High-Performance 8-Bit Microcontrollers

Z8 Encore! XP® F6482 Series API

Programmer's Reference Manual

RM006404-0215

zilog
Embedded in Life
An ◻IXYS Company

⚠ **Warning:** DO NOT USE THIS PRODUCT IN LIFE SUPPORT SYSTEMS.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

iii

Revision History

Each instance in this document's revision history reflects a change from its previous version. For more details, refer to the corresponding pages linked in the table below.

Date	Revision Level	Description	Page No.
Feb 2015	04	Updated description for BSP_USB_EPABORT, BSP_USB_EPTRANSMIT; added content for fpUserEnum and Correct Usage in the BSP_USB section.	128, 134, 214
Dec 2014	03	Updated language for the Off_Thresh flag in the UART_RX_DMA data structure for clarity.	209
Apr 2014	02	Corrected CLKS Data Structure in the BSP API section.	148
Oct 2013	01	Original issue.	All

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

iv

Table of Contents

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

v

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An ◼ IXYS Company

vi

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An ◼ IXYS Company

vii

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

viii

The F6482 Series Board Support Package

This document serves as a reference to the F6482 Series Board Support Package (BSP) API, which is a blend of macros and C functions that facilitate application development with Zilog's Z8F6482 Series of MCUs by abstracting direct SFR manipulation from the programmer. Macros provide near-assembly-level access to SFRs without incurring the overhead (i.e., code size and execution time) of C functions for basic operations (e.g., masking a particular interrupt source, stopping a timer). More complex operations are implemented in C functions (user-visible source code), thereby eliminating the requirement for the customer to (re)write code for frequently-used operations such as transmitting a block of data through the UART.

The API of most BSP peripherals includes at least an *init function* (i.e., BSP_Xxx_Init, in which Xxx identifies the particular peripheral device being initialized) that configures the underlying peripheral and establishes the application developer's intended use model of the device (e.g., data transfer using polling or interrupts). Typically, the Init API requires a reference to a peripheral-specific data structure containing configuration information used to initialize the peripheral's special function registers. Applications must call a peripheral's init function before calling any of the peripheral's other BSP functions.

Many BSP peripherals include data transfer APIs. These peripherals can be configured (via the peripheral's Init API) to perform the data transfer operation synchronously or asynchronously. Typically, this operation involves selecting between a poll-mode (synchronous) setup function vs. a DMA- or interrupt-driven (both asynchronous) setup function within the peripheral's configuration structure.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

2

API function calls that perform synchronous transfers do not rerun control to the caller until the operation specified by the API completes (or aborts). API function calls that perform asynchronous transfers typically return immediately while the actual data transfer completes in the background. Peripherals that can be configured for asynchronous data transfer also include an (optional) transfer complete callback function pointer that the application can set to the address of a routine the BSP calls when the transfer completes. With asynchronous data transfers, the application should not modify the contents of the buffer used in the transfer until the BSP calls the application's transfer complete handler. If the initial call to an asynchronous API fails and the transfer operation cannot be performed the transfer complete callback is not called.

Applications that call the init API of any of the BSP peripherals are cautioned against directly modifying any of that peripheral's special function registers; such modifying can cause the peripheral to operate unexpectedly (or not at all). If an application must modify the configuration of a peripheral after the peripheral's init API is called, the application should call the peripheral's *stop API* (i.e., BSP_Xxx_Stop, in which Xxx identifies the target peripheral). The application can next modify the peripheral's configuration structure, then call the peripheral's init routine. Similarly, when BSP services are no longer required for a given peripheral, applications can call the peripheral's BSP_Xxx_Stop API to direct the BSP to stop using the device.

Sample Programs

Included in Zilog Developer Studio II (ZDSII) for Z8 Encore! are sample programs that demonstrate the APIs described in this document. Upon accepting the default prompts during the ZDSII installation, these samples will be located in either of the following paths, depending on OS, as follows:

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An∎IXYS Company **3**

On 64-Bit Windows Installations

```
C:\Program Files (x86)\Zilog\ZDSII_Z8Encore!_X.Y.Z\
BSP\Z8F6482\Samples
```

On 32-Bit Windows Installations

```
C:\Program Files\Zilog\ZDSII_Z8Encore!_X.Y.Z\BSP\
Z8F6482\Samples
```

▶ **Note:** In the above paths, X.Y.Z refers to the ZDSII version number.

ZDSII – Z8 Encore! is available for download from the Free Software category of the Zilog Store.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

4

Advanced Encryption Standard Accelerator

The AES API implements the following functions:

- BSP_AES_Init – see page 5
- BSP_AES_Stop – see page 7
- BSP_AES_Transform – see page 8

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An∎IXYS Company

5

BSP_AES_INIT

Prototype

```
BSP_STATUS BSP_AES_Init (BSP_AES * pAES )
```

Parameter

pAES   A pointer to a BSP_AES structure that configures the AES
            for encryption or decryption operations; see the AES
            Structures and Unions in the BSP API section on page 141.

Return Value

BSP_ERR_SUCCESS if no errors occur.

BSP_ERR_IN_USE if the AES has already been initialized and
BSP_AES_Stop() was not subsequently called, or if a DMA channel is
requested which was previously initialized.

BSP_ERR_INVALID_PARAM for some invalid configurations of the
BSP_AES structure.

Description

This API can be used to configure the AES for encryption or decryption
before its first use, and can also be used to reconfigure the AES, which
can change any of the following operations:

- Encryption mode (ECB, OFB, CBC, or decrypt key derivation)

- From encryption to decryption, or vice versa

- Data transfer mode (polling, interrupt, or DMA)

- Encryption (or decryption) key

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

6

Correct Usage

This function should not be called if the AES has been previously initialized, and should only be called after first calling `BSP_AES_Stop()`; otherwise, an error status will be returned.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An◼IXYS Company

7

BSP_AES_STOP

Prototype

```
BSP_STATUS BSP_AES_Stop (void )
```

Parameters

None.

Return Value

BSP_ERR_SUCCESS if no errors occur.

BSP_ERR_BUSY if called while an encryption, decryption, or key derivation operation is in progress.

BSP_ERR_INVALID_PARAM if the AES has not been initialized, or if BSP_AES_Stop() has already been called since the previous initialization.

Description

This API can be used to shut down the AES gracefully when it is no longer required, and can also be used to undo a previous initialization prior to reconfiguring the AES.

