	0	Endpoint descriptor pointer. This field points to a set of EDB registers that is to be used for a given transfer.	
3	T/R	0	This bit is always zero, indicating that the DMA data transfer is from SRAM to the UART TDR register. (The MCU cannot change this bit.)	
4	XY	0	X/Y buffer select bit. Valid only when CNT = 0	
			XY = 0 XY = 1	Next buffer to transmit/receive is the X buffer Next buffer to transmit/receive is the Y buffer
5	CNT	0	DMA continuous transfer control bit. This bit defines the mode of the DMA transfer.	
			CNT = 0	**Burst mode:** The DMA stops the transfer when the byte count is zero or when a partial packet has been received (byte count < 64). At the end of transfer, the high-to-low transition of EN interrupts the MCU (if enabled). In this mode, the X/Y bit is set by the MCU to define the current buffer (X or Y).
			CNT = 1	**Continuous mode:** In this mode, the DMA and UBM alternate between the X- and Y-buffers. The DMA sets the X/Y bit and the UBM uses it for the transfer. The DMA alternates between the X-/Y-buffers and continues transmitting (from X-/Y-buffer) without MCU intervention. The DMA terminates, and interrupts the MCU, under the following conditions: 1. When the UBM byte count < buffer size (in EDB), the DMA transfers the partial packet and interrupt the MCU on completion. 2. Transaction timer expires. The DMA interrupts the MCU.
6	INE	0	DMA Interrupt enable/disable bit. This bit is used to enable/disable the interrupt on transfer completion.	
			INE = 0	Interrupt is disabled. In addition, PPKT and TXFT do not clear the EN-bit and the DMAC is not disabled.
			INE = 1	Enables the EN interrupt. When this bit is set, the DMA interrupts the MCU on a 1 to 0 transition of the EN bit. (When transfer is completed, EN = 0)
7	EN	0	DMA channel enable bit. The MCU sets this bit to start the DMA transfer. When the transfer completes, or when it is terminated due to error, this bit is cleared. The 1 to 0 transition of this bit generates an interrupt (if interrupt is enabled).	
			EN = 0	DMA is halted. The DMA is halted when the byte count reaches zero or transaction time-out occurs. When halted, the DMA updates the byte count, sets NAK = 0 in OEDB, and interrupts the MCU (if INE = 1).
			EN = 1	Setting this bit starts the DMA transfer.

6.1.2 DMACSR1: DMA Control And Status Register (UART Transmit Channel) (Addr:FFE1)

This register is used to define the transaction time-out value. In addition, it contains a completion code that reports any errors or a time-out condition.

7	6	5	4	3	2	1	0
TEN	**C4**	**C3**	**C2**	**C1**	**C0**	**TXFT**	**PPKT**
R/W	R/W	R/W	R/W	R/W	R/W	R/C	R/C

BIT	NAME	RESET	FUNCTION	
0	PPKT	0	Partial packet condition bit. This bit is set by the DMA and cleared by the MCU (see Table 6–2).	
			PPKT = 0	No partial-packet condition
			PPKT = 1	Partial-packet condition detected. When IEN = 0, this bit does not clear the EN bit in DMACDR; therefore, the DMAC stays enabled, ready for the next transaction. Clears when MCU writes a 1. Writing a 0 has no effect.
1	TXFT	0	Transfer time-out condition (see Table 6–2)	
			TXFT = 0	DMA stopped transfer without time-out
			TXFT = 1	DMA stopped due to transaction time-out. When IEN = 0, this bit does not clear the EN bit in DMACDR; therefore, the DMAC stays enabled, ready for the next transaction. DMA clears when the MCU writes a 1. Writing a 0 has no effect.
6–2	C[4:0]	0	This field is used to define the transaction time-out value in 1-ms increments. This value is loaded to a down counter every time a byte transfer occurs. The down counter is decremented every SOF pulse (1 ms). If the counter decrements to zero it sets TXFT = 1 (in DMACSR register) and halts the DMA transfer. The counter starts counting only when TEN = 1 and EN = 1 (in DMACDR**)** and the first byte has been transmitted (see Figure 6–1). 00000 = 0-ms time-out : : 11111 = 31-ms time-out	
7	TEN	0	Transaction time-out counter enable/disable bit.	
			TEN = 0 TEN = 1	Counter is disabled (does not time-out) Counter is enabled

Table 6–2. DMA OUT-Termination Condition

OUT TERMINATION	TXFT	PPKT	COMMENTS
UART partial packet	0	1	This condition occurs when the host sends a partial packet.
UART time-out	1	0	This condition occurs when X- and Y-output buffers are full and the UART transmitter cannot transmit (due to flow-control restriction) or if host has no data to transmit.

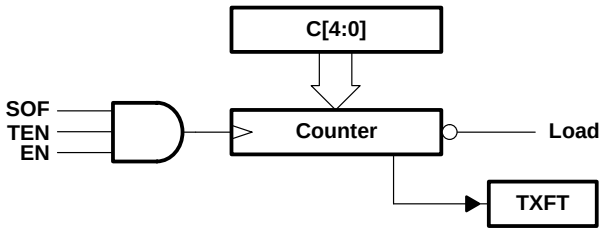


Figure 6–1. Transaction Time-Out Diagram

6.1.3 DMACDR3: DMA Channel Definition Register (UART Receive Channel) (Addr:FFE4)

These registers are used to define the EDB number that the DMA uses for data transfer from the UARTS. In addition, these registers define the data transfer direction and selects X or Y as the transaction buffer.

7	6	5	4	3	2	1	0
EN	**INE**	**CNT**	**XY**	**T/R**	**E2**	**E1**	**E0**
R/W	R/W	R/W	R/W	R/O	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION	
2–0	E[2:0]	0	Endpoint descriptor pointer. This field points to a set of EDB registers that are used for a given transfer.	
3	T/R	1	This bit is always 1. This indicates that the DMA data transfer is from UART RDR register to SRAM. (The MCU cannot change this bit.)	
4	XY	0	XY Buffer select bit. Valid only when CNT = 0.	
			XY = 0 XY = 1	Next buffer to transmit/receive is X Next buffer to transmit/receive is Y
5	CNT	0	DMA continuous transfer control bit. This bit defines the mode of the DMA transfer.	
			CNT = 0	**Burst mode:** DMA stops the transfer when the byte count = 0 or when a receiver error occurs. At the end of transfer, the high-to-low transition of EN interrupts the MCU (if enabled). In this mode, the XY bit is set by the MCU to define the current buffer (X or Y).
			CNT = 1	**Continuous mode:** In this mode, the DMA and UBM alternate between the X- and Y-buffers. The UBM sets the XY bit and the DMA uses it for the transfer. The DMA alternates between the X-/Y-buffers and continues receiving (to X-/Y-buffer) without MCU intervention. The DMA terminates the transfer and interrupts the MCU, under the following conditions: 1. Transaction time-out expired: DMA updates EDB and interrupts the MCU. UBM transfers the partial packet to the host. 2. UART receiver error condition: DMA updates EDB and does not interrupt the MCU. UBM transfers the partial packet to the host.
6	INE	0	DMA interrupt enable/disable bit. This bit is used to enable/disable the interrupt on transfer completion.	
			INE = 0	Interrupt is disabled. In addition, OVRUN and TXFT do not clear the EN bit and the DMAX is not disabled.
			INE = 1	Enables the EN interrupt. When this bit is set, the DMA interrupts the MCU on a 1 to 0 transition of the EN bit. (When transfer is completed, EN = 0).
7	EN	0	DMA channel enable bit. The MCU sets this bit to start the DMA transfer. When transfer completes, or when terminated due to error, this bit is cleared. The 1 to 0 transition of this bit generates an interrupt (if interrupt is enabled).	
			EN = 0	DMA is halted. The DMA is halted when transaction time-out occurs, or under a UART receiver-error condition. When halted, the DMA updates the byte count and sets NAK = 0 in IEDB. If the termination is due to transaction time-out, the DMA generates an interrupt. However, if the termination is due to a UART error condition, the DMA does not generate an interrupt. (The UART generates the interrupt.)
			EN = 1	Setting this bit starts the DMA transfer.

6.1.4 DMACSR3: DMA Control And Status Register (UART Receive Channel) (Addr:FFE5)

This register is used to define the transaction time-out value. In addition, it contains a completion code that reports any errors or a time-out condition.

7	6	5	4	3	2	1	0
TEN	**C4**	**C3**	**C2**	**C1**	**C0**	**TXFT**	**OVRUN**
R/W	R/W	R/W	R/W	R/W	R/W	R/C	R/C

BIT	NAME	RESET	FUNCTION	
0	OVRUN	0	Overrun condition bit. This bit is set by DMA and cleared by the MCU (see Table 6–3)	
			OVRUN = 0	No overrun condition
			OVRUN = 1	Overrun condition detected. When IEN = 0, this bit does not clear the EN bit in DMACDR; therefore, the DMAC stays enabled, ready for the next transaction. Clears when the MCU writes a 1. Writing a 0 has no effect.
1	TXFT	0	Transfer time-out condition bit (see Table 6–3)	
			TXFT = 0	DMA stopped transfer without time-out
			TXFT =1	DMA stopped due to transaction time-out. When IEN = 0, this bit does not clear the EN bit in DMACDR; therefore, the DMAC stays enabled, ready for the next transaction. Clears when the MCU writes a 1. Writing a 0 has no effect.
6–2	C[4:0]	00000b	This field is used to define the transaction time-out value in 1-ms increments. This value is loaded to a down counter every time a byte transfer occurs. The down counter is decremented every SOF pulse (1 ms). If the counter decrements to zero it sets TXFT = 1 (in DMACSR register) and halts the DMA transfer. The counter starts counting only when TEN = 1 and EN = 1 (in DMACDR**)** and the first byte has been received (see Figure 6–1). 00000 = 0-ms time-out : : 11111 = 31-ms time-out	
7	TEN	0	Transaction time-out counter enable/disable bit	
			TEN = 0 TEN = 1	Counter is disabled (does not time-out) Counter is enabled

Table 6–3. DMA IN-Termination Condition

IN TERMINATION	TXFT	OVRUN	COMMENTS
UART error	0	0	UART error condition detected
UART partial packet	1	0	This condition occurs when UART receiver has no more data for the host (data starvation).
UART overrun	1	1	This condition occurs when X- and Y-input buffers are full and the UART FIFO is full (host is busy).

6.2 Bulk Data I/O Using the EDB

The UBM (USB buffer manager) and the DMAC (DMA controller) access the EDB to fetch buffer parameters for IN and OUT transactions (IN and OUT are with respect to host). In this discussion, it is assumed that (a) the MCU initialized the EDBs, (b) DMA-continuous mode is being used, (c) double buffering is being used, and (d) the X/Y toggle is controlled by the UBM.

NOTE: The IN and OUT transfers apply to UART.

6.2.1 IN Transaction (TUSB3410 to Host)

1. The MCU initializes the IEDB (64-byte packet, and double buffering is used) and the following DMA registers:
   - **DMACSR:** Defines the transaction time-out value.
   - **DMACDR:** Defines the IEDB being used and the DMA mode of operation (continuous mode). Once this register is set with EN = 1, the transfer starts.

2. The DMA transfers data from the UART to the X buffer. When a block of 64 bytes is transferred, the DMA updates the byte count and sets NAK = 0 in IEDB (indicating to the UBM that the X buffer is ready to be transferred to host). The UBM starts X-buffer transfer to host using the byte-count value in IEDB and toggles the X/Y bit. The DMA continues transferring data from a device to Y-buffer. At the end of the block transfer, the DMA updates the byte count and sets NAK = 0 in IEDB (indicating to the UBM that the Y-buffer is ready to be transferred to host). The DMA continues the transfer from the device to host, alternating between X-and Y-buffers without MCU intervention.
3. Transfer termination: As mentioned, the DMA/UBM continues the data transfer, alternating between the X- and Y-buffers. Termination of the transfer can happen under the following conditions:
   - **Stop Transfer:** The host notifies the MCU (via control-end-point) to stop the transfer. Under this condition, the MCU sets EN = 0 in the DMACDR register.
   - **Partial Packet:** The device receiver has no data to be transferred to host. Under this condition, the byte-count value is less than 64 when the transaction timer time-out occurs. When the DMA detects this condition, it sets TXFT = 1 and OVRUN = 0, updates the byte count and NAK bit (partial packet) in the IEDB, and interrupts the MCU. UBM transfers the partial packet to host.
   - **Buffer Overrun:** The host is busy, X- and Y-buffers are full (X NAK = 0 and Y–NAK = 0) and the DMA cannot write to these buffers. The transaction time-out stops the DMA transfer, the DMA sets TXFT = 1 and OVRUN = 1, and interrupts the MCU.
   - **UART Error Condition:** When receiving from a UART, a receiver-error condition stops the DMA and sets TXFT = 1 and OVRUN = 0, but the EN bit remains set at 1. Therefore, the DMA does not interrupt the MCU. However, the UART generates a status interrupt, notifying the MCU that an error condition has occurred.

6.2.2 OUT Transaction (Host to TUSB3410)

1. The MCU initializes the OEDB (64-byte packet, and double buffering is used) and the following DMA registers:
   - **DMACSR:** Defines the transaction time-out value.
   - **DMACDR:** Defines the OEDB being used, and the DMA mode of operation (continuous mode). Once the EN bit is set to 1 in this register, the transfer starts.
2. The UBM transfers data from host to X-buffer. When a block of 64 bytes is transferred, the UBM updates the byte count and sets NAK = 1 in OEDB (indicating to DMA that the X-buffer is ready to be transferred to the UART). The DMA starts X-buffer transfer using the byte-count value in OEDB. The UBM continues transferring data from host to Y-buffer. At the end of the block transfer, the UBM updates the byte count and sets NAK = 1 in OEDB (indicating to DMA that the Y-buffer is ready to be transferred to device). The DMA continues the transfer from the X-/Y-buffers to the device, alternating between X- and Y-buffers without MCU intervention.
3. Transfer termination: The DMA/UBM continues the data transfer alternating between X- and Y-buffers. The termination of the transfer can happen under the following conditions:
   - **Stop Transfer:** The host notifies the MCU (via control-end point) to stop the transfer. Under this condition, the MCU sets EN = 0 in the DMACDR register.
   - **Partial-Packet:** UBM receives a partial packet from host. Under this condition, the byte-count value is less than 64 and the transaction timer does not time-out. When the DMA detects this condition, it transfers the partial packet to the device, sets TXFT = 0 and PPKT = 1, updates NAK = 0 in OEDB, and interrupts the MCU.
   - **Time-out:** The device is busy, X- and Y-buffers are full (X-NAK = 1 and Y-NAK = 1) and the UBM cannot write to these buffers. Under this condition the transaction timer time-out stops the DMA transfer, sets TXFT = 1 and OVRUN = 0, and interrupts the MCU.