Correct Usage

This function should not be called if an encryption, decryption, or decrypt key derivation is in progress. This function should not be called if the AES is not currently in an initialized state; in either case, an error status will be returned.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

z i l o g
Embedded in Life
An ▢ IXYS Company

8

BSP_AES_TRANSFORM

Prototype

```
BSP_STATUS BSP_AES_Transform (HANDLE hInput,
HANDLE hOutput,
HANDLE iv,
BSP_SIZE nBlocks)
```

Parameters

hInput	A pointer to the buffer of input data to be encrypted or decrypted.
hOutput	A pointer to a buffer, in which the transformed data (the encrypted or decrypted text, or the decrypt key) may be stored.
iv	A pointer to the initialization vector, if using those encryption modes (OFB, CBC) which require one. For other modes (ECB or decrypt key derivation) set this to NULLPTR.
nBlocks	The number of 16-byte blocks of input data to be transformed.

Return Value

BSP_ERR_SUCCESS if no errors occur.

BSP_ERR_BUSY if a transform operation is already in progress.

Description

Performs any of the three supported data transforms provided by the AES: encryption, decryption, or the derivation of a decrypt key.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An∎IXYS Company

9

Correct Usage

The data to be encrypted or decrypted must be an exact number of 16-byte blocks. This statement will always be true of a message that has already been encrypted using AES, but if you have a plain text message to encrypt, you must pad it if necessary to an integral number of blocks before calling BSP_AES_Transform(). Several padding schemes are in widespread use.

To decrypt messages encrypted with either the ECB or CBC encryption mode, a decrypt key must first be derived (using the encryption key and any input message) and stored. This task is not necessary for decrypting messages encrypted with OFB Encryption Mode; this requirement is is a feature of the AES standard itself.

When deriving a decrypt key, you must use polling or interrupt data transfers, and DMA transfers will hang; this requirement is a feature of the Z8F6482 MCU's AES accelerator hardware.

When using OFB Decryption Mode, the operation (i.e., the Decrypt member of the BSP_AES structure) must be set to AES_ENCRYPT for both encrypting and decrypting messages; it is a feature of OFB Encryption Mode.

To decrypt a message encrypted with CBC Encryption Mode, you must use the ECB decryption mode (after having first derived and stored the decrypt key); this requirement is a feature of the Z8F6482 MCU's AES accelerator hardware.

To recover the plain text of a message encrypted with CBC Encryption Mode, a further software transform must be applied to the output data from the Z8F6482 MCU's AES accelerator; this requirement is a feature of the Z8F6482 MCU's AES accelerator hardware. An example of such software is included in some of the AES example programs.

This function should not be called while a previous transform initiated by BSP_AES_Transform() is still in progress. This issue can occur even after BSP_AES_Transform() has returned, if interrupts or DMA are

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

10

being used for the data transfers. In such cases, your callback function can alert your application that the transform is complete. If a transform is still in progress, `BSP_AES_Transform()` will return an error status.

Clock System API Reference

The BSP Clock System (CLKS) driver is used to simplify the process of programming the individual clock system special function registers. Applications can use the BSP-supplied default clock configuration or a custom configuration defined by the application. The BSP CLKS driver next performs the individual programming tasks necessary to enable the specified configuration.

CLKS Functions in the BSP API

The CLKS API implements the following function:

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

z i l o g
Embedded in Life
An ◻ IXYS Company

12

BSP_CLKS_CONFIG

Prototype

```
void BSP_CLKS_Config( BSP_CLKS * pClks );
```

Parameters

pClks   References a variable of type BSP_CLKS that specifies the special function register values to be used during initialization of the clock system.

Return Value

None.

Description

Typically, the first BSP function that application programs call is the BSP_CLKS_Config API. This routine initializes the clock system using the register values referenced by the pClks parameter. After the clock system has been initialized, other BSP peripheral drivers can be initialized using a known set of clock frequencies and sources.

This function disables interrupts and switches the system clock source to the watchdog timer oscillator while the clocks system special function registers are reprogrammed. After the entire clock system has been reconfigured the system clock source is switched to the clock source specified in the BSP_CLKS structure and interrupts are reenabled. When this function returns control to the caller, the new clock configuration is active. However, if the FLL frequency is modified as a result of calling this API, then the FLL may not yet have attained final lock by the time this API returns control. In essence, this API returns control after the FLLRDY bit has been set, which typically occurs before the FLLDONE bit is also set. If the application intends to enter Stop Mode immediately after calling the BSP_CLKS_Config API to modify the FLL frequency, then the applica-

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An ∎IXYS Company

13

tion should poll for the FLLDONE bit, as shown in the code fragment below. If the application does not wait for FLLDONE before entering Stop Mode, then the MCU may fail to operate as expected.

```
BSP_CLKS_Config( &BSP_Default_CLKS_Cfg );
/*
* Wait for FLLDONE before entering Stop Mode after
* calling the BSP_CLKS_Config to modify the FLL
* frequency.
*/
while( !(DCOCTL & CLKS_FLLDONE) );
asm( "STOP" );
```

Correct Usage

The `BSP_CLKS_Config` routine does not perform any error checking or validation of the clock system SFR values supplied through the `pClks` parameter. If the `pClks` pointer references a `BSP_CLKS` structure containing invalid or inconsistent values, then the system could fail to operate as expected; or will not operate at all.

The BSP library includes a default clock system configuration (as defined by the `BSP_Default_CLKS_Cfg` global variable) that may be used by applications targeting the Z8F6482 Series Development Kit. The BSP default clock configuration is enabled by making the following function call:

```
BSP_CLKS_Config( &BSP_Default_CLKS_Cfg );
```

Alternatively, applications can initialize a local variable of type `BSP_CLKS` with values appropriate for the target hardware platform and then pass the address of the local variable to the `BSP_CLKS_Config` routine.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

14

Digital to Analog Converter

The DAC API implements the following functions:

- <u>BSP_DAC_Init</u> – see page 15
- <u>BSP_DAC_Stop</u> – see page 17
- <u>BSP_DAC_Abort</u> – see page 18
- <u>BSP_DAC_OutputOneByteSignedValue</u> – see page 19
- <u>BSP_DAC_OutputOneByteUnsignedValue</u> – see page 21
- <u>BSP_DAC_OutputTwoByteSignedValue</u> – see page 23
- <u>BSP_DAC_OutputTwoByteUnsignedValue</u> – see page 25
- <u>BSP_DAC_OutputBuffer</u> – see page 27

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

z i l o g
Embedded in Life
An ◼ IXYS Company

15

BSP_DAC_INIT

Prototype

```
BSP_STATUS BSP_DAC_Init (BSP_DAC * pDac )
```

Parameter

pDac   A pointer to a BSP_DAC structure that configures the DAC
              for conversion operations; see the DAC Structures and
              Unions in the BSP API section on page 152.