7 UART

7.1 UART Registers

Table 7–1 summarizes the UART registers. These registers are used for data I/O, control, and status information. UART setup is done by the MCU. Data transfer is typically performed by the DMAC. However, the MCU can perform data transfer without DMA; this is useful when debugging the firmware.

Table 7–1. UART Registers Summary

REGISTER NAME	ACCESS	FUNCTION	COMMENTS
RDR	R/O	UART receiver data register	Can be accessed by MCU or DMA
TDR	W/O	UART transmitter data register	Can be accessed by MCU or DMA
LCR	R/W	UART line control register	
FCRL	R/W	UART flow control register	
MCR	R/W	UART modem control register	
LSR	R/O	UART line status register	Can generate an interrupt
MSR	R/O	UART modem status register	Can generate an interrupt
DLL	R/W	UART divisor register (low byte)	
DLH	R/W	UART divisor register (high byte)	
XON	R/W	UART Xon register	
XOFF	R/W	UART Xoff register	
MASK	R/W	UART interrupt mask register	Can control three interrupt sources

7.1.1 RDR: Receiver Data Register (Addr:FFA0)

The receiver data register consists of a 32-byte FIFO. Data received from the SIN pin are converted from serial-to-parallel format and stored in this FIFO. Data transfer from this register to the RAM buffer is the responsibility of the DMA controller.

7	6	5	4	3	2	1	0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/O	R/O	R/O	R/O	R/O	R/O	R/O	R/O

BIT	NAME	RESET	FUNCTION
7–0	D[7:0]	0	Receiver byte

7.1.2 TDR: Transmitter Data Register (Addr:FFA1)

The transmitter data register is double buffered. Data written to this register is loaded into the shift register, and shifted out on SOUT. Data transfer from the RAM buffer to this register is the responsibility of the DMA controller.

7	6	5	4	3	2	1	0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
W/O	W/O	W/O	W/O	W/O	W/O	W/O	W/O

BIT	NAME	RESET	FUNCTION
7–0	D[7:0]	0	Transmit byte

7.1.3 LCR: Line Control Register (Addr:FFA2)

This register controls the data communication format. The word length, number of stop bits, and parity type are selected by writing the appropriate bits to the LCR.

7	6	5	4	3	2	1	0
FEN	**BRK**	**FPTY**	**EPRTY**	**PRTY**	**STP**	**WL1**	**WL0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION
1:0	WL{1–0]	0	Specifies the word length for transmit and receive 00b = 5 bits 01b = 6 bits 10b = 7 bits 11b = 8 bits
2	STP	0	Specifies the number of stop bits for transmit and receive
			STP = 0 1 stop bit (word length = 5, 6, 7, 8) STP = 1 1.5 stop bits (word length = 5) STP = 1 2 stop bits (word length = 6, 7, 8)
3	PRTY	0	Specifies whether parity is used
			PRTY = 0 No parity PRTY = 1 Parity is generated
4	EPRTY	0	Specifies whether even or odd parity is generated
			EPRTY = 0 Odd parity is generated (if PRTY = 1) EPRTY = 1 Even parity is generated (if PRTY = 1)
5	FPTY	0	Selects the forced parity bit
			FPTY = 0 Parity is not forced FPTY = 1 Parity bit is forced. If [EPRTY = 0], the parity bit is forced to 1
6	BRK	0	This bit is the break-control bit
			BRK = 0 Normal operation BRK = 1 Forces SOUT into break condition (logic 0)
7	FEN	0	FIFO enable. This bit is used to disable/enable the FIFO. To reset the FIFO, the MCU clears and then sets this bit.
			FEN = 0 The FIFO is cleared and disabled. When disabled the selected receiver flow control is activated. FEN = 1 The FIFO is enabled and it can receive data.

7.1.4 FCRL: UART Flow Control Register (Addr:FFA3)

This register provides the flow-control modes of operation (see Table 7–3 for more details).

7	6	5	4	3	2	1	0
485E	**DTR**	**RTS**	**RXOF**	**DSR**	**CTS**	**TXOA**	**TXOF**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION
0	TXOF	0	This bit controls the transmitter Xon/Xoff flow control.
			TXOF = 0 Disable transmitter Xon/Xoff flow control TXOF = 1 Enable transmitter Xon/Xoff flow control
1	TXOA	0	This bit controls the transmitter Xon-on-any/Xoff flow control
			TXOA = 0 Disable the transmitter Xon-on-any/Xoff flow control TXOA = 1 Enable the transmitter Xon-on-any/Xoff flow control
2	CTS	0	Transmitter $\overline{\text{CTS}}$ flow-control enable bit
			CTS = 0 Disables transmitter $\overline{\text{CTS}}$ flow control CTS = 1 $\overline{\text{CTS}}$ flow control is enabled, i.e., when $\overline{\text{CTS}}$ input pin is high, transmission is halted; when the $\overline{\text{CTS}}$ pin is low, transmission resumes.
3	DSR	0	Transmitter $\overline{\text{DSR}}$ flow-control enable bit
			DSR = 0 Disables transmitter $\overline{\text{DSR}}$ flow control DSR = 1 $\overline{\text{DSR}}$ flow control is enabled, i.e., when $\overline{\text{DSR}}$ input pin is high, transmission is halted; when the $\overline{\text{DSR}}$ pin is low, transmission resumes.
4	RXOF	0	This bit controls the receiver Xon/Xoff flow control.
			RXOF = 0 Receiver does not attempt to match Xon/Xoff characters RXOF = 1 Receiver searches for Xon/Xoff characters
5	RTS	0	Receiver $\overline{\text{RTS}}$ flow control enable bit
			RTS = 0 Disables receiver $\overline{\text{RTS}}$ flow control RTS = 1 Receiver $\overline{\text{RTS}}$ flow control is enabled. $\overline{\text{RTS}}$ output pin goes high when the receiver FIFO HALT trigger level is reached; it goes low, when the receiver FIFO RESUME receiving trigger level is reached.
6	DTR	0	Receiver $\overline{\text{DTR}}$ flow-control enable bit
			DTR = 0 Disables receiver $\overline{\text{DTR}}$ flow control DTR = 1 Receiver $\overline{\text{DTR}}$ flow control is enabled. $\overline{\text{DTR}}$ output pin goes high when the receiver FIFO HALT trigger level is reached; it goes low, when the receiver FIFO RESUME receiving trigger level is reached.
7	485E	0	RS485 enable bit. This bit is used to configure the UART to control external RS485 transceivers. When configured in half-duplex mode (485E=1), $\overline{\text{RTS}}$ or $\overline{\text{DTR}}$ can be used to enable the RS485 driver or receiver. See Figure 5.
			485E = 0 UART is in normal operation mode (full duplex) 485E = 1 The UART is in half duplex RS485 mode. In this mode $\overline{\text{RTS}}$ and $\overline{\text{DTR}}$ are active with opposite polarity (when $\overline{\text{RTS}}$ = 0, $\overline{\text{DTR}}$ = 1). When the DMA is ready to transmit, it drives $\overline{\text{RTS}}$ = 1 (and $\overline{\text{DTR}}$ = 0) 2–bit–time before transmission starts. When DMA terminates the transmission, it drives $\overline{\text{RTS}}$ = 0 (and $\overline{\text{DTR}}$ = 1) after transmission stops. When 485E is set to 1, the DTR and RTS bits in the MCR register have no effect. Also, see the RCVE bit in *MCR: modem-control register*.

7.1.5 Transmitter Flow Control

On reset (power up, USB or soft reset) the transmitter defaults to the Xon state and the flow control is set to mode–0 (flow control is disabled).

Table 7–2. Transmitter Flow-Control Modes

MODE		3	2	1	0
		DSR	CTS	TXOA	TXOF
0	All flow control is disabled	0	0	0	0
1	Xon/Xoff flow control is enabled	0	0	0	1
2	Xon on any/ Xoff flow control	0	0	1	0
3	Not permissible (see Note 1)	X	X	1	1
4	$\overline{\text{CTS}}$ flow control	0	1	0	0
5	Combination flow control (see Note 2)	0	1	0	1
6	Combination flow control	0	1	1	0
7	$\overline{\text{DSR}}$ flow control	1	0	0	0
9-E	Combination flow control				

NOTES: 1. This is a nonpermissible combination. If used, TXOA and TXOF are cleared.
2. Combination example: Transmitter stops when either CTS or Xoff is detected. Transmitter resumes when both CTS is negated and Xon is detected.

Table 7–3. Receiver Flow-Control Possibilities

MODE		6	5	4
		DTR	RTS	RXOF
0	All flow control is disabled	0	0	0
1	Xon/Xoff flow control is enabled	0	0	1
2	$\overline{\text{RTS}}$ flow control	0	1	0
3	Combination flow control (see Note 3)	0	1	1
4	$\overline{\text{DTR}}$ flow control	1	0	0
5	Combination flow control	1	0	1
6	Combination flow control (see Note 4)	1	1	0
7	Combination flow control	1	1	1

NOTES: 3. Combination example: Both RTS is asserted and Xoff transmitted when FIFO is full. Both RTS is deasserted and Xon is transmitted when FIFO is empty.
4. Combination example: Both DTR and RTS are asserted when FIFO is full. Both DTR and RTS are deasserted when FIFO is empty.

7.1.6 MCR: Modem-Control Register (Addr:FFA4)

This register provides control for modem interface I/O and definition of the flow control mode.

7	6	5	4	3	2	1	0
LCD	**LRI**	**RTS**	**DTR**	**SEN**	**LOOP**	**RCVE**	**URST**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION	
0	URST	0	Uart soft reset. This bit can be used by the MCU to reset the UART.	
			URST = 0	Normal operation. Writing a 0 by MCU has no effect.
			URST = 1	When the MCU writes a 1 to this bit, a UART reset is generated (ORed with hard reset). When the UART exits the reset state, URST is cleared. The MCU can monitor this bit to determine if the UART completed the reset cycle.
1	RCVE	0	Receiver enable bit. This bit is valid only when 485E in FCRL is 1 (RS485 mode). When 485E = 0, this bit has no effect on the receiver.	
			RCVE = 0	When 485E = 1, the UART receiver is disabled when $\overline{RTS}$ = 1, i.e., when data is being transmitted, the UART receiver is disabled.
			RCVE = 1	When 485E = 1, the UART receiver is enabled regardless of the $\overline{RTS}$ state, i.e., UART receiver is enabled all the time. This mode can be used to detect collisions on the RS-485 bus when received data does not match transmitted data.
2	LOOP	0	This bit controls the normal-/loop-back mode of operation (see Figure 7–1).	
			LOOP = 0	Normal operation
			LOOP = 1	Enable loop-back mode of operation. In this mode the following occur: • SOUT is set high • SIN is disconnected from the receiver input. • The transmitter serial output is looped back into the receiver serial input. • The four modem-control inputs: $\overline{CTS}$, $\overline{DSR}$, $\overline{DCD}$, and $\overline{RI}$ are disconnected. • DTR, RTS, LRI and LCD are internally connected to the four modem-control inputs, and read in the MSR register as follows: • DTR is reflected in MSR[4] bit • RTS is reflected in MSR[5] bit • LRI is reflected in MSR[6] bit • LCD is reflected in MSR[7] bit
3	RSV	0	Reserved	
4	DTR	0	This bit controls the state of the $\overline{DTR}$ output pin (see Figure 7–1). This bit has no effect when auto-flow control is used or when 485E = 1 (in FCRL register).	
			DTR = 0	Forces the $\overline{DTR}$ output pin to inactive (high)
			DTR = 1	Forces the $\overline{DTR}$ output pin to active (low)
5	RTS	0	This bit controls the state of the $\overline{RTS}$ output pin (see Figure 7–1). This bit has no effect when auto-flow control is used or when 485E = 1 (in FCRL register).	
			RTS = 0	Forces the $\overline{RTS}$ output pin to inactive (high)
			RTS = 1	Forces the $\overline{RTS}$ output pin to active (low)

6	LRI	0	This bit is used for loop-back mode only. When in loop-back mode, this bit is reflected in MSR[6]-bit (see Figure 7–1).	
			LRI = 0	Clears MSR[6] = 0
			LRI = 1	Sets MSR[6] = 1
7	LCD	0	This bit is used for loop-back mode only. When in loop-back mode, this bit is reflected in MSR[7]-bit (see Figure 7–1).	
			LCD = 0	Clears MSR[7] = 0
			LCD = 1	Sets MSR[7] = 1

7.1.7 LSR: Line-status Register (Addr:FFA5)

This register provides the status of the data transfer. DMA transfer is halted when any of OVR, PTE, FRE, BRK, or EXIT is 1.

7	6	5	4	3	2	1	0
RSV	**TEMT**	**TxE**	**RxF**	**BRK**	**FRE**	**PTE**	**OVR**
R/O	R/O	R/O	R/O	R/C	R/C	R/C	R/C

BIT	NAME	RESET	FUNCTION
0	OVR	0	This bit indicates the overrun condition of the receiver. If set, it halts the DMA transfer and generates a status interrupt (if enabled).
			OVR = 0 No overrun error OVR = 1 Overrun error has occurred. Clears when the MCU writes a 1. Writing a 0 has no effect.
1	PTE	0	This bit indicates the parity condition of the received byte. If set, it halts the DMA transfer and generates a status interrupt (if enabled).
			PTE = 0 No parity error in data received PTE = 1 Parity error in data received. Clears when the MCU writes a 1. Writing a 0 has no effect.
2	FRE	0	This bit indicates the framing condition of the received byte. If set, it halts the DMA transfer and generates a status interrupt (if enabled).
			FRE = 0 No framing error in data received FRE = 1 Framing error in data received. Clears when MCU writes a 1. Writing a 0 has no effect.
3	BRK	0	This bit indicates the break condition of the received byte. If set, it halts the DMA transfer and generates a status interrupt (if enabled).
			BRK = 0 No break condition BRK = 1 A break condition in data received was detected. Clears when the MCU writes a 1. Writing a 0 has no effect.
4	RxF	0	This bit indicates the condition of the receiver data register. Typically, the MCU does not monitor this bit since data transfer is done by the DMA controller.
			RxF = 0 No data in the RDR RxF = 1 RDR contains data. Generates Rx interrupt (if enabled).
5	TxE	1	This bit indicates the condition of the transmitter data register. Typically, the MCU does not monitor this bit since data transfer is done by the DMA controller.
			TxE = 0 TDR is not empty TxE = 1 TDR is empty. Generates Tx interrupt (if enabled).
6	TEMT	1	This bit indicates the condition of both transmitter data register and shift register is empty.
			TEMT = 0 Either TDR or TSR is not empty TEMT = 1 Both TDR and TSR are empty
7	RSV	0	Reserved = 0

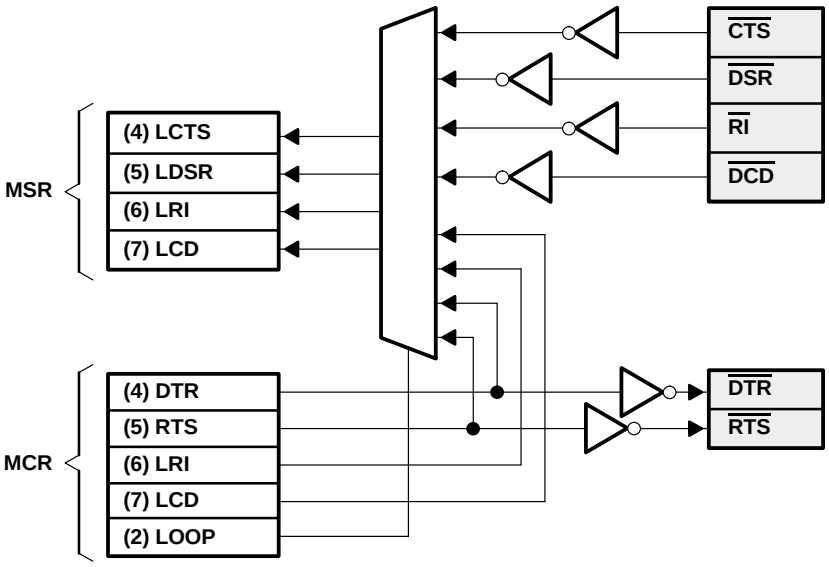


Figure 7–1. MSR and MCR Registers in Loop-Back Mode

7.1.8 MSR: Modem-Status Register (Addr:FFA6)

This register provides information about the current state of the control lines from the modem.

7	6	5	4	3	2	1	0
LCD	**LRI**	**LDSR**	**LCTS**	**ΔCD**	**TRI**	**ΔDSR**	**ΔCTS**
R/O	R/O	R/O	R/O	R/C	R/C	R/C	R/C

BIT	NAME	RESET	FUNCTION
0	ΔCTS	0	This bit indicates that the $\overline{\text{CTS}}$ input has changed state. Cleared when the MCU writes a 1 to this bit. Writing a 0 has no effect.
			ΔCTS = 0 Indicates no change in the $\overline{\text{CTS}}$ input ΔCTS = 1 Indicates that the $\overline{\text{CTS}}$ input has changed state since the last time it was read. Clears when the MCU writes a 1. Writing a 0 has no effect.
1	ΔDSR	0	This bit indicates that the $\overline{\text{DSR}}$ input has changed state. Cleared when the MCU writes a 1 to this bit. Writing a 0 has no effect.
			ΔDSR = 0 Indicates no change in the $\overline{\text{DSR}}$ input ΔDSR = 1 Indicates that the $\overline{\text{DSR}}$ input has changed state since the last time it was read. Clears when the MCU writes a 1. Writing a 0 has no effect.
2	TRI	0	Trailing edge of the ring indicator. This bit indicates that the $\overline{\text{RI}}$ input has changed from low to high. This bit is cleared when the MCU writes a 1 to this bit. Writing a 0 has no effect.
			TRI = 0 Indicates no applicable transition on the $\overline{\text{RI}}$ input TRI = 1 Indicates that an applicable transition has occurred on the $\overline{\text{RI}}$ input.
3	ΔCD	0	This bit indicates that the $\overline{\text{CD}}$ input has changed state. Cleared when the MCU writes a 1 to this bit. Writing a 0 has no effect.
			ΔCD = 0 Indicates no change in the $\overline{\text{CD}}$ input ΔCD = 1 Indicates that the $\overline{\text{CD}}$ input has changed state since the last time it was read.
4	LCTS	0	During loopback, this bit reflects the status of MCR[1] (see Figure 7–1)
			LCTS = 0 $\overline{\text{CTS}}$ input is high LCTS = 1 $\overline{\text{CTS}}$ input is low
5	LDSR	0	During loop back, this bit reflects the status of MCR[0] (see Figure 7–1).
			LDSR = 0 $\overline{\text{DSR}}$ input is high LDSR= 1 $\overline{\text{DSR}}$ input is low
6	LRI	0	During loop back, this bit reflects the status of MCR[2] (see Figure 7–1).
			LRI = 0 $\overline{\text{RI}}$ input is high LRI = 1 $\overline{\text{RI}}$ input is low

BIT	NAME	RESET	FUNCTION
7	LCD	0	During loopback, this bit reflects the status of MCR[3] (see Figure 7–1).
			LCD = 0 $\overline{CD}$ input is high LCD = 0 $\overline{CD}$ input is low

7.1.9 DLL: Divisor Register Low Byte (Addr:FFA7)

This register contains the low byte of the baud-rate divisor.

7	6	5	4	3	2	1	0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION
7–0	D[7:0]	08h	Low-byte value of the 16-bit divisor for generation of the baud clock in the baud-rate generator.

7.1.10 DLH: Divisor Register High Byte (Addr:FFA8)

This register contains the high byte of the baud-rate divisor.

7	6	5	4	3	2	1	0
D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION
7–0	D[15:8]	00h	High-byte value of the 16-bit divisor for generation of the baud clock in the baud-rate generator.

7.1.11 Baud-rate Calculation

The following formulas are used to calculate the baud-rate clock and the divisors. The baud-rate clock is derived from the 96-MHz master clock (dividing by 6.5). The table below presents the divisors used to achieve the desired baud rates, together with the associate rounding errors.

$$\text{Baud CLK} = \frac{96\ \text{MHz}}{6.5} = 14.76923077\ \text{MHz}$$

$$\text{Divisor} = \frac{14.76923077 \times 10^6}{\text{Baud Rate} \times 16}$$

Table 7–4. DLL/DLH Values and Resulted Baud Rates

DESIRED BAUD	DLL/DLH VALUE		ACTUAL BAUD	ERROR %
	DEC.	HEX.		
1 200	769	0301	1 200.36	0.03
2 400	385	0181	2 397.60	0.01
4 800	192	00C0	4 807.69	0.16
7 200	128	0080	7 211.54	0.16
9 600	96	0060	9 615.38	0.16
14 400	64	0040	14 423.08	0.16
19 200	48	0030	19 230.77	0.16
38 400	24	0018	38 461.54	0.16
57 600	16	0010	57 692.31	0.16
115 200	8	0008	115 384.62	0.16
230 400	4	0004	230 769.23	0.16
460 800	2	0002	461 538.46	0.16
921 600	1	0001	923 076.92	0.16

NOTE: The TUSB3410 does support baud rates lower than 1200 bps, which are not listed due to less interest.

7.1.12 XON: Xon Register (Addr:FFA9)

This register contains a value that is compared to the received data stream. Detection of a match interrupts the MCU (only if the interrupt enable bit is set). This value is also used for Xon transmission.

7	6	5	4	3	2	1	0
D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION
7–0	D[7:0]	0000	Xon value to be compared to the incoming data stream

7.1.13 XOFF: Xoff Register (Addr:FFAA)

This register contains a value that is compared to the received data stream. Detection of a match halts the DMA transfer, and interrupts the MCU (only if the interrupt enable bit is set). This value is also used for Xoff transmission.

7	6	5	4	3	2	1	0
D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION
7–0	D[7:0]	0000	Xoff value to be compared to the incoming data stream

7.1.14 MASK: UART Interrupt-Mask Register (Addr:FFAB)

This register controls the UARTs interrupt sources.

7	6	5	4	3	2	1	0
RSV	**RSV**	**RSV**	**RSV**	**RSV**	**RRIE**	**SIE**	**MIE**
R/O	R/O	R/O	R/O	R/O	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION
0	MIE	0	This bit controls the UART-modem interrupt.
			MIE = 0 Modem interrupt is disabled MIE = 1 Modem interrupt is enabled
1	SIE	0	This bit controls the UART-status interrupt.
			SIE = 0 Status interrupt is disabled MIE = 1 Status interrupt is enabled
2	TRI	0	This bit controls the UART-TxE/RxF interrupts
			TRIE = 0 TxE/RxF interrupts are disabled TRIE = 1 TxE/RxF interrupts are enable
7–3	RSV	0	Reserved = 0

7.2 UART Data Transfer

Figure 7–2 illustrates the data transfer between the UART and the host using the DMA controller and the USB buffer manager (UBM). A buffer of 512 bytes is reserved for buffering the UART channel (transmit and receive buffers). The UART channel has 64 bytes of double-buffer space (X- and Y-buffer). When the DMA writes to the X-buffer, the UBM reads from the Y-buffer. Similarly, when the DMA reads from the X-buffer, the UBM writes to the Y-buffer. The DMA channel is configured to operate in the continuous mode (by setting DMACDR[CNT] = 1). Once the MCU enables the DMA, data transfer toggles between the UMB and the DMA without MCU intervention. See *IN transaction (TUSB3410 to host)* for DMA transfer-termination condition.

7.2.1 Receiver Data Flow

The UART receiver has a 32-byte FIFO. The receiver FIFO has two trigger levels. One is the high-level mark (HALT), which is set to 12 bytes, and the other is the low-level mark (RESUME), which is set to 4 bytes. When the HALT mark is reached, either the $\overline{\text{RTS}}$ pin goes high or Xoff is transmitted (depending on the auto setting). When the FIFO reaches the RESUME mark, then either the $\overline{\text{RTS}}$ pin goes low or Xon is transmitted.

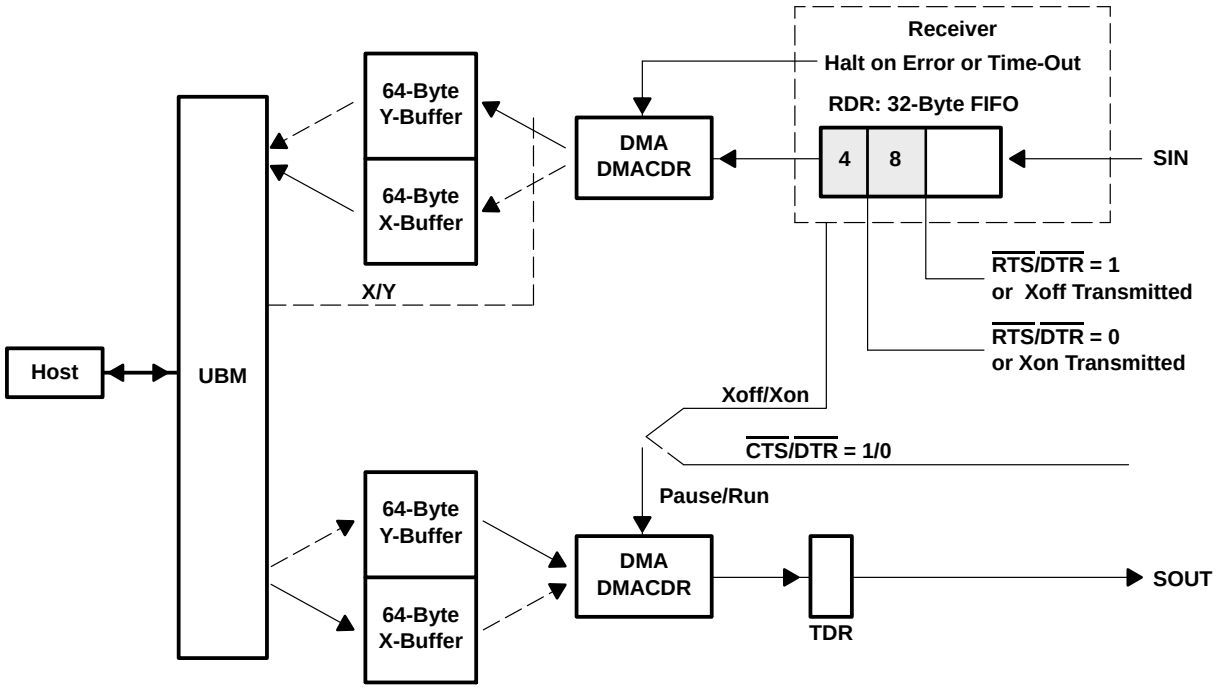


Figure 7–2. Receiver/Transmitter Data Flow

7.2.2 Hardware Flow Control

Figure 7–3 illustrates the connection necessary to achieve hardware flow control. The $\overline{\text{CTS}}$ and $\overline{\text{RTS}}$ signals are provided for this purpose. Auto $\overline{\text{CTS}}$ and auto $\overline{\text{RTS}}$ (and Xon/Xoff) can be enabled/disabled independently by programming the FCRL register.