Return Value

BSP_ERR_SUCCESS if no errors occur.

BSP_ERR_IN_USE if the DAC has already been initialized and
BSP_DAC_Stop() was not subsequently called, or if the DAC has
already been enabled in the PWRCTL1 Register, or if a DMA channel
which was previously initialized is requested.

BSP_ERR_INVALID_PARAM if DMA data transfer is requested but no
DMA channel is specified.

Description

This API can be used to enable and configure the DAC, before its first
use, and can also be used to reconfigure the DAC, which can change any
of the following issues:

- The power level at which to operate the DAC

- The DAC voltage reference

- Whether data are to be treated as signed or unsigned

- Whether to drive DAC conversions directly or via the Event System

- Whether data are to be treated as 8, 12, or 16 bits

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An ⬛ IXYS Company

16

Correct Usage

Some of the possible DAC voltage reference selections require external hardware, or impose requirements on the power supply voltage, or interact with settings of the Analog to Digital Converter (ADC). The BSP software does not check for any of these conditions. It is your responsibility to ensure that the software configuration used in BSP_DAC_Init() is consistent with your hardware setup.

This function should not be called if the DAC has been previously initialized, and should only be called after first calling BSP_DAC_Stop(); otherwise, an error status will be returned.

It is not necessary to call this function again and reinitialize the DAC, if an operation (such as output of an endlessly looping buffer) was terminated by calling BSP_DAC_Abort(), and you now want to output more data using the same hardware configuration.

After calling this function, it is your responsibility to ensure that the data to be converted are either one-byte or two-byte values, in agreement with the parameter used to initialize the DAC.

This function automatically sets up the GPIO system to drive DAC output on pin PC3.

An initial output value must be specified. The DAC output will be driven at this level from the time when this function finishes execution, enabling the DAC, until the user first requests a conversion.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An ■IXYS Company

17

BSP_DAC_STOP

Prototype

```
BSP_STATUS BSP_DAC_Stop (void )
```

Parameters

n/a

Return Value

BSP_ERR_SUCCESS if no errors occur.

BSP_ERR_BUSY if called while a DAC conversion is in progress.

BSP_ERR_FAILURE if the DAC has not been initialized, or if
BSP_DAC_Stop() has already been called since the previous initialization.

Description

This API can be used to shut down the DAC gracefully when it is no longer required, and can also be used to disable the DAC (thereby reducing power consumption) and to undo a previous initialization prior to reconfiguring the DAC.

Correct Usage

This function should not be called if a DAC conversion is in progress. This function should not be called if the DAC is not currently in an initialized state; in either case, an error status will be returned. If you must terminate an ongoing DAC operation, such as output of an endlessly looping buffer, call BSP_DAC_Abort() instead.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An∎IXYS Company

18

BSP_DAC_ABORT

Prototype

```
void BSP_DAC_Abort (void )
```

Parameters

n/a

Return Value

n/a

Description

This API can be used to terminate any in-progress DAC conversion. Its primary intended use is to terminate the output of an endlessly looping buffer. Data transfers to the DAC are terminated, but the DAC is not disabled and the DAC software and hardware remain configured as they were in the ongoing conversion (if any). As a result, a new conversion can immediately begin using the same configuration, if desired. The DAC will be left outputting the last value that it converted before this function was called.

Correct Usage

If you wish to disable or reconfigure the DAC, you should call BSP_DAC_Stop() after this function call returns. The DAC remains enabled when BSP_DAC_Abort() returns, and may continue to consume some power.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An◻IXYS Company

19

BSP_DAC_OUTPUTONEBYTESIGNEDVALUE

Prototype

```
BSP_STATUS BSP_DAC_OutputOneByteSignedValue (INT8
value )
```

Parameter

value   An 8-bit, signed integer value which is to be driven on the
          DAC output pin.

Return Value

BSP_ERR_SUCCESS if no errors occur.

BSP_ERR_BUSY if the DAC is already busy converting a buffer of data.

Description

This API can be used for direct and immediate output on the DAC of a
signed, 8-bit value.

This function is identical to BSP_DAC_OutputOneByteUnsignedValue()
except for the type of its parameter.

Correct Usage

When making one call or a sequence of calls to this function, you are
essentially operating the DAC as an 8-bit DAC with only 256 discrete
output levels rail-to-rail. It is your responsibility to determine that the
DAC hardware was configured consistently with this operational mode in
your call to BSP_DAC_Init().

When calling this function, the DAC will immediately terminate any pre-
vious output of a single value that might be in progress due to a previous
call to this function. It will hold the output value indefinitely, until a sub-
sequent call to this function or to BSP_DAC_Stop(). Controlling the

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

20

setup and hold time for the DAC conversion is the responsibility of your application code.

If you call this function while a buffer of data is still being output following a previous call to `BSP_DAC_OutputBuffer()`, an error will be returned.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An◼IXYS Company

21

BSP_DAC_OUTPUTONEBYTEUNSIGNEDVALUE

Prototype

```
BSP_STATUS BSP_DAC_OutputOneByteUnsignedValue (UINT8
value )
```

Parameter

value   An 8-bit, unsigned integer value which is to be driven on
             the DAC output pin.

Return Value

BSP_ERR_SUCCESS if no errors occur.

BSP_ERR_BUSY if the DAC is already busy converting a buffer of data.

Description

This API can be used for direct and immediate output on the DAC of an
unsigned, 8-bit value.

This function is identical to BSP_DAC_OutputOneByteSignedValue()
except for the type of its parameter.

Correct Usage

When making one call or a sequence of calls to this function, you are
essentially operating the DAC as an 8-bit DAC with only 256 discrete
output levels rail-to-rail. It is your responsibility to determine that the
DAC hardware was configured consistently with this operational mode in
your call to BSP_DAC_Init().

When calling this function, the DAC will immediately terminate any pre-
vious output of a single value that might be in progress due to a previous
call to this function. It will hold the output value indefinitely, until a sub-
sequent call to this function or to BSP_DAC_Stop(). Controlling the

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

22

setup and hold time for the DAC conversion is the responsibility of your application code.

If you call this function while a buffer of data is still being output following a previous call to `BSP_DAC_OutputBuffer()`, an error will be returned.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

23

BSP_DAC_OUTPUTTWOBYTESIGNEDVALUE

Prototype

```
BSP_STATUS BSP_DAC_OutputTwoByteSignedValue (INT16 value )
```

Parameter

value   A 16-bit, signed integer value which is to be driven on the DAC output pin.

Return Value

BSP_ERR_SUCCESS if no errors occur.