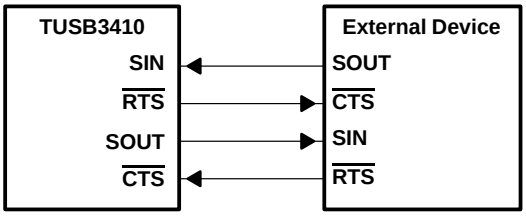


Figure 7–3. Auto Flow Control Interconnect

7.2.3 Auto $\overline{\text{RTS}}$ (Receiver Control)

In this mode, the $\overline{\text{RTS}}$ output pin signals the receiver-FIFO status to an external device. The $\overline{\text{RTS}}$ output signal is controlled by the high- and low-level marks of the FIFO. When the high-level mark is reached, $\overline{\text{RTS}}$ goes high, signaling to an external sending device to halt its transfer. Conversely, when the low-level mark is reached, $\overline{\text{RTS}}$ goes low, signaling to an external sending device to resume its transfer.

Data transfer from the FIFO to the X-/Y-buffer is performed by the DMA controller. See *OUT transaction (TUSB3410 to host)* for DMA transfer-termination condition.

7.2.4 Auto $\overline{\text{CTS}}$ (Transmitter Control)

In this mode, the $\overline{\text{CTS}}$ input pin controls the transfer from internal buffer (X or Y) to the TDR. When the DMA controller transfers data from the Y-buffer to the TDR and the $\overline{\text{CTS}}$ input pin goes high, the DMA controller is suspended until $\overline{\text{CTS}}$ goes low. Meanwhile, the UBM is transferring data from the host to the X-buffer. When $\overline{\text{CTS}}$ goes low, the DMA resumes the transfer. Data transfer continues alternating between the X- and Y-buffers, without MCU intervention. See *OUT transaction (TUSB3410 to host)* for DMA transfer-termination condition.

7.2.5 Xon/Xoff Receiver Flow Control

To enable Xon/Xoff flow control, certain MCR bits must be set as follows: MCR[5] = 1 and MCR[7:6] = 0. In this mode, the Xon/Xoff bytes are transmitted to an external sending device to control the device's transmission. When the high-level mark (of the FIFO) is reached, the Xoff byte is transmitted, signaling to an external sending device to halt its transfer. Conversely, when the low-level mark is reached, the Xon byte is transmitted, signaling to an external sending device to resume its transfer. The data transfer from the FIFO to X-/Y-buffer is performed by the DMA controller.

7.2.6 Xon/Xoff Transmit Flow Control

To enable Xon/Xoff flow control, certain MCR bits must be set as follows: MCR[5] = 1 and MCR[7:6] = 0. In this mode, the incoming data are compared to the XON and XOFF registers. If a match to XOFF is detected, the DMA is paused. If a match to XON is detected, the DMA resumes. Meanwhile, the UBM is transferring data from the host to the X-buffer. The MCU does not switch the buffers unless the Y-buffer is empty and the X-buffer is full. When Xon is detected, the DMA resumes the transfer.

8 Expanded GPIO Port

8.1 Input/Output and Control Registers

The TUSB3410 has four general-purpose I/O pins (P3.0, P3.1, P3.3, P3.4) that are controlled by firmware running on the MCU. Each pin can be controlled individually and each is implemented with a 12-mA push/pull Cmos output with tristate control plus input. The MCU treats the outputs as open drain types in that the output can be driven low continuously, but a high output is driven for two clock cycles and then the output is tristated.

An input pin can be read using the MOV instruction. For example, MOV C,P3.3 reads the input on P3.3. As a precaution, be certain the associated output is tristated before reading the input.

An output can be set high (and then tristated) using the SETB instruction. For example, SETB P3.1 sets P3.1 high. An output can be set low using the CLR instruction, as in CLR P3.4, which sets P3.4 low (driven continuously until changed).

Each GPIO pin has an associated internal pullup resistor. It is strongly recommended that the pullup resistor remain connected to the pin to prevent oscillations in the input buffer. The only exception is if an external source always drives the input.

8.1.1 PUR_3: GPIO Pullup Register For Port 3 (Addr:FF9E)

7	6	5	4	3	2	1	0
RSV	**RSV**	**RSV**	**RSV**	**Pin3**	**RSV**	**Pin1**	**Pin0**
R/O	R/O	R/O	R/W	R/W	R/O	R/W	R/W

BIT	NAME	RESET	FUNCTION
0–7	Pin N (N = 0 to 7)	0	The MCU may write to this register. If the MCU sets this bit to 1, the pullup resistor is disconnected from the pin. If the MCU clears this bit to 0, the pullup resistor is connected to the pin. The pullup resistor is connected to the V_{CC} power supply.

9 Interrupts

9.1 8052 Interrupt and Status Registers

All 8052 standard, five interrupt sources are preserved. SIE is the standard interrupt-enable register that controls the five interrupt sources. All the additional interrupt sources are ORed together to generate EX0. The $\overline{\text{XINTO}}$ signal is provided to interrupt an external MCU (see Figure 9–1).

Table 9–1. 8052 Interrupt Location Map

INTERRUPT SOURCE	DESCRIPTION	START ADDRESS	COMMENTS
ES	UART interrupt	0023H	
ET1	Timer-1 interrupt	001BH	
EX1	External interrupt-1	0013H	
ET0	Timer-0 interrupt	000BH	
EX0	External interrupt-0	0003H	Used for all internal peripherals
Reset		0000H	

9.1.1 8052 Standard Interrupt Enable (SIE) Register

7	6	5	4	3	2	1	0
EA	**X**	**X**	**ES**	**ET1**	**EX1**	**ET0**	**EX0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION
0	EX0	0	Enable or disable external interrupt-0 EX0 = 1 External interrupt-0 is disabled EX0 = 1 External interrupt-0 is enabled
1	ET0	0	Enable or disable timer-0 interrupt ET0 = 0 Timer-0 interrupt is disabled ET0 = 1 Timer-0 interrupt is enabled
2	EX1	0	Enable or disable external interrupt-1 EX1 = 0 External interrupt-1 is disabled EX1 = 1 External interrupt-1 is enabled
3	ET1	0	Enable or disable timer-1 interrupt ET1 = 0 Timer-1 interrupt is disabled EX1 = 1 Timer-1 interrupt is enabled
4	ES	0	Enable or disable serial port interrupts ES = 0 Serial-port interrupt is disabled ES = 1 Serial-port interrupt is enabled
5, 6	RSV	0	Reserved
7	EA	0	Enable or disable all interrupts (global disable) EA = 0 Disable all interrupts EA = 1 Each interrupt source is individually controlled

9.1.2 Additional Interrupt Sources

All nonstandard 8052 interrupts (DMA, I^2C, etc.) are ORed to generate an internal INT0. Note, the external INT0 is not used. Furthermore, the INT0 must be programmed as an active low-level interrupt (not edge triggered). A vector interrupt register is provided to identify all interrupt sources (see *VECINT: vector-interrupt register*). Up to 64 interrupt vectors are provided. It is the responsibility of the MCU to read the vector and dispatch to the proper interrupt routine.

9.1.3 VECINT: Vector Interrupt Register (Addr:FF92)

This register contains a vector value, which identifies the internal interrupt source that trapped to location 0003H. Writing (any value) to this register removes the vector and updates the next vector value (if another interrupt is pending). Note: the vector value is offset; therefore, its value is in increments of two (bit 0 is set to 0). When no interrupt is pending, the vector is set to 00h (see Table 9–2). As shown, the interrupt vector is divided to two fields: I[2:0] and G[3:0]. The I field defines the interrupt source within a group (on a first-come-first-served basis). In the G field, which defines the group number, group G0 is the lowest, and G15 is the highest priority.

7	6	5	4	3	2	1	0
G3	**G2**	**G1**	**G0**	**I2**	**I1**	**I0**	**0**
R/O	R/O	R/O	R/O	R/O	R/O	R/O	R/O

BIT	NAME	RESET	FUNCTION
3–1	I[2:0]	0H	This field defines the interrupt source in a given group. See Table 9–2. Bit 0 = 0 always; therefore, vector values are offset by two.
7–4	G[3:0]	0H	This field defines the interrupt group. I[2:0] and G[3:0] combine to produce the actual interrupt vector.

Table 9–2. Vector Interrupt Values

G[3:0] (Hex)	I[2:0] (Hex)	VECTOR (Hex)	INTERRUPT SOURCE
0	0	00	No interrupt
1	0	10	Not used
1	1	12	Output endpoint-1
1	2	14	Output endpoint-2
1	3	16	Output endpoint-3
2	0	20	Not used
2	1	22	Input endpoint-1
2	2	24	Input endpoint-2
2	3	26	Input endpoint-3
3	0	30	STPOW packet received
3	1	32	SETUP packet received
3	2	34	RESERVED
3	3	36	RESERVED
3	4	38	RESR interrupt
3	5	3A	SUSR interrupt
3	6	3C	RSTR interrupt
3	7	3E	Reserved
4	0	40	I^2C TXE interrupt
4	1	42	I^2C RXF interrupt
4	2	44	Input endpoint-0
4	3	46	Output endpoint-0
4	4–7	48 → 4E	Not used
5	0	50	UART status interrupt
5	1	52	UART modem interrupt
5	4–7	58 → 5E	Not used
6	0	60	UART RXF interrupt
6	1	62	UART TXE interrupt
6	4–7	68 → 6E	Not used
7	5–7	70 → 7E	Not used
8	0	80	DMA1 interrupt
8	2	84	DMA3 interrupt
8	5–7	88–8E	Not used
9–15	X	90 → FE	Not used

9.1.4 Logical Interrupt Connection Diagram (Internal/External)

Figure 9–1 shows the logical connection of the interrupt sources and its relation with $\overline{\text{XINTO}}$. The priority encoder generates an 8-bit vector, corresponding to 64 interrupt sources (not all are used). The interrupt priorities are hard wired. Vector 0x88 is the highest and 0x12 is the lowest.

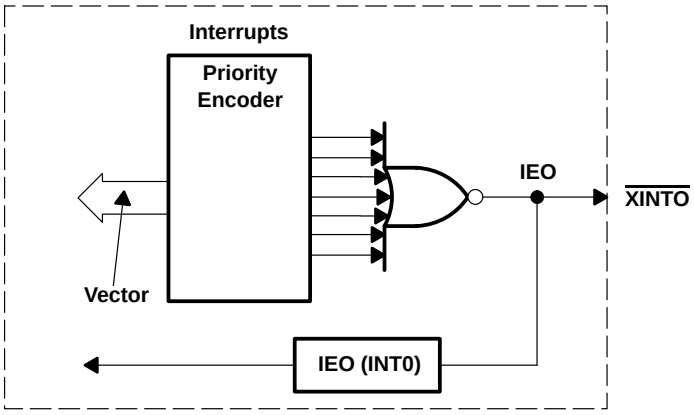


Figure 9–1. Internal Vector Interrupt

10 I²C-Port

10.1 I²C Registers

10.1.1 I2CSTA: I²C Status and Control Register (Addr:FFF0)

This register is used to control the stop condition for read and write operations. In addition, it provides transmitter and receiver handshake signals with their respective interrupt enable bits.

7	6	5	4	3	2	1	0
RXF	**RIE**	**ERR**	**1/4**	**TXE**	**TIE**	**SRD**	**SWR**
R/O	R/W	R/C	R/W	R/O	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION
0	SWR	0	Stop write condition. This bit determines if the I²C controller generates a stop condition when data from the I2CDAO register is transmitted to an external device. SWR = 0 Stop condition is not generated when data from the I2CDAO register is shifted out to an external device. SWR = 1 Stop condition is generated when data from the I2CDAO register is shifted out to an external device.
1	SRD	0	Stop read condition. This bit determines if the I²C controller generates a stop condition when data is received and loaded into the I2CDAI register. SRD = 0 Stop condition is not generated when data from the SDA line is shifted into the I2CDAI register. SRD = 1 Stop condition is generated when data from the SDA line are shifted into the I2CDAI register.
2	TIE	0	I²C transmitter empty interrupt enable TIE = 0 Interrupt disable TIE = 1 Interrupt enable
3	TXE	1	I²C transmitter empty. This bit indicates that data can be written to the transmitter. It can be used for polling or it can generate an interrupt. TXE = 0 Transmitter is full. This bit is cleared when the MCU writes a byte to the II2CDAO register. TXE = 1 Transmitter is empty. The I²C controller sets this bit when the contents of the I2CDAO register are copied to the SDA shift register.
4	1/4	0	Bus speed selection 1/4 = 0 100-kHz bus speed 1/4 = 1 400-kHz bus speed
5	ERR	0	Bus error condition. This bit is set by the hardware when the device does not respond. It is cleared by the MCU. ERR = 0 No bus error ERR = 1 Bus error condition has been detected. Clears when the MCU writes a 1. Writing a 0 has no effect.
6	RIE	0	I²C receiver ready interrupt enable RIE = 0 Interrupt disable RIE = 1 Interrupt enable
7	RXF	0	I²C receiver full. This bit indicates that the receiver contains new data. It can be used for polling or it can generate an interrupt. RXF = 0 Receiver is empty. This bit is cleared when the MCU reads the I2CDAI register. RXF = 1 Receiver contains new data. This bit is set by the I²C controller when the received serial data has been loaded into the I2CDAI register.

10.1.2 I2CADR: I²C Address Register (Addr:FFF3)

This register holds the device address and the read/write command bit.

7	6	5	4	3	2	1	0
A6	**A5**	**A4**	**A3**	**A2**	**A1**	**A0**	**R/W**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

BIT	NAME	RESET	FUNCTION
0	R/W	0	Read/write command bit
			R/W = 0 Write operation R/W = 1 Read operation
7–1	A[6:0]	0h	Seven address bits for device addressing

10.1.3 I2CDAI: I²C Data-Input Register (Addr:FFF2)

This register holds the received data from an external device.

7	6	5	4	3	2	1	0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/O	R/O	R/O	R/O	R/O	R/O	R/O	R/O

BIT	NAME	RESET	FUNCTION
7–0	D[7:0]	0	8-bit input data from an I²C device

10.1.4 I2CDAO: I²C Data-Output Register (Addr:FFF1)

This register holds the data to be transmitted to an external device. Writing to this register starts the transfer on the SDA line.

7	6	5	4	3	2	1	0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
W/O	W/O	W/O	W/O	W/O	W/O	W/O	W/O

BIT	NAME	RESET	FUNCTION
7–0	D[7:0]	0	8-bit output data to an I²C device

10.2 Random-Read Operation

A random read requires a dummy byte-write sequence to load in the data word address. Once the device-address word and the data-word address are clocked out and acknowledged by the device, the MCU starts a current-address sequence. The following describes the sequence of events to accomplish this transaction.

Device Address + EPROM [High Byte]

- The MCU sets I2CSTA[SRD] = 0. This forces the I²C controller not to generate a stop condition after the contents of the I2CDAI register are received.
- The MCU sets I2CSTA[SWR] = 0. This forces the I²C controller not to generate a stop condition after the contents of the I2CDAO register are transmitted.
- The MCU writes the device address (R/W bit = 0) to the I2CADR register (write operation)
- The MCU writes the high byte of the E2PROM address into the I2CDAO register (this starts the transfer on the SDA line).
- The TXE bit in the I2CSTA register is cleared (indicates busy).
- The content of the I2CADR register is transmitted to E2PROM (preceded by start condition on SDA).

- The contents of the I2CDAO register are transmitted to E2PROM. (EPROM address).
- The TXE bit in the I2CSTA register is set and interrupts the MCU, indicating that the I2CDAO register has been transmitted.
- A stop condition is not generated.

EPROM [Low Byte]

- The MCU writes the low byte of the E2PROM address into the I2CDAO register.
- The TXE bit in the I2CSTA register is cleared (indicates busy).
- The contents of the I2CDAO register are transmitted to the device (E2PROM address).
- The TXE bit in the I2CSTA register is set and interrupts the MCU, indicating that the I2CDAO register has been transmitted.
- This completes the dummy write operation. At this point, the E2ROM address is set and the MCU can do either a single- or a sequential-read operation.

10.3 Current-Address Read Operation

Once the E^2PROM address is set, the MCU can read a single byte by executing the following steps:

- The MCU sets I2CSTA[SRD] = 1. This forces the I^2C controller to generate a stop condition after the I2CDAI-register contents are received.
- The MCU writes the device address (R/W bit = 1) to the I2CADR register (read operation).
- The MCU writes a dummy byte to the I2CDAO register (this starts the transfer on SDA line).
- The RXF bit in the I2CSTA register is cleared.
- The contents of the I2CADR register are transmitted to the device (preceded by start condition on SDA).
- The data from E2PROM are latched into the I2CDAI register (stop condition is transmitted).
- The RXF bit in the I2CSTA register is set and interrupts the MCU, indicating that the data are available.
- The MCU reads the I2CDAI register. This clears the RXF bit (I2CSTA[RXF] = 0).
- End

10.4 Sequential-Read Operation

Once the E^2PROM address is set, the MCU can execute a sequential read operation by executing the following (this example illustrates a 32-byte sequential read):

Device Address

- The MCU sets I2CSTA[SRD] = 0. This forces the I^2C controller not to generate a stop condition after the I2CDAI register contents are received.
- The MCU writes the device address (R/W bit = 1) to the I2CADR register (read operation).
- The MCU writes a dummy byte to the I2CDAO register (this starts the transfer on the SDA line).
- The RXF bit in the I2CSTA register is cleared.
- The contents of the I2CADR register are transmitted to the device (preceded by start condition on SDA).

N-Byte Read (31 Bytes)

- The data from the device are latched into the I2CDAI register (stop condition is not transmitted).
- The RXF bit in the I2CSTA register is set and interrupts the MCU, indicating that data are available.
- The MCU reads the I2CDAI register. This clears the RXF bit (I2CSTA[RXF] = 0).
- This operation repeats 31 times.

Last-Byte Read (Byte 32)

- MCU sets I2CSTA[SRD] = 1. This forces the I^2C controller to generate a stop condition after the I2CDAI register contents are received.

- The data from the device is latched into the I2CDAI register (stop condition is transmitted).
- The RXF bit in the I2CSTA register is set and interrupts the MCU, indicating that data are available.
- The MCU reads the I2CDAI register. This clears the RXF bit (I2CSTA[RXF] = 0)
- End

10.5 Byte-Write Operation

The byte-write operation involves three phases: device address + EPROM [high byte] phase**,** EPROM [low byte] phase, and EPROM [DATA] phase. The following describes the sequence of events to accomplish the byte-write transaction.

Device Address + EPROM [High Byte]

- The MCU sets I2CSTA[SWR] = 0. This forces the I^2C controller to not generate a stop condition after the contents of the I2CDAO register are transmitted.
- The MCU writes the device address (R/W bit = 0) to the I2CADR register (write operation).
- The MCU writes the high byte of the E2PROM address into the I2CDAO register (this starts the transfer on the SDA line).
- The TXE bit in the I2CSTA register is cleared (indicates busy).
- The contents of the I2CADR register are transmitted to the device (preceded by start condition on SDA).
- The contents of the I2CDAO register are transmitted to the device (E2PROM high address).
- The TXE bit in the I2CSTA register is set and interrupts the MCU, indicating that the I2CDAO register contents have been transmitted.

EPROM [Low Byte]

- The MCU writes the low byte of the E2PROM address into the I2CDAO register.
- The TXE bit in the I2CSTA register is cleared (indicating busy).
- The contents of the I2CDAO register are transmitted to the device (E2PROM address).
- The TXE bit in the I2CSTA register is set and interrupts the MCU, indicating that the I2CDAO register contents have been transmitted.

EPROM [DATA]

- The MCU sets I2CSTA[SWR] = 1. This forces the I^2C controller to generate a stop condition after the contents of I2CDAO register are transmitted.
- The The data to be written to E2PROM is written by the MCU into the I2CDAO register.
- The TXE bit in the I2CSTA register is cleared (indicates busy).
- The contents of the I2CDAO register are transmitted to the device (E2PROM data).
- The TXE bit in the I2CSTA register is set and interrupts the MCU, indicating that the I2CDAO register contents have been transmitted.
- The I^2C controller generates a stop condition after the contents of the I2CDAO register are transmitted.
- End

10.6 Page-Write Operation

The page-write operation is initiated in the same way as byte write, with the exception that a stop condition is not generated after the first EPROM [DATA] is transmitted. The following describes the sequence of writing 32 bytes in page mode.

Device Address + EPROM [High Byte]

- The MCU sets I2CSTA[SWR] = 0. This forces the I^2C controller not to generate a stop condition after the contents of the I2CDAO register are transmitted.
- The MCU writes the device address (R/W bit = 0) to the I2CADR register (write operation).
- The MCU writes the high byte of the E2PROM address into the I2CDAO register
- The TXE bit in the I2CSTA register is cleared (indicating busy).
- The contents of the I2CADR register are transmitted to the device (preceded by start condition on SDA).
- The contents of the I2CDAO register are transmitted to the device (E2PROM address).
- The TXE bit in the I2CSTA register is set and interrupts the MCU, indicating that the I2CDAO register contents have been transmitted.

EPROM [Low Byte]

- The MCU writes the low byte of the E2PROM address into the I2CDAO register.
- The TXE bit in the I2CSTA register is cleared (indicates busy).
- The contents of the I2CDAO register are transmitted to the device (E2PROM address).
- The TXE bit in the I2CSTA register is set and interrupts the MCU, indicating that the I2CDAO register contents have been transmitted.

EPROM [DATA]—31 Bytes

- The data to be written to the E2PROM are written by the MCU into the I2CDAO register.
- The TXE bit in the I2CSTA register is cleared (indicates busy).
- The contents of the I2CDAO register are transmitted to the device (E2PROM data).
- The TXE bit in the I2CSTA register is set and interrupts the MCU, indicating that the I2CDAO register contents have been transmitted.
- This operation repeats 31 times.

EPROM [DATA]—Last Byte

- The MCU sets I2CSTA[SWR] = 1. This forces the I^2C controller to generate a stop condition after the contents of the I2CDAO register are transmitted.
- The MCU writes the last date byte to be written to the E2PROM, into the I2CDAO register.
- The TXE bit in the I2CSTA register is cleared (indicates busy).
- The contents of the I2CDAO register are transmitted to E2PROM (E2PROM data).
- The TXE bit in the I2CSTA register is set and interrupts the MCU, indicating that the I2CDAO register contents have been transmitted.
- The I^2C controller generates a stop condition after the contents of the I2CDAO register are transmitted.
- End of 32-byte page-write operation.

11 TUSB3410 Bootcode Flow

11.1 Introduction

TUSB3410 bootcode is a program embedded within TUSB3410 device. This program is designed to load application firmware from either external memory device or USB host bootloader device driver. After finished downloading, bootcode releases its control to the application firmware.

This document describes how the bootcode initializes the TUSB3410 device in detail. In addition, the default USB descriptor, I^2C device header format, USB host driver firmware downloading format, and supported built-in USB vendor specific requests are listed for reference. Users should carefully follow the appropriate format to interface with the bootcode. All unsupported formats might cause unexpected results.

The bootcode source code is also provided for programming reference.

11.2 Bootcode Programming Flow

After power-on reset, the bootcode initializes the I^2C and USB registers along with internal variables. The bootcode then checks to see if the I^2C device contains a valid signature. If the I^2C device has a valid signature, the bootcode continues searching for descriptor blocks and then processes them if the checksum is correct. If application firmware was found, the bootcode downloads it and releases the control to the application firmware. Otherwise, the bootcode connects to the USB and waits for host driver to download application firmware. Once firmware downloading is finished, the bootcode releases the control to the firmware.

The following is the bootcode step-by-step operation.

- Check if bootcode is in the application mode. If the bootcode is in the application mode, the bootcode releases the control to the application firmware. Otherwise, the bootcode continues.
- Initialize all the default settings.
  - Call CopyDefaultSettings() routine.

    Set I^2C to 400-kHz speed.
  - Call UsbDataInitialization() routine.

    Set bFUNADR = 0
    Disconnect from USB (bUSBCTL = 0x00)
    Bootcode handles USB reset
    Copy predefined device, configuration, and string descriptors to RAM
    Disable all endpoints and enable USB interrupt(SETUP, RSTR, SUSPR, and RESU)
- Search for product signature
  - Check if valid signature is in I^2C. If not, skip I^2C process.

    Read 2 bytes from address 0x0000 with type III and device address 0. Stop searching if valid signature is found.

    Read 2 bytes from address 0x0000 with type II and device address 4. Stop searching if valid signature is found.
- Load customized device, configuration and string descriptors from I^2C EEPROM.
  - Process each descriptor block from I^2C until *end of header* is found

    If descriptor block is device, configuration or string descriptors, the bootcode overwrites the default descriptors.

    If descriptor block is binary firmware, the bootcode makes a note and loads the firmware later on.

If descriptor block is auto-execution firmware, the bootcode loads it and releases the control to the firmware.

If descriptor block is *end of header*, the bootcode stops searching.

- Set header pointer to the beginning of the binary firmware in I^2C EEPROM.
- Enable global and USB interrupts and set connection bit to 1.
  - Set global interrupt bit. EA = 1.
  - Set internal interrupt bit. EX0 = 1.
  - Set connection bit. CONT = 1.
- Wait for any interrupt events until Get DEVICE DESCIPTOR setup packet arrives.
  - Suspend interrupt

    Set IDLE = 1 to enter suspend mode. USB reset wakes up the microcontroller.
  - Resume interrupt

    Bootcode wakes up and waits for new USB requests.
  - Reset interrupt

    Call UsbReset() routine.
  - Setup interrupt

    Bootcode process the request.
  - Reboot

    If Reboot=1, disconnect from USB and restart at address 0x0000.
- Download firmware from I^2C EEPROM
  - Disable global interrupt. Reset EA = 0.
  - Load firmware to xdata space if available.
- Download firmware from USB.
  - If no firmware in I^2C EEPROM, host downloads firmware via output endpoint 1.
  - In the first data packet to output endpoint 1, host driver add 3 bytes before the application firmware in binary format. These three bytes are LSB and MSB of firmware size and then arithmetic checksum of binary firmware.
- Release control to firmware.
  - Update USB configuration and interface number.
  - Release control to application firmware.
- Application firmware
  - Either disconnect from bus or continue responding to USB requests.

11.3 Default Bootcode Settings

The bootcode has its own predefined device, configuration, and string descriptors. These default descriptors should be used in evaluation only. They should not be used in end-user product.

11.3.1 Device Descriptor

Device descriptor describes the USB version that the device supports, device class, protocol, vendor, product identifications, strings, and number of configuration. The OS (operation system like Windows, MAC, or Linux) reads this descriptor to decide which device driver should be used to communicate to this device.

The bootcode uses 0x0451(Texas Instruments) as vendor ID and 0x3410(TUSB3410) as product ID. It also supports three different strings and one configuration. Table 11–1 lists the device descriptor.