BSP_ERR_BUSY if the DAC is already busy converting a buffer of data.

Description

This API can be used for direct and immediate output on the DAC of a signed, 2-byte value.

This function is conceptually similar to BSP_DAC_OutputOneByteSignedValue() except for the type of its parameter, but the code that performs this function is somewhat larger and more complex.

This function is conceptually similar to BSP_DAC_OutputTwoByteUnsignedValue() except for the type of its parameter, but the code that performs this function is somewhat different; the size and complexity are similar.

Correct Usage

When making one call or a sequence of calls to this function, you are operating the DAC at its full resolution as a 12-bit DAC with 4096 discrete output levels rail-to-rail. The restriction to only 12 usable bits from

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

24

an input value that is formally 16 bits may be achieved in one of two ways. You may opt to provide input data with a full 16-bit range and allow the DAC to use only the most significant 12 bits; in this case, you must specify left justification of the DAC data registers in the BSP_DAC structure that you pass to BSP_DAC_Init(). Alternatively, you may provide input data that are already restricted to a 12-bit range; in that case, you must specify right justification. In either case, it is your responsibility to determine that the DAC hardware was configured consistently with the range of your input data, in your call to BSP_DAC_Init().

When calling this function, the DAC will immediately terminate any previous output of a single value that might be in progress due to a previous call to this function. It will hold the output value indefinitely, until a subsequent call to this function or to BSP_DAC_Stop(). Controlling the setup and hold time for the DAC conversion is the responsibility of your application code.

If you call this function while a buffer of data is still being output following a previous call to BSP_DAC_OutputBuffer(), an error will be returned.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An ◼ IXYS Company

25

BSP_DAC_OUTPUTTWOBYTEUNSIGNEDVALUE

Prototype

BSP_STATUS BSP_DAC_OutputTwoByteUnsignedValue (UINT16 value)

Parameter

value   A 16-bit, unsigned integer value which is to be driven on the DAC output pin.

Return Value

BSP_ERR_SUCCESS if no errors occur.

BSP_ERR_BUSY if the DAC is already busy converting a buffer of data.

Description

This API can be used for direct and immediate output on the DAC of an unsigned, 2-byte value.

This function is conceptually similar to BSP_DAC_OutputOneByteUnsignedValue() except for the type of its parameter, but the code that performs this function is somewhat larger and more complex.

This function is conceptually similar to BSP_DAC_OutputTwoByteSignedValue() except for the type of its parameter, but the code that performs this function is somewhat different; the size and complexity are similar.

Correct Usage

When making one call or a sequence of calls to this function, you are operating the DAC at its full resolution as a 12-bit DAC with 4096 discrete output levels rail-to-rail. The restriction to only 12 usable bits from

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An ∎IXYS Company

26

an input value that is formally 16 bits may be achieved in one of two ways. You may opt to provide input data with a full 16-bit range and allow the DAC to use only the most significant 12 bits; in this case, you must specify left justification of the DAC data registers in the BSP_DAC structure that you pass to BSP_DAC_Init(). Alternatively, you may provide input data that are already restricted to a 12-bit range; in that case, you must specify right justification. In either case, it is your responsibility to determine that the DAC hardware was configured consistently with the range of your input data, in your call to BSP_DAC_Init().

When calling this function, the DAC will immediately terminate any previous output of a single value that might be in progress due to a previous call to this function. It will hold the output value indefinitely, until a subsequent call to this function or to BSP_DAC_Stop(). Controlling the setup and hold time for the DAC conversion is the responsibility of your application code.

If you call this function while a buffer of data is still being output following a previous call to BSP_DAC_OutputBuffer(), an error will be returned.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An∎IXYS Company

27

BSP_DAC_OUTPUTBUFFER

Prototype

```
BSP_STATUS BSP_DAC_OutputBuffer  ( HANDLE hInput,
                                   BSP_SIZE Len,
                                   BSP_SIZE nRepeats )
```

Parameters

hInput	A pointer to the buffer of input data to be driven on the DAC output.
Len	The size of the input buffer, in bytes (which may be the same as the number of values in the buffer, or may be twice that number, depending on whether an individual input value is one byte or two).
nRepeats	The number of times that the output of the entire buffer is to be repeated. If nRepeats is set to 0, the buffer will be output indefinitely until BSP_DAC_Abort() is called.

Return Value

BSP_ERR_SUCCESS if no errors occur.

BSP_ERR_BUSY if the DAC is already busy converting a buffer of data.

Description

Drives the output of the data buffer to the DAC output pin. The size and *signedness* of input data values must be consistent with the setup of the DAC hardware given by the structure passed to BSP_DAC_Init(). The buffer may be output a single time, multiple times, or indefinitely. The timing of the conversions is driven by the Event System, which must be set up and enabled separately. Data transfer will begin on the next Event System event after this function call.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

28

Correct Usage

This function call will not do anything until the application code has separately set up and enabled the Event System with a source (such as a timer) connected to a particular Event System channel, with the DAC connected as a destination to that same channel. Each time the Event System source fires (for example, on every rising edge of a timer output signal), the DAC will both begin conversion of the existing value in the DAC data registers, and request loading of the next value from the ISR or DMA.

Additionally, the DAC must have been previously configured for Event System operation and for either interrupt or DMA-driven data transfers in the call to `BSP_DAC_Init()`.

A most important note about use of this function is that if individual data values are two bytes and the DMA (rather than interrupts) is used for data transfer, the data values in the buffer must be byte-swapped (i.e., their endianness must be reversed) before calling this function. In essence, to drive the `0x06B3` value as a 12-bit unsigned value, the value stored in the buffer must be `0xB306`. This particular issue is a consequence of the Z8F6482 MCU's DAC design, in which the DAC DMA request is deasserted upon writing the high, not the low, byte to the DAC data registers; that design decision in turn was chosen to support the 8-bit output mode for the DAC. The BSP software arranges for the byte-swapped input data to be placed correctly in the DAC data registers by the DMA.

A fine point of the operation of `BSP_DAC_OutputBuffer()` concerns the action of the `stopInstantly` member of the `BSP_DAC` structure that was used in `BSP_DAC_Init()`. You may want to set this value true f you want to transition as quickly as possible from output of one buffer to another, for example if streaming data to the DAC from an external device. (You may also want to insert a dummy value as the final element of the input buffer in such an application.) In most other cases, you will probably want to set `stopInstantly` to false. To learn more, see the

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An◻IXYS Company

29

description of stopInstantly in the the <u>DAC Structures and Unions in the BSP API</u> section on page 152.

This function should not be called while a previous buffer output initiated by BSP_DAC_OutputBuffer() is still in progress. This issue can occur even after BSP_DAC_OutputBuffer() has returned, because interrupts or DMA are used for the data transfers to the DAC in a nonblocking mode. Your callback function can alert your application that the buffer output is complete. If output is still in progress, BSP_DAC_OutputBuffer() will return an error status.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

30

Direct Memory Access API Reference

The BSP DMA driver allows applications to perform DMA transfers with very little programming. The BSP library can be configured to enable support for up to BSP_DMA_NUM_CH (currently defined as 4) channels and supports both direct transfers and linked list operation.

DMA Functions in the BSP API

The DMA API implements the following functions:

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

z i l o g
Embedded in Life
An ◼ IXYS Company

31

BSP_DMA_INIT

Prototype

```
void BSP_DMA_Init( void );

#define BSP_DMA_Init() \
 (DMACTL = DMA_ROUND_ROBIN | DMA_BURST_4 |
DMA_AUTOINC)
```

Parameters

None.

Return Value

None.

Description

The BSP_DMA_init API is a macro that initializes the DMA global control register (DMACTL) for use by other DMA routines within the BSP. Applications should call the BSP_DMA_Init API before calling other BSP DMA functions or using other BSP drivers that are configured to use a DMA channel(s).

Correct Usage

Applications can modify the definition of the BSP_DMA_Init macro or programmatically initialize the DMACTL Register to use a custom configuration. The setting of the PRIORITY and BURST fields within the DMACTL Register may be set as appropriate for the application; but the AUOTINC field should be set to 1 and all other fields set to 0 to ensure BSP DMA routines function as described in this document.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An ◼ IXYS Company

32

BSP_DMA_ACQUIRE

Prototype

```
BSP_STATUS BSP_DMA_Acquire
(
 UINT16 Base,
 FP_DMA_CB fpDone
);
```

Parameters

Base   Specifies which DMA channel is being acquired. Base
            must be in the range of BSP_DMA0 to BSP_DMA3.

fpDone   Specifies the application callback routine that the DMA
            driver calls when the DMA channel completes a transfer,
            or when the remaining transfer count crosses the
            watermark threshold. This parameter must not be NULLPTR
            (0).

Return Value

BSP_ERR_SUCCESS is returned if no errors occur.

BSP_IN_USE is returned if the specified DMA channel has already been
acquired from a previous call to this API.

BSP_ERR_INVALID_PARAM is returned (with the debug version of the
BSP library) if the Base parameter exceeds BSP_DMA3. This error code is
also returned (with both the debug and release versions of the library) if
the fpDone parameter is NULLPTR, or if the specified DMA channel is
not available for allocation through the BSP.

Description

Applications call this API to request (exclusive) access to the DMA chan-
nel corresponding to the Base parameter. After a DMA channel has been

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An◼IXYS Company

33

acquired, it may not be reacquired until it is released (see the BSP_DMA_Release API on page 35). This limitation ensures that BSP peripheral drivers which use BSP DMA services and applications will not be able to accidentally acquire a channel that is already in use by some other entity in the system.

If this API returns BSP_ERR_SUCCESS the application may use the requested DMA channel for data transfer (see the BSP_DMA_Setup API on page 36 and the BSP_DMA_SetupLL API on page 39). After the data transfer completes (or crosses a watermark threshold) the application's callback routine (referenced by the fpDone parameter) is called. The function prototype of the callback routine is shown in the following code snippet:

```
reentrant void DmaCallback
(
 UINT16 Base
);
```

The Base parameter indicates which DMA channel has completed its transfer (or crossed the watermark threshold). The value of Base is between BSP_DMA0 and BSP_DMA3.

Correct Usage

Several BSP peripheral drivers can optionally be configured to transfer data using DMA. These BSP drivers will include routines to acquire, configure, and start the DMA channel(s) allocated to them. In this instance the only DMA API that the application must call is BSP_DMA_Init. If, however, the application must perform a DMA transfer that does not involve a BSP peripheral driver, then it is necessary for the application to make explicit calls to the BSP_DMA_Acquire API and possibly other APIs to initiate the actual transfer operation.

The BSP DMA driver uses an array of function pointers to indicate which DMA channels are available for allocation through the BSP_DMA_Acquire API. The array is defined in the DMA_Cfg.c file

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

34

located in the `..\BSP\Cfg` folder. By default, all available DMA chan-
nels are assigned to the BSP for allocation through this API as shown in
the following code snippet:

```
FP_DMA_SETUP fpBSP_Default_DmaSetup[ BSP_DMA_NUM_CH ] =
{
 DMA_Ch0_Init,
 DMA_Ch1_Init,
 DMA_Ch2_Init,
 DMA_Ch3_Init
};
```

Applications that must reserve one or more DMA channels for their pri-
vate use can modify the contents of the `fpBSP_Default_DmaSetup`
array, add the `DMA_Cfg.c` file to their project and rebuild the application.
To reserve channel 0 set the `fpBSP_Default_DmaSetup[0]` array
value to `NULLPTR (0)`. Similarly, to reserve channel n, in which n is in
the range of 0 to `BSP_NUM_Ch` −1, set the
`fpBSP_Default_DmaSetup[n]` array value to `NULLPTR (0)`.

If an entry in the `fpBSP_Default_DmaSetup`array is `NULLPTR` and
the `BSP_DMA_Acquire` API is called to acquire that same DMA channel,
then the `BSP_DMA_Acquire` API will return
`BSP_ERR_INVALID_PARAM`, thereby preventing the BSP peripheral driv-
ers from using the DMA channel.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An◻IXYS Company

35

BSP_DMA_RELEASE

Prototype

```
void BSP_DMA_Release( UINT16 Base );
```

Parameters

Base   Specifies which DMA channel is being acquired. Base
            must be in the range of BSP_DMA0 to BSP_DMA3.

Return Value

BSP_ERR_SUCCESS is returned if no errors occur.

BSP_ERR_INVALID_PARAM is returned (with the debug version of the
BSP library) if the Base parameter exceeds BSP_DMA3.

Description

When an application is finished using a DMA channel it can be returned
to the BSP for subsequent reallocation through the BSP_DMA_Acquire
API.