Table 11–1. Device Descriptor

OFFSET	FIELD	SIZE	VALUE	DESCRIPTION
0	bLength	1	0x12	Size of this descriptor in bytes
1	bDescriptorType	1	1	Device Descriptor type
2	bcdUSB	2	0x0110	USB spec 1.1
4	bDeviceClass	1	0xFF	Device class is vendor–specific
5	bDeviceSubClass	1	0	We have no subclasses.
6	bDeviceProtocol	1	0	We use no protocols.
7	bMaxPacketSize0	1	8	Max. packet size for endpoint zero
8	idVendor	2	0x0451	USB–assigned vendor ID = TI
10	idProduct	2	0x3410	TI part number = TUSB3410
12	bcdDevice	2	0x100	Device release number = 1.0
14	iManufacturer	1	1	Index of string descriptor describing manufacturer
15	iProducct	1	2	Index of string descriptor describing product
16	iSerialNumber	1	3	Index of string descriptor describing device's serial number
17	bNumConfigurations	1	1	Number of possible configurations:

11.3.2 Configuration Descriptor

The configuration descriptor describes the number of interfaces supported by this configuration, power configuration, and current consumption.

The bootcode declares only one interface running in bus-powered mode. It consumes up to 100 mA at boot time. Table 11–2 lists the configuration descriptor.

Table 11–2. Configuration Descriptor

OFFSET	FIELD	SIZE	VALUE	DESCRIPTION
0	bLength	1	9	Size of this descriptor in bytes.
1	bDescriptor Type	1	2	Configuration descriptor type
2	wTotalLength	2	25 = 9 + 9 + 7	Total length of data returned for this configuration. Includes the combined length of all descriptors (configuration, interface, endpoint, and class- or vendor-specific) returned for this configuration.
4	bNumInterfaces	1	1	Number of interfaces supported by this configuration
5	bConfigurationValue	1	1	Value to use as an argument to the SetConfiguration() request to select this configuration.
6	iConfiguration	1	0	Index of string descriptor describing this configuration.
7	bmAttributes	1	0x80	Configuration characteristics D7: Reserved (set to one) D6: Self-powered D5: Remote wakeup is supported D4–0: Reserved (reset to zero)
8	bMaxPower	1	0x32	This device consumes 100 mA.

11.3.3 Interface Descriptor

The interface descriptor describes the number of endpoints supported by this interface as well as interface class, subclass, and protocol.

The bootcode supports only one endpoint and use its own class. Table 11–3 lists the interface descriptor.

Table 11–3. Interface Descriptor

OFFSET	FIELD	SIZE	VALUE	DESCRIPTION
0	bLength	1	9	Size of this descriptor in bytes
1	bDescriptorType	1	4	Interface descriptor type
2	bInterfaceNumber	1	0	Number of interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.
3	bAlternateSetting	1	0	Value used to select alternate setting for the interface identified in the prior field
4	bNumEndpoints	1	1	Number of endpoints used by this interface (excluding endpoint zero). If this value is zero, this interface only uses the default control pipe.
5	bInterfaceClass	1	0xFF	The interface class is vendor specific.
6	bInterfaceSubClass	1	0	
7	bInterfaceProtocol	1	0	
8	iInterface	1	0	Index of string descriptor describing this interface

11.3.4 Endpoint Descriptor

The endpoint descriptor describes the type and size of communication pipe supported by this endpoint.

The bootcode supports only one output endpoint with the size of 64 bytes in addition to control endpoint 0 (required by all USB devices). Table 11–4 lists the endpoint descriptor.

Table 11–4. Output Endpoint1 Descriptor

OFFSET	FIELD	SIZE	VALUE	DESCRIPTION
0	bLength	1	7	Size of this descriptor in bytes
1	bDescriptorType	1	5	Endpoint descriptor type
2	bEndpointAddress	1	0x01	Bit 3…0: The endpoint number Bit 7: Direction 0 = OUT endpoint 1 = IN endpoint
3	bmAttributes	1	2	Bit 1…0: Transfer type 10 = Bulk 11 = Interrupt
4	wMaxPacketSize	2	64	Maximum packet size this endpoint is capable of sending or receiving when this configuration is selected.
6	bInterval	1	0	Interval for polling endpoint for data transfers. Expressed in milliseconds.

11.3.5 String Descriptor

The string descriptor contains string in the unicode format. It is used to show the manufacturers name, product model, and serial number in human readable format.

The bootcode supports three strings. The first string is the manufacturers name, the second string is the product name, and the last string is the serial number. Table 11–5 lists the string descriptor.

Table 11–5. String Descriptor

OFFSET	FIELD	SIZE	VALUE	DESCRIPTION
0	bLength	1	4	Size of string 0 descriptor in bytes
1	bDescriptorType	1	0x03	String descriptor type
2	wLANGID[0]	2	0x0409	English
4	bLength	1	36	Size of string 1 descriptor in bytes
5	bDescriptorType	1	0x03	String descriptor type
6	bString	2	‘T’,0x00	Unicode, T is the first byte
8		2	‘e’,0x00	Texas Instruments
10		2	‘x’,0x00	
12		2	‘a’,0x00	
14		2	‘s’,0x00	
16		2	‘ ’,0x00	
18		2	‘I’,0x00	
20		2	‘n’,0x00	
22		2	‘s’,0x00	
24		2	‘t’,0x00	
26		2	‘r’,0x00	
28		2	‘u’,0x00	
30		2	‘m’,0x00	
32		2	‘e’,0x00	
34		2	‘n’,0x00	
36		2	‘t’,0x00	
38		2	‘s’,0x00	
40	bLength	1	42	Size of string 2 descriptor in bytes
41	bDescriptorType	1	0x03	STRING descriptor type
42	bString	2	‘T’,0x00	UNICODE, T is first byte
44		2	‘U’,0x00	TUSB3410 boot device
46		2	‘S’,0x00	
48		2	‘B’,0x00	
50		2	‘3’,0x00	
52		2	‘4’,0x00	
54		2	‘1’,0x00	
56		2	‘0’,0x00	
58		2	‘ ‘,0x00	
60		2	‘B‘,0x00	
62		2	‘o’,0x00	
64		2	‘o’,0x00	
66		2	‘t’,0x00	
68		2	‘ ’,0x00	
70		2	‘D’,0x00	

Table 11–5. String Descriptor (Continued)

OFFSET	FIELD	SIZE	VALUE	DESCRIPTION
72		2	‘e‘,0x00	
74		2	‘v’,0x00	
76		2	‘I,0x00	
78		2	‘c’,0x00	
80		2	‘e’,0x00	
82	bLength	1	34	Size of string 3 descriptor in bytes
84	bDescriptorType	1	0x03	STRING descriptor type
86	bString	2	r0,0x00	UNICODE
88		2	r1,0x00	R0 to rF are BCD of SERNUM0 to
90		2	r2,0x00	SERNUM7 registers. 16 digit hex
92		2	r3,0x00	16 digit hex numbers are created from
94		2	r4,0x00	SERNUM0 to SERNUM7 registers
96		2	r5,0x00	
98		2	r6,0x00	
100		2	r7,0x00	
102		2	r8,0x00	
104		2	r9,0x00	
106		2	rA,0x00	
108		2	rB,0x00	
110		2	rC,0x00	
112		2	rD,0x00	
114		2	rE,0x00	
116		2	rF,0x00	

11.4 External Device Header Format

The header can be restored in various storage devices such as ROM, parallel/serial EEPROM, I^2C, or flash ROM. A valid header should contain a product signature and one or more descriptor blocks. The descriptor block contains the descriptor prefix and content. In the descriptor prefix, the data type, size, and checksum are specified to describe the content. The descriptor content contains the necessary information for the bootcode to process.

The header processing routine always counts from the first descriptor block until the desired block number is reached. The header reads in descriptor prefix with the size of 4 bytes. This prefix contains the type of block, size, and checksum. For example, if the bootcode would like to find the position on third descriptor block, it reads in the first descriptor prefix, calculates the position on the second descriptor prefix based on the size specified in the prefix. bootcode, then repeats the same calculation to find out the position of the third descriptor block.

Note that the header-processing routine of the TUSB3410 only supports the I^2C device. No other storage device should be used to store header information.

11.4.1 Product Signature

The product signature should be stored at the first 2 bytes of storage device. These 2 bytes should match the product number. The order of these 2 bytes should be the LSB first and then the MSB. For example, UMP (TUSB5152) is 0x5152. Therefore, the first byte should be 0x52 and the second byte should be 0x51.

The TUSB3410 bootcode searches the first 2 bytes of the I^2C device. If the first 2 bytes are not 0x10 and 0x34, the bootcode skips the header processing.

11.4.2 Descriptor Block

Each descriptor block contains prefix and content. The size of the prefix is always 4 bytes. It contains the data type, size, and checksum for data integrity. The descriptor content contains the corresponding information specified in the prefix. It could be as small as 1 byte or as large as 65535 bytes. The next descriptor immediately follows the previous descriptor. If there are no more descriptors, an extra byte with a value of zero should be added to indicate the end of header.

11.4.2.1 Descriptor Prefix

The first byte of the descriptor prefix is the data type. This tells the bootcode how to process the data in the descriptor content. The second and third bytes are the size of descriptor content. The second byte is the low byte of the size and the third byte is the high byte. The last byte is the 8-bit arithmetic checksum of descriptor content.

11.4.2.2 Descriptor Content

Information stored in the descriptor content can be the USB information, firmware, or other type of data. The size of the content should be from 1 byte to 65535 bytes.

11.5 Checksum in Descriptor Block

Each descriptor prefix contains one checksum of the descriptor content. If the checksum is wrong, the bootcode simply ignores the descriptor block.

11.6 Header Examples

The header can be specified in different ways. The following descriptors show examples of the header format and the supported descriptor block.

11.6.1 TUSB3410 Bootcode Supported Descriptor Block

The TUSB3410 bootcode supports the following descriptor blocks.

- USB Device Descriptor
- USB Configuration Descriptor
- USB String Descriptor
- Binary Firmware[1]
- Autoexec Binary Firmware[2]

[1] Binary firmware is loaded when the bootcode receives the first *get device descriptor request* from host. Downloading the firmware should either continue that request in the data stage or disconnect from the USB and then reconnect to the USB as a new device.

[2] The bootcode loads this autoexec binary firmware before it connects to the USB. The firmware should connect to the USB once it is loaded.

11.6.2 USB Descriptor Header

Table 11–6 contains the USB device, configuration, and string descriptors for the bootcode. The last byte is zero to indicate the end of header.

Table 11–6. USB Descriptors Header

OFFSET	TYPE	SIZE	VALUE	DESCRIPTION
0	Signature0	1	0x10	FUNCTION_PID_L
1	Signature1	1	0x34	FUNCTION_PID_H
2	Data Type	1	0x03	USB device descriptor
3	Data Size (low byte)	1	0x12	The device descriptor is 18 bytes.
4	Data Size (high byte)	1	0x00	
5	Check Sum	1	0xCC	Checksum of data below
6	bLength	1	0x12	Size of device descriptor in bytes
7	bDescriptorType	1	0x01	Device descriptor type
8	bcdUSB	2	0x0110	USB spec 1.1
10	bDeviceClass	1	0xFF	Device class is vendor-specific
11	bDeviceSubClass	1	0x00	We have no subclasses.
12	bDeviceProtocol	1	0x00	We use no protocols
13	bMaxPacketSize0	1	0x08	Maximum packet size for endpoint zero
14	idVendor	2	0x0451	USB–assigned vendor ID = TI
16	idProduct	2	0x3410	TI part number = TUSB3410
18	bcdDevice	2	0x0100	Device release number = 1.0
20	iManufacturer	1	0x01	Index of string descriptor describing manufacturer
21	iProducct	1	0x02	Index of string descriptor describing product
22	iSerialNumber	1	0x03	Index of string descriptor describing device's serial number
23	bNumConfigurations	1	0x01	Number of possible configurations:
24	Data Type	1	0x04	USB configuration descriptor
25	Data Size (low byte)	1	0x19	25 bytes
26	Data Size (high byte)	1	0x00	
27	Check Sum	1	0xC6	Checksum of data below
28	bLength	1	0x09	Size of this descriptor in bytes
29	bDescriptorType	1	0x02	CONFIGURATION Descriptor type
30	wTotalLength	2	25(0x19) = 9 + 9 + 7	Total length of data returned for this configuration. Includes the combined length of all descriptors (configuration, interface, endpoint, and class- or vendor-specific) returned for this configuration.
32	bNumInterfaces	1	0x01	Number of interfaces supported by this configuration
33	bConfigurationValue	1	0x01	Value to use as an argument to the SetConfiguration() request to select this configuration
34	iConfiguration	1	0x00	Index of string descriptor describing this configuration.
35	bmAttributes	1	0xE0	Configuration characteristics D7: Reserved (set to one) D6: Self–powered D5: Remote Wakeup is supported D4–0: Reserved (reset to zero)
36	bMaxPower	1	0x64	This device consumes 100 mA.
37	bLength	1	0x09	Size of this descriptor in bytes
38	bDescriptorType	1	0x04	INTERFACE descriptor type
39	bInterfaceNumber	1	0x00	Number of interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.

Table 11–6. USB Descriptors Header (Continued)

OFFSET	TYPE	SIZE	VALUE	DESCRIPTION
40	bAlternateSetting	1	0x00	Value used to select alternate setting for the interface identified in the prior field
41	bNumEndpoints	1	0x01	Number of endpoints used by this interface (excluding endpoint zero). If this value is zero, this interface only uses the default control pipe.
42	bInterfaceClass	1	0xFF	The interface class is vendor specific.
43	bInterfaceSubClass	1	0x00	
44	bInterfaceProtocol	1	0x00	
45	iInterface	1	0x00	Index of string descriptor describing this interface
46	bLength	1	0x07	Size of this descriptor in bytes
47	bDescriptorType	1	0x05	ENDPOINT descriptor type
48	bEndpointAddress	1	0x01	Bit 3…0: The endpoint number Bit 7: Direction 0 = OUT endpoint 1 = IN endpoint
49	bmAttributes	1	0x02	Bit 1…0: Transfer Type 10 = Bulk 11 = Interrupt
50	wMaxPacketSize	2	0x0040	Maximum packet size this endpoint is capable of sending or receiving when this configuration is selected.
52	bInterval	1	0x00	Interval for polling endpoint for data transfers. Expressed in milliseconds.
53	Data Type	1	0x05	USB String descriptor
54	Data Size (low byte)	1	0x1A	26(0x1A) = 4 + 6 + 6 + 10
55	Data Size (high byte)	1	0x00	
56	Check Sum	1	0x50	Checksum of data below
57	bLength	1	0x04	Size of string 0 descriptor in bytes
58	bDescriptorType	1	0x03	STRING descriptor type
59	wLANGID[0]	2	0x0409	English
61	bLength	1	0x06	Size of string 1 descriptor in bytes
62	bDescriptorType	1	0x03	STRING descriptor type
63	bString	2	‘T’,0x00	UNICODE, ‘T’ is the first byte.
65		2	‘I’,0x00	TI = 0x54, 0x49
67	bLength	1	0x06	Size of string 2 descriptor in bytes
68	bDescriptorType	1	0x03	STRING descriptor type
69	bString	2	‘u’,0x00	UNICODE, ‘u’ is the first byte.
71		2	‘C’,0x00	‘uC’ = 0x75, 0x43
73	bLength	1	0x0A	Size of string 3 descriptor in bytes
74	bDescriptorType	1	0x03	STRING descriptor type
75	bString	2	‘3’,0x00	UNICODE, ‘T’ is the first byte.
77		2	‘4’,0x00	‘3410’ = 0x33, 0x34, 0x31, 0x30
79		2	‘1’,0x00	
81		2	‘0’,0x00	
83	Data Type	1	0x00	End of header

11.6.3 Autoexec Binary Firmware

If the application requires firmware loaded prior to USB connection, the following header can be used. The bootcode loads the firmware and release the control to the firmware directly without connecting to the USB. However, per the USB specification requirement, any USB device should connect to the bus and respond to the host within the first 100 ms. Therefore, if downloading time is more than 100 ms, the USB and header speed descriptor blocks should be added before the autoexec binary firmware. Table 11–7 shows an example of autoexec binary firmware header.

Table 11–7. Autoexec Binary Firmware

OFFSET	TYPE	SIZE	VALUE	DESCRIPTION
0x0000	Signature0	1	0x10	FUNCTION_PID_L
0x0001	Signature1	1	0x34	FUNCTION_PID_H
0x0002	Data Type	1	0x07	Autoexec binary firmware
0x0003	Data Size (low byte)	1	0x67	0x4567 bytes of application code
0x0004	Data Size (high byte)	1	0x45	
0x0005	Check Sum	1	0xNN	Checksum of the following firmware
0x0006	Program	0x4567		Binary application code
0x456d	Data Type	1	0x00	End of header

11.7 Host Driver Downloading Header Format

If firmware downloading from the host driver is desired, the host driver should follow the format in Table 11–8. The Texas Instruments bootloader driver generates the proper format. Therefore, users only need to provide the binary image of the application firmware for the Bootloader. If the checksum is wrong, the bootcode disconnects from the USB and waits before it reconnects to the USB.

Table 11–8. Host Driver Downloading Format

OFFSET	TYPE	SIZE	VALUE	DESCRIPTION
0x0000	Firmware size (low byte)	1	0xXX	Application firmware size
0x0001	Firmware size (low byte)	1	0xYY	
0x0002	Checksum	1	0xZZ	Checksum of binary application code
0x0003	Program	0xYYXX		Binary application code

11.8 Built-In Vendor Specific USB Requests

The bootcode supports several vendor specific USB requests. These requests are primarily for internal testing only. These functions should not be used in normal operation.

11.8.1 Reboot

The reboot command forces the bootcode to reboot. The bootcode starts over.

bmRequestType	USB_REQ_TYPE_DEVICE | USB_REQ_TYPE_VENDOR | USB_REQ_TYPE_OUT	01000000b
bRequest	BTC_REBOOT	0x85
wValue	None	0x0000
wIndex	None	0x0000
wLength	None	0x0000
Data	None	

11.8.2 Force Execute Firmware

The force execute firmware command requests the bootcode to execute the downloaded firmware unconditionally.

bmRequestType	USB_REQ_TYPE_DEVICE | USB_REQ_TYPE_VENDOR | USB_REQ_TYPE_OUT	01000000b
bRequest	BTC_FORCE_EXECUTE_FIRMWARE	0x8F
wValue	None	0x0000
wIndex	None	0x0000
wLength	None	0x0000
Data	None	

11.8.3 External Memory Read

The bootcode returns the content of the specified address.

bmRequestType	USB_REQ_TYPE_DEVICE | USB_REQ_TYPE_VENDOR | USB_REQ_TYPE_IN	11000000b
bRequest	BTC_EXETERNAL_MEMORY_READ	0x90
wValue	None	0x0000
wIndex	Data address	0xNNNN (From 0x0000 to 0xFFFF)
wLength	1 byte	0x0001
Data	Byte in the specified address	0xNN

11.8.4 External Memory Write

The external memory write command tells the bootcode to write data to the specified address.

bmRequestType	USB_REQ_TYPE_DEVICE | USB_REQ_TYPE_VENDOR | USB_REQ_TYPE_OUT	01000000b
bRequest	BTC_EXETERNAL_MEMORY_WRITE	0x91
wValue	HI: 0x00 LO: Data	0x00NN
wIndex	Data address	0xNNNN (From 0x0000 to 0xFFFF)
wLength	None	0x0000
Data	None	

11.8.5 I2C Memory Read

The bootcode returns the content of the specified address in I2C EEPROM.

In the wValue field, the I2C device number is from 0x00 to 0x07 in high filed. The memory type is from 0x01 to 0x03 for CAT I to CAT III devices. If bit 7 of bValueL is set, then 400 kHz is used. Otherwise, 100 kHz is used. This request is also used to set the device number and speed before the I2C write request.

bmRequestType	USB_REQ_TYPE_DEVICE | USB_REQ_TYPE_VENDOR | USB_REQ_TYPE_IN	11000000b
bRequest	BTC_I2C_MEMORY_READ	0x92
wValue	HI: I2C device number LO: Memory type bit[1:0] Speed bit[7]	0xXXYY
wIndex	Data address	0xNNNN (From 0x0000 to 0xFFFF)
wLength	1 byte	0x0001
Data	Byte in the specified address	0xNN

11.8.6 I2C Memory Write

The I2C memory write command tells the bootcode to write data to the specified address. The SPI mode setting is done in the SPI read command.

bmRequestType	USB_REQ_TYPE_DEVICE | USB_REQ_TYPE_VENDOR | USB_REQ_TYPE_OUT	01000000b
bRequest	BTC_I2C_MEMORY_WRITE	0x93
wValue	HI: should be zero LO: Data	0x00NN
wIndex	Data address	0xNNNN (From 0x0000 to 0xFFFF)
wLength	None	0x0000
Data	None	

11.8.7 Internal ROM Memory Read

The bootcode returns the byte of the specified address in ROM. That is, the binary code of the bootcode.

bmRequestType	USB_REQ_TYPE_DEVICE | USB_REQ_TYPE_VENDOR | USB_REQ_TYPE_OUT	01000000b
bRequest	BTC_INTERNAL_ROM_MEMORY_READ	0x94
wValue	None	0x0000
wIndex	Data address	0xNNNN (From 0x0000 to 0xFFFF)
wLength	1 byte	0x0001
Data	Byte in the specified address	0xNN

11.9 Bootcode Programming Consideration

11.9.1 USB Requests

For each USB request, the bootcode follows the steps below to ensure proper operation of the hardware.

1. Determine the direction of the request by checking the MSB of the bmRequestType field and set the USBCTL_DIR bit accordingly.
2. Decode the command
3. If another setup is pending, then return. Otherwise, serve the request.
4. Check again, if another setup is pending then go to step 2.
5. Clear the interrupt source and then the VECINT register.
6. Exit the interrupt routine.

11.9.1.1 USB Requests

The USB request consist of three types of transfers. They are control-read-with-data-stage, control-write-without-data-stage, and control-write-with-data-stage transfer. In each transfer, arrows indicate interrupts generated after receiving the setup packet, in or out token.

Figure 11–1 and Figure 11–2 show the USB data flow and how the hardware and firmware respond to the USB requests. Table 11–9 and Table 11–10 lists the bootcode reposes to the standard USB requests.

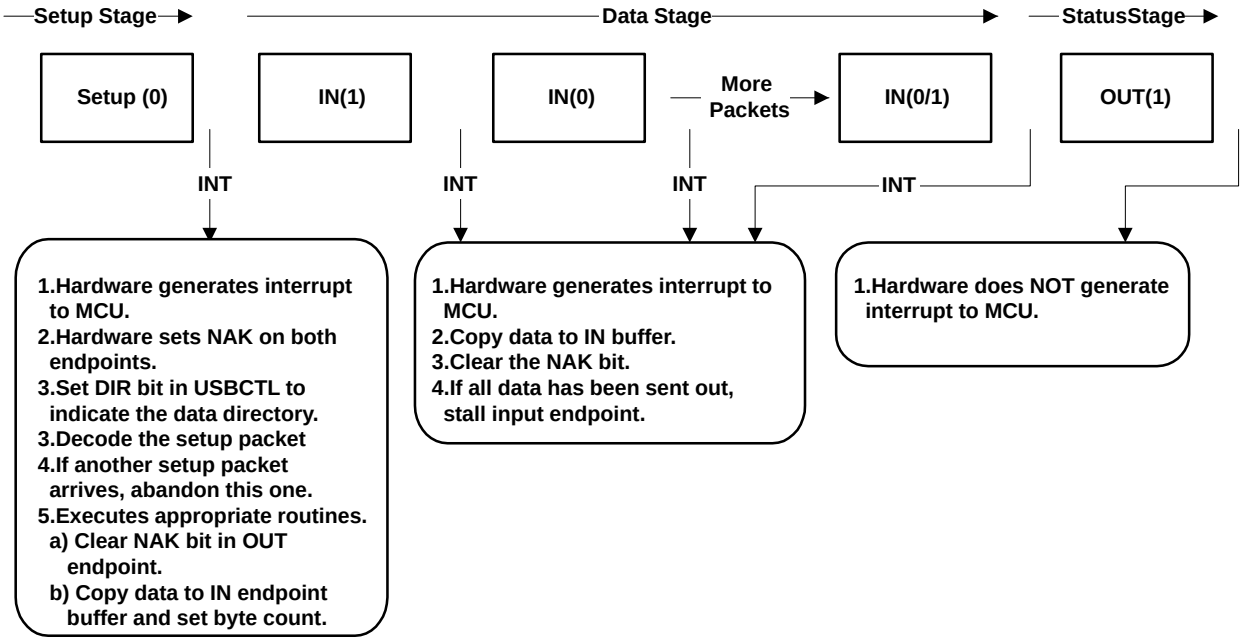


Figure 11–1. Control Read Transfer

Table 11–9. Bootcode Response to Control Read Transfer

CONTROL READ	ACTION IN BOOTCODE
Get status of device	Return power and remote wakeup settings
Get status of interface	Return 2 bytes of zeros
Get status of endpoint	Return endpoint status
Get descriptor of device	Return device descriptor
Get descriptor of configuration	Return configuration descriptor
Get descriptor of string	Return string descriptor
Get descriptor of interface	Stall
Get descriptor of endpoint	Stall
Get configuration	Return bConfiguredNumber value
Get interface	Return bInterfaceNumber value

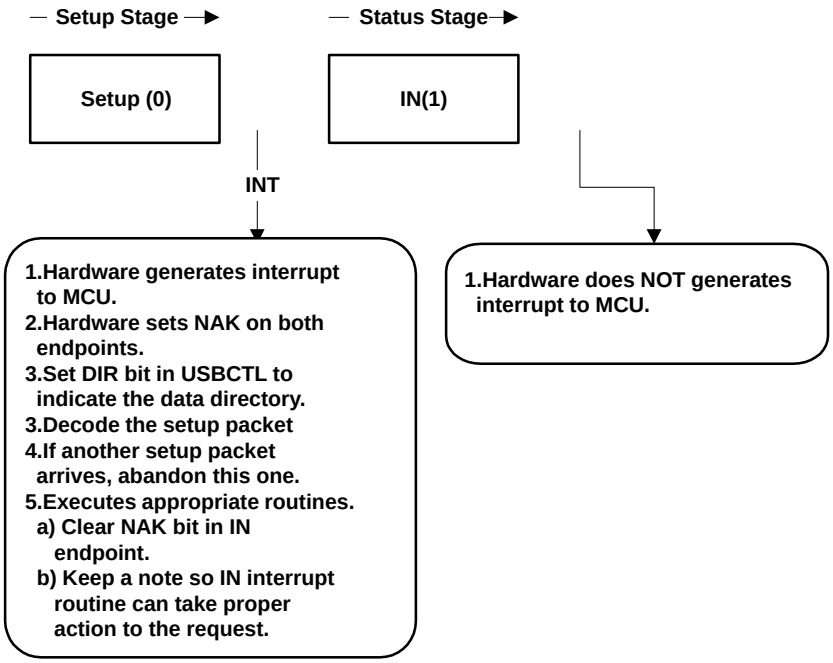


Figure 11–2. Control Write Transfer Without Data Stage

Table 11–10. Bootcode Response to Control Write Without Data Stage

CONTROL WRITE WITHOUT DATA STAGE	ACTION IN BOOTCODE
Clear feature of device	Stall
Clear feature of interface	Stall
Clear feature of endpoint	Clear endpoint stall
Set feature of device	Stall
Set feature of interface	Stall
Set feature of endpoint	Stall endpoint
Set address	Set device address
Set descriptor	Stall
Set configuration	Set bConfiguredNumber
Set interface	SetbInterfaceNumber
Sync. frame	Stall

11.9.1.2 Interrupt Handling Routine

The higher-vector number has a higher priority than the lower-vector number. Table 11–11 lists all the interrupts and source of interrupts.