Correct Usage

Most applications are not required to call this function. Applications typi-
cally acquire one or more DMA channels, then continue to use the DMA
channels during the lifetime of the application. In this instance, there is no
requirement for an application to call the BSP_DMA_Release API.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

36

BSP_DMA_SETUP

Prototype

```
void BSP_DMA_Setup
(
 UINT16 Base,
 HANDLE hSrc,
 HANDLE hDst,
 UINT16 Count,
 UINT8 Ctl0,
 UINT8 Ctl1
);

#define BSP_DMA_Setup( DmaBase, Src, Dst, Cnt, Ctl0,
Ctl1 ) \
{ \
 DMA_SA( (DmaBase) ) = DMA_SA_SRCH; \
 DMA_SD( (DmaBase) ) = (UINT16)(Src) >> 8; \
 DMA_SD( (DmaBase) ) = (UINT8)(Src); \
 DMA_SD( (DmaBase) ) = (UINT16)(Dst) >> 8; \
 DMA_SD( (DmaBase) ) = (UINT8)(Dst); \
 DMA_SD( (DmaBase) ) = (Cnt) >> 8; \
 DMA_SD( (DmaBase) ) = (Cnt); \
 DMA_SD( (DmaBase) ) = (Ctl0); \
 DMA_SD( (DmaBase) ) = (Ctl1); \
}
```

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

z i l o g
Embedded in Life
An ◼ IXYS Company

37

Parameters

Base	Specifies which DMA channel is being configured for a data transfer operation. Base must be in the range of BSP_DMA0 to BSP_DMA3.
hSrc	Address of the (RAM) memory buffer containing the data to be transferred or the Special Function Register (SFR) address of a peripheral device from which data is to be extracted.
hDst	Address of the (RAM) memory buffer into which data is transferred or the Special Function Register (SFR) address of a peripheral device to which data is to be transferred.
Count	Specifies the number of bytes of data to be transferred.
Ctl0	Specifies the value to be written to the DMA channel's DMAxCTL0 Special Function Register. To learn more about the meaning of this value, refer to the Z8F6482 Series Product Specification.
Ctl1	Specifies the value to be written to the DMA channel's DMAxCTL1 Special Function Register. To learn more about the meaning of this value, refer to the Z8F6482 Series Product Specification.

Return Value

None.

Description

The BSP_DMA_Setup macro is used to configure the specified DMA channel's special function registers for a pending DMA transfer. If the ENABLE bit in the Ctl1 parameter is specified, then the transfer begins when the DMA requestor specified in the Ctl1 parameter issues a DMA request. If the ENABLE control bit is not specified application should use the BSP_DMA_Start API to enable the DMA transfer.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

38

To learn more about the parameter values passed to this API, refer to the
Z8F6482 Series Product Specification.

Correct Usage

For proper operation of this macro the AUTOINC bit in the DMA global
control register (DMACTL) must be set to 1.

Z8 Encore! XP® F6482 Series API Programmer's Reference Manual

zilog
Embedded in Life
An■IXYS Company

39

BSP_DMA_SETUPLL

Prototype

```
void BSP_DMA_SetupLL( UINT16 Base, DMA_DESC * pDescLL );
```

Parameters

Base   Specifies which DMA channel is being configured for a
            data transfer operation using a linked list of DMA
            descriptors. Base must be in the range of BSP_DMA0 to
            BSP_DMA3.

pDescLL   Pointer to the first (or only)DMA descriptor in the chain of
            descriptors. This pointer should reference an array of one
            or more DMA_DESC data structures that specify each of the
            individual operations within the overall transfer.

Return Value

None.

Description

The BSP_DMA_SetupLL macro is used to configure the specified DMA
channel's special function registers for a pending DMA transfer using a
linked list of DMA descriptors.

After calling this macro the first descriptor referenced by the pDescLL
parameter will be transferred into the special function registers of the
DMA channel specified by the value of the Base parameter. Additionally
if the ENABLE bit is set in the Ctl1 member of the first DMA descriptor
the first stage of the DMA transfer will be enabled.

To learn more about DMA transfers using linked lists of descriptors, refer
to the Z8F6482 Series Product Specification.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

40

Correct Usage

For proper operation of this macro the AUTOINC bit in the DMA global control register (DMACTL) must be set to 1. Also, the linked list of DMA descriptors must be aligned in memory such that each descriptor's start address is evenly divisible by 8.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An ∎IXYS Company

41

BSP_DMA_START

Prototype

```
void BSP_DMA_Start( UINT16 Base );
```

Parameters

Base   Specifies which DMA channel is being activated. Base
              must be in the range of BSP_DMA0 to BSP_DMA3.

Return Value

None.

Description

The BSP_DMA_Start API is used to set the ENABLE bit in the
DMAxCTL1 Special Function Register of the DMA channel that corre-
sponds to the Base parameter. As a result, the DMA transfer specified by
the current configuration of that channel's DMA special function registers
will be enabled.

Correct Usage

When either the BSP_DMA_Setup or BSP_DMA_SetupLL routines are
used to configure the special function registers of a given DMA channel,
the DMA transfer can be immediately enabled by specifying the
DMA_ENABLE flag in the Ctl1 parameter/structure member. If the DMA
channel is not enabled at the time the DMA special function registers are
configured than this API may be used to enable the DMA transfer.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

42

BSP_DMA_ABORT

Prototype

```
void BSP_DMA_Abort( UINT16 Base );
```

Parameters

Base   Specifies which DMA channel is being stopped. Base must
              be in the range of BSP_DMA0 to BSP_DMA3.

Return Value

None.

Description

The BSP_DMA_Abort API is used to terminate an enabled/in-progress
DMA transfer regardless of whether the transfer uses direct SFR address-
ing or linked list control. After the DMA transfer has been aborted, the
specified DMA channel con be reconfigured to initiate a new DMA trans-
fer without having to reacquire the channel (i.e., the DMA abort operation
does not implicitly release the DMA channel).

Correct Usage

None.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An◼IXYS Company | **43**

BSP_DMA_GETCOUNT

Prototype

```
BSP_SIZE BSP_DMA_GetCount( UINT16 Base );
```

Parameters

Base   Specifies the DMA channel for which the remaining
            transfer count is sought. Base must be in the range of
            `BSP_DMA0` to `BSP_DMA3`.

Return Value

A value between 0 and the transfer count specified in the call to
`BSP_DMA_Setup` (or the `Cnt` member of the active DMA descriptor)
indicating the number of bytes of data that have not yet been transferred.

Description

This function returns the number of bytes remaining in the current DMA
transfer operation. When using linked lists, the returned count does not
include the remaining transfer counts of inactive DMA descriptors.

If the DMA transfer has completed this function returns the value 0 indi-
cating there are no more bytes to be transferred.

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

44

BSP_DMA_GETCOUNTLL

Prototype

```
BSP_SIZE BSP_DMA_GetCount
(
 UINT16 Base,
 DMA_DESC * pDescLL
);
```

Parameters

Base   Specifies the DMA channel for which the remaining
            transfer count is sought. Base must be in the range of
            BSP_DMA0 to BSP_DMA3.

pDescLL   Pointer to the first (or only)DMA descriptor in the chain of
            descriptors. This pointer should reference an array of one
            or more DMA_DESC data structures that specify each of the
            individual operations within the overall transfer.

Return Value

A value between 0 and the sum of the transfer counts from all descriptors
within the linked list specified in the call to BSP_DMA_SetupLL indicat-
ing the number of bytes of data that have not yet been transferred.

Description

This function returns the number of bytes remaining in the current DMA
transfer operation that is composed of one or more descriptors in a linked
list descriptor chain. If the DMA transfer has completed this function
returns the value 0 indicating there are no more bytes to be transferred.

If the lined list of descriptors loops back to the first descriptor in the
chain, then the maximum transfer count only includes a single pass
through the list. For example if the linked list includes 3 descriptors, in
which the first indicates a transfer of 30 bytes; the second a transfer of 20

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

45

bytes and the third specifies a transfer-in-list address that matches that of the first descriptor, then the maximum transfer count of the chain is 50 bytes and this function will return a value between 0 and 50.

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

46

Event System API Reference

The event system module is used to coordinate access to event system channels. Most of the event system API is implemented using macros to minimize code footprint and execution time.

Event System Functions in the BSP API

The Event System API implements the following functions and macros:

- <u>BSP_Event_Acquire</u> – see page 47
- <u>BSP_Event_Release</u> – see page 49
- <u>BSP_Event_Connect</u> – see page 51
- <u>BSP_Event_Disconnect</u> – see page 53

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An ⬛IXYS Company

47

BSP_EVENT_ACQUIRE

Prototype

```
BSP_STATUS BSP_Event_Acquire( UINT8 Chan, UINT8 Src );
```

Parameters

Chan   Specifies which event system channel is being acquired.
            Chan must be in the range of ES_SSA_CH0 to
            ES_SSA_CH7.

Src   Identifies which peripheral or GPIO pin will trigger event
            signals on the channel specified by the Chan parameter.
            The ..\Inc\Z8F6482_Event_SFR.h source file
            contains a set of macro definitions that identify each of the
            possible Z8F6482 event system signal sources (i.e.,
            ESCHxSRC). The value of the Src parameter should match
            one of these macros (prefixed with ES_CHSRCSEL_).

Return Value

BSP_ERR_SUCCESS is returned if no errors occur.

BSP_IN_USE is returned if the event system channel corresponding to the
Chan parameter has already been acquired from a previous call to this
API.

Description

Applications call this API to request (exclusive) access to the event sys-
tem channel corresponding to the Chan parameter. After an event system
channel has been acquired, it may not be reacquired until it is released by
calling the BSP_Event_Release API.

If this API returns BSP_ERR_SUCCESS the application may connect the
requested channel to one or more destinations allowing the event system

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

48

signal source to trigger an action on each of the destinations to which it is connected.

Correct Usage

This API does not enable or otherwise configure the event system source identified by the `Src` parameter. It is the application's responsibility to configure and enable the peripheral device(s) and/or GPIO pin(s) that are connected together by the event system.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An◻IXYS Company

49

BSP_EVENT_RELEASE

Prototype

```
#define BSP_Event_Release( Chan ) \
{ \
 ESSSA = Chan; \
 ESSSD = ES_CHSRCSEL_DBLD; \
}
```

Parameters

Chan   Specifies which event system channel is being released.
              Chan must be in the range of ES_SSA_CH0 to
              ES_SSA_CH7.

Return Value

None.

Description

The BSP_Event_Release macro is used to return the specified event
system channel to the system for subsequent reallocation via the
BSP_Event_Acquire API.

Correct Usage

When an event system channel is released, the event system signal source
is disconnected from the channel but the destination peripheral(s) and/or
GPIO pin(s) remain connected to the inactive channel. To prevent the
event system from generated unwanted triggers when the channel being
released is reacquired, applications should disconnect all event system
destination device(s) from the channel being released.

This API does not disable or reconfigure any source or destination
device(s) or GPIO pin(s) connected to the channel being released. If

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

50

appropriate, applications should disable source and destination device(s) and/or reconfigure GPIO pin(s) associated with the event system channel being released.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

z i l o g
Embedded in Life
An ∎IXYS Company

51

BSP_EVENT_CONNECT

Prototype

```
#define BSP_Event_Connect( Chan, Dst ) \
{ \
 ESDSA = Dst; \
 ESDSD = ( ES_DST_CON | Chan); \
}
```

Parameters

Chan   Specifies the event system channel to which the destination
            device (Dst) is being connected. Chan must be in the range
            of ES_DST_CHSEL_0 to ES_DST_CHSEL_7.

Dst   Identifies the peripheral device or GPIO pin to be added to
            the set of event system destinations for the channel
            corresponding to the Chan parameter. The
            ..\Inc\Z8F6482_Event_SFR.h source file contains a
            set of macro definitions that identify each of the possible
            Z8F6482 event system destinations (i.e., ESDSA). The
            value of the Dst parameter should match one of these
            macros (prefixed with ES_DSA_).

Return Value

None.

Description

This macro is used to connect the peripheral device or GPIO pin corre-
sponding to the Dst parameter to the event system channel specified by
the Chan parameter. Consequently, when the event system signal source
for that channel generates an event the destination corresponding to the
Dst parameter will be triggered (as will all other destinations connected

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

52

to the same channel and all other destinations connected to the same
source on different channels).

Correct Usage

It is not possible to connect a destination peripheral or GPIO pin to more
than one channel at a time. If this macro is invoked multiple times, and
each invocation specifies a different channel, then the destination corre-
sponding to the Dst parameter will only be connected to the channel
specified on the last invocation of this macro.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An◼IXYS Company

53

BSP_EVENT_DISCONNECT

Prototype

```
#define BSP_Event_Disconnect( Dst ) \
{ \
 ESDSA = Dst; \
 ESDSD = ( 0 ); \
}
```

Parameters

Dst   Identifies the peripheral device or GPIO pin to be removed
                from the set of event system destinations for the channel
                corresponding to the Chan parameter. The
                ..\Inc\Z8F6482_Event_SFR.h source file contains a
                set of macro definitions that identify each of the possible
                Z8F6482 event system destinations (ESDSA). The value of
                the Dst parameter should match one of these macros
                (prefixed with ES_DSA_).

Return Value

None.

Description

This macro is used to disconnect the peripheral device or GPIO pin corre-
sponding to the Dst parameter to the event system channel specified by
the Chan parameter.