Table 11–11. Vector Interrupt Values and Sources

G[3:0] (Hex)	I[2:0] (Hex)	VECTOR (Hex)	INTERRUPT SOURCE	INTERRUPT SOURCE SHOULD BE CLEARED
0	0	00	No Interrupt	No Source
1	1	10	Output–endpoint–1	VECINT register
1	2	12	Output–endpoint–2	VECINT register
1	3	14	Output–endpoint–3	VECINT register
1	4	16	Output–endpoint–4	VECINT register
2	4–7	18→1E	NOT USED	
2	1	20	Input–endpoint–1	VECINT register
2	2	22	Input–endpoint–2	VECINT register
2	3	24	Input–endpoint–3	VECINT register
2	4	26	Input–endpoint–4	VECINT register
2	4–7	28→2E	NOT USED	
3	0	30	STPOW packet received	USBSTA/ VECINT registers
3	1	32	SETUP packet received	USBSTA/ VECINT registers
3	2	34	PSOF interrupt	USBSTA/ VECINT registers
3	3	36	RESR interrupt	USBSTA/ VECINT registers
3	4	38	FSPR interrupt	USBSTA/ VECINT registers
3	5	3A	RTSR interrupt	USBSTA/ VECINT registers
3	6	3C	HSTL interrupt	USBSTA/ VECINT registers
3	7	3E	NOT USED	
4	0	40	I2C TXE interrupt	VECINT register
4	1	42	I2C TXE interrupt	VECINT register
4	2	44	Input–endpoint–0	VECINT register
4	3	46	Output–endpoint–0	VECINT register
4	4–7	48→4E	NOT USED	
5	0	50	UART1 status interrupt	LSR/VECNT register
5	1	52	UART1 modern interrupt	LSR/VECINT register
5	3–7	54→5E	NOT USED	
6	0	60	UART1 RXF interrupt	LSR/VECNT register
6	1	62	UART1 TXE interrupt	LSR/VECINT register
6	2–7	64→6E	NOT USED	
7	0–7	70→7E	NOT USED	
8	0	80	DMA1 interrupt	DMACSR/VECNT register
8	1	82	NOT USED	
8	2	84	DMA3 interrupt	DMACSR/VECNT register
8	3–7	86→7E	NOT USED	
9–15	0–7	90→FE	NOT USED	

11.9.2 Hardware Reset Introduced by the Firmware

This feature can be used in firmware upgrade. Once the upgrade is done, the application firmware disconnects from the USB for at least 200 ms to ensure OS has unloaded the device driver. The firmware then enables the watchdog timer (enabled by default after power-on reset) and enters an endless loop without resetting the watchdog timer. Once the watchdog timer times out, it resets the chip as if the chip gets the power-on reset. The bootcode takes over control and starts the power-on sequence again.

11.10 File Listings

The bootload code can be obtained from the TI website on the TUSB3410 product page, under *Related Software* with the name *TUSB3410 Bootcode Source Listing* (SLLC139.zip). The list shown below are the names of the files that are included in the downloaded zip.

- Types.h
- USB.h
- TUSB3410.h
- Bootcode.h
- Watchdog.h
- Bootcode.c
- Bootlsr.c
- BootUSB.c
- Header.h
- Header.c
- I2c.h
- I2c.c

12 Electrical Specifications

12.1 Absolute Maximum Ratings†

Supply voltage, V_{CC} . . . . . –0.5 V to 3.6 V
Input voltage, V_I . . . . . –0.5 V to V_{CC} + 0.5 V
Output voltage, V_O . . . . . –0.5 V to V_{CC} + 0.5 V
Input clamp current, I_{IK} . . . . . ±20 mA
Output clamp current, I_{OK} . . . . . ±20 mA

† Stresses beyond those listed under “absolute maximum ratings” may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under “recommended operating conditions” is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

12.2 Commercial Operating Condition (3.3 V)

PARAMETER			MIN	TYP	MAX	UNIT
V_{CC}	Supply voltage		3	3.3	3.6	V
V_I	Input voltage		0		V_{CC}	V
V_{IH}	High-level input voltage	TTL	2		V_{CC}	V
		CMOS	$0.7 \times V_{CC}$		V_{CC}	
V_{IL}	Low-level input voltage	TTL	0		0.8	V
		CMOS	0		$0.2 \times V_{CC}$	
T_A	Operating temperature		0		70	°C

12.3 Electrical Characteristics T_A = 25°C, V_{CC} = 3.3 V ±5%, V_{SS} = 0 V

PARAMETER			TEST CONDITIONS	MIN	TYP	MAX	UNIT
V_{OH}	High-level output voltage	TTL	I_{OH} = –4 mA	V_{CC} – 0.5			V
		CMOS		V_{CC} – 0.5			
V_{OL}	Low-level output voltage	TTL	I_{OL} = 4 mA			0.5	V
		CMOS				0.5	
V_{IT+}	Positive threshold voltage	TTL	$V_I = V_{IH}$			1.8	V
		CMOS				$0.7 \times V_{CC}$	
$V_{IT–}$	Negative threshold voltage	TTL	$V_I = V_{IH}$	0.8		1.8	V
		CMOS		$0.2 \times V_{CC}$			
V_{hys}	Hysteresis ($V_{IT+} – V_{IT–}$)	TTL	$V_I = V_{IH}$	0.3		0.7	V
		CMOS		$0.17 \times V_{CC}$		$0.3 \times V_{CC}$	
I_{IH}	High-level input current	TTL	$V_I = V_{IH}$			±20	µA
		CMOS				±1	
I_{IL}	Low-level input current	TTL	$V_I = V_{IL}$			±20	µA
		CMOS				±1	
I_{OZ}	Output leakage current (Hi-Z)		$V_I = V_{CC}$ or V_{SS}			±20	µA
I_{OL}	Output low drive current			0.1			mA
I_{OH}	Output high drive current			0.1			mA
I_{CC}	Supply current (operating)		Serial data at 921.6 k			15	mA
	Supply current (suspended)					200	µA

12.3 Electrical Characteristics T_A = 25°C, V_{CC} = 3.3 V ±5%, V_{SS} = 0 V (continued)

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
	Clock duty cycle‡			50%		
	Jitter specification‡				±100	ppm
C_I	Input capacitance				18	pF
C_O	Output capacitance				10	pF

‡ Applies to all clock outputs

13 Application Notes

13.1 Crystal Selection

The TUSB3410 requires a 12-MHz clock source to work properly. This clock source can be a crystal placed across the X1 and X2 terminals. A parallel resonant crystal is recommended. Most parallel resonant crystals are specified at a frequency with a load capacitance of 18 pF. This load can be realized by placing 33-pF capacitors from each end of the crystal to ground. Together with the input capacitance of the TUSB3410 and stray board capacitance, this provides close to two 36-pF capacitors in series to emulate the 18-pF load requirement. Note, that when using a crystal, it takes about 2 ms after power up for a stable clock to be produced.

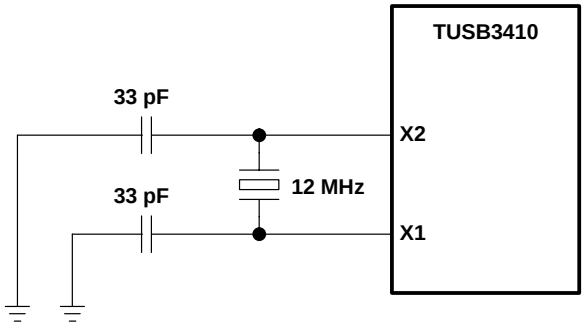


Figure 13–1. Crystal Selection

13.2 External Circuit Required for Reliable Bus Powered Suspend Operation

TI has found a potential problem with the action of the SUSPEND output pin immediately after power on. In some cases the SUSPEND pin can power up asserted high. When used in a bus powered application this can cause a problem because the VREGEN# input is usually connected to the SUSPEND output. This in turn causes the internal 1.8-V voltage regulator to shut down, which means an external crystal may not have time to begin oscillating, thus the device will not initialize itself correctly.

TI has determined an on-chip fix for this problem, but has not determined a schedule on when the fix will be implemented. In the meantime, the components R2 and D1 (rated to 25 mA) in the circuit shown below can be used as a workaround. Note that R1 and C1 are required components for proper reset operation, unless the reset signal is provided by another means. R2 and D1 can be left in place or removed once the silicon is modified.

Note that use of an external oscillator (1.8-V output) versus a crystal would avoid this situation, but it is not expected that many applications would use an oscillator. Also note that self-powered applications would probably not see this problem because the VREGEN# input would likely be tied low, enabling the internal 1.8-V regulator at all times.

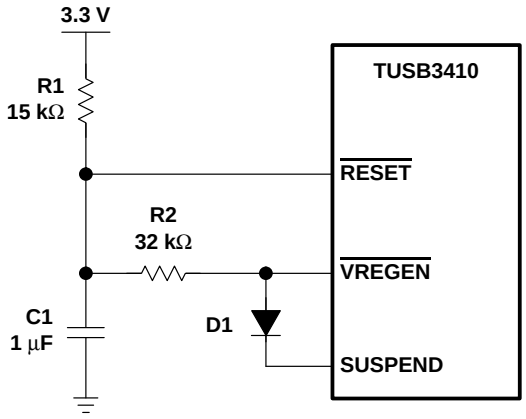


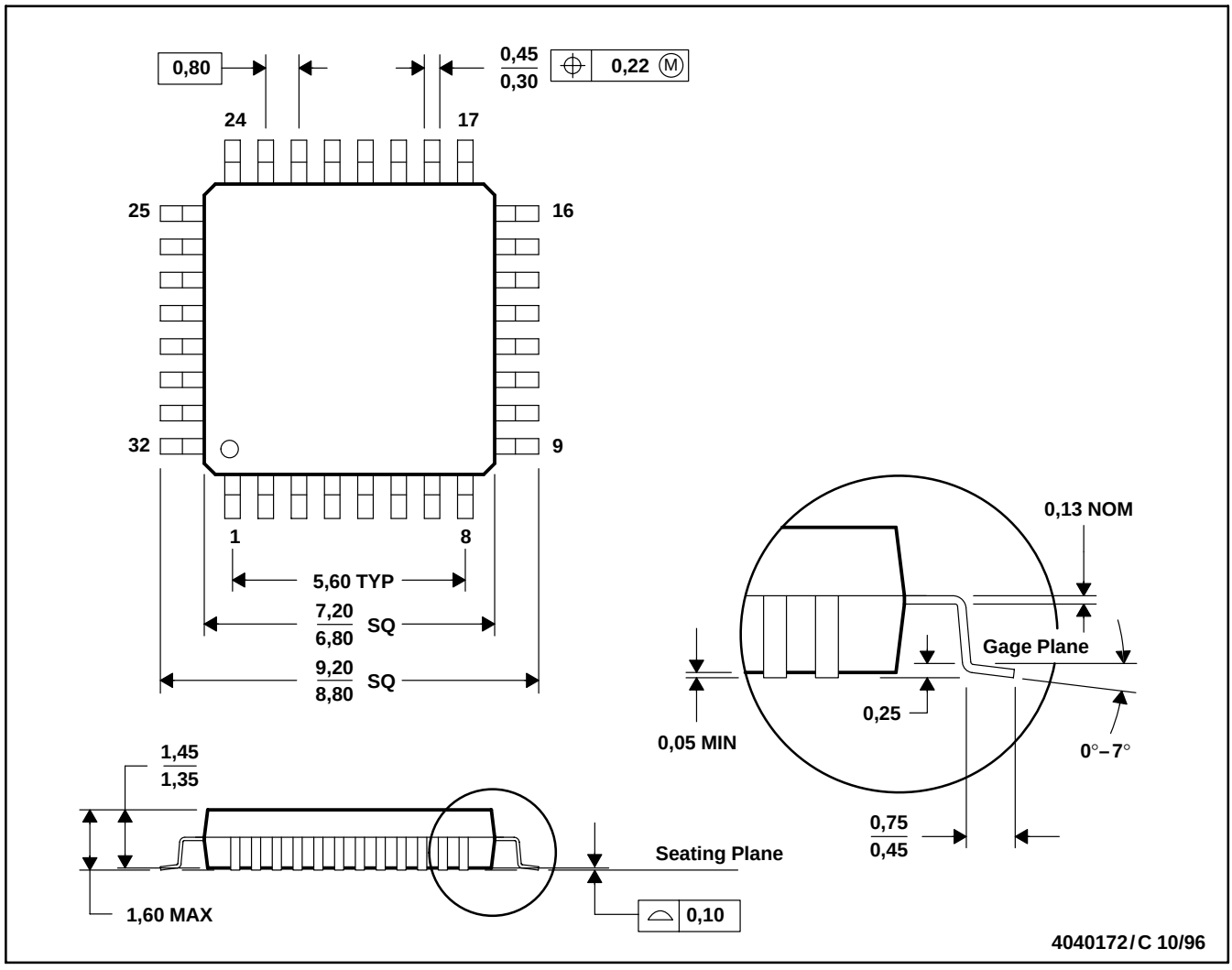
Figure 13–2. External Circuit

13.3 Wakeup Timing From $\overline{\text{WAKEUP}}$ or $\overline{\text{RI}}$ Pin

The TUSB3410 can be brought out of the suspended state, or woken up, by a command from the host. The TUSB3410 also supports remote wakeup and can be awakened by either of two input signals. A low pulse on the $\overline{\text{WAKEUP}}$ pin or a low-to-high transition on the $\overline{\text{RI}}$ pin wakes the device up. Note that for reliable operation, either condition must persist for approximately 3 ms minimum. This allows time for the crystal to power up since in the suspend mode the crystal interface is powered down. The state of the $\overline{\text{WAKEUP}}$ or $\overline{\text{RI}}$ pin is then sampled by the clock to verify there was a valid wakeup event.

14 Mechanical

VF (S-PQFP-G32)
PLASTIC QUAD FLATPACK



NOTES: A. All linear dimensions are in millimeters.
B. This drawing is subject to change without notice.
C. Falls within JEDEC MS-026