Correct Usage

It is not necessary for applications to invoke this macro before switching
an event system destination device or GPIO pin to a different channel (via
BSP_Event_Connect), because an event system destination may only
be connected to 1 channel at any given time. However it is strongly rec-

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

54

ommended that all event system destination device(s) and/or GPIO pin(s) be disconnected from an event system channel prior to connecting that channel to a different event system signal source. Otherwise the destination device(s) and GPIO pin(s) could be inadvertently triggered by the new event system signal source for that channel.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

55

General Purpose Input/Output API Reference

The GPIO module is used to simplify the process of configuring and manipulating sets of GPIO port pins. Most of the GPIO API is implemented using macros to minimize code footprint and execution time.

GPIO Functions in the BSP API

The GPIO API implements the following functions and macros:

-
-
-
-
-
-

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

56

BSP_GPIO_ALTFUNC

Prototype

```
void BSP_GPIO_AltFunc( rom BSP_GPIO_CFG * pCfg );
```

Parameter

pCfg   References an array of `BSP_GPIO_CFG` structures that
            specifies one or more sets of GPIO port pins to be
            configured for alternate function mode. The array must be
            terminated with an entry in which the Port structure
            member is 0. Additionally, the structure must be located in
            Z8F6482 series read-only memory space (i.e., Flash). To
            learn more, see the GPIO Data Structures in the BSP API
            section on page 165.

Return Value

None.

Description

This routine allows application programmers to configure multiple sets of
GPIO port pins in alternate function mode. Typically, GPIO port pins are
configured for one of four possible alternate function modes so that the
pin can be used by an integrated peripheral device such as the SPI or
UART controllers. Many port pins are multiplexed between different
peripherals and the particular alternate subfunction selected determines
which peripheral will use the pin(s).

Most BSP peripheral drivers accept a configuration parameter of type
`rom BSP_GPIO_CFG *`, which determines the set of GPIO pins and the
alternate function modes that will be used by the peripheral. In this
instance, the application is not required to explicitly call the
`BSP_GPIO_AltFunc` API; the application is only required to define

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An ◼ IXYS Company **57**

`BSP_GPIO_CFG` structure. In other instances (such as configuring a GPIO port pin to be used as an event system output) the application must define the `BSP_GPIO_CFG` structure and explicitly call the `BSP_GPIO_AltFunc` API.

Correct Usage

Each entry in the array of `BSP_GPIO_CFG` structures referenced by the `pCfg` parameter pertains to a set of one or more GPIO pins in the same port; and each pin grouping must use the exact same AFS1 and AFS2 configuration. As an example consider the following fictitious definition:

```
rom BSP_GPIO_CFG GpioCfg[ 4 ] =
{
 {BSP_GPIO_PORT_C, (BIT1 | BIT0), 1, 0},
 {BSP_GPIO_PORT_C, (GPIOC_ESOUT0), 0, 1},
 {BSP_GPIO_PORT_D, (GPIOD_C0OUT), 0, 1},
 {0,0,0,0}
};
```

The sample `GpioCfg` array is configuring 3 sets of GPIO pins (the 4th entry contains a GPIO port value of 0 indicating the end of the array). The first 2 entries are configuring GPIO pins in port C for different alternate subfunctions. PC1 and PC0 are being configured for AFS1=1 and AFS2=0 but PC6 (`GPIOC_ESOUT0`) is being configured for subfunction AFS=0 and AFS2=1. Although these pins are all in Port C, two entries in the `GpioCfg` array are required, because the pin sets use different alternate subfunction settings. GPIO pin PD7 (`GPIOD_COUT`) is also being configured for AFS=0 and AFS2=1 (such as PC6), but because PD7 is in a different port, a separate `GpioCfg` array entry is required.

The `..\Inc\Z8F8482_GPIO_SFR.h` header file contains macro definitions for various GPIO pins that may be used instead of bit values. For example, the first entry in the `GpioCfg` array above used the macros BIT1 (0x02) and BIT0 (0x01) but the definition could have used `GPIOC_ANA5_C0INN` and `GPIOC_ANA4_VBIAS_C0INP` instead.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

58

To learn more about GPIO alternate functions, refer to the Z8F6482
Series Product Specification.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An ◻ IXYS Company

59

BSP_GPIO_DD_IN

Prototype

```
void BSP_GPIO_DD_In( UINT16 Port, UINT8 BitMask );

#define BSP_GPIO_DD_In( Port, BitMask ) \
{ \
 GPIO_PxADDR( (Port) ) = __DATA_DIRECTION; \
 GPIO_PxCTL( (Port) ) |= (BitMask); \
}
```

Parameters

Port   Specifies the GPIO port whose pin(s) are being configured
              as inputs. The value of the Port parameter must be
              between BSP_GPIO_PORT_A to BSP_GPIO_PORT_J
              (excluding BSP_GPIO_PORT_I which is not defined for
              the Z8F6482 Series).

BitMask   Specifies the bit value of one or more pins within the port
              that are being configured as general purpose inputs.

Return Value

None.

Description

The BSP_GPIO_DD_In macro is used to configure the specified GPIO
port pin(s) as inputs.

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

z i l o g
Embedded in Life
An ◻ IXYS Company

60

BSP_GPIO_DD_OUT

Prototype

```
void BSP_GPIO_DD_Out( UINT16 Port, UINT8 BitMask );

#define BSP_GPIO_DD_Out( Port, BitMask ) \
{ \
 GPIO_PxADDR( (Port) ) = __DATA_DIRECTION; \
 GPIO_PxCTL( (Port) ) &= ~(BitMask); \
}
```

Parameters

Port   Specifies the GPIO port whose pin(s) are being configured
            as outputs. The value of the Port parameter must be
            between BSP_GPIO_PORT_A to BSP_GPIO_PORT_J
            (excluding BSP_GPIO_PORT_I, which is not defined for
            the Z8F6482 Series).

BitMask   Specifies the bit value of one or more pins within the port
            that are being configured as general purpose outputs.

Return Value

None.

Description

The BSP_GPIO_DD_Out macro is used to configure the specified GPIO
port pin(s) as outputs.

Correct Usage

None.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An ◼ IXYS Company

61

BSP_GPIO_SET

Prototype

```
void BSP_GPIO_Set( UINT16 Port, UINT8 BitMask );

#define BSP_GPIO_Set( Port,BitMask ) \
{ \
 GPIO_PxOUT( (Port) ) |= (BitMask); \
}
```

Parameters

Port   Specifies the GPIO port whose pin(s) are to be driven to
             the logic 1 (set) state. The value of the Port parameter
             must be between BSP_GPIO_PORT_A to
             BSP_GPIO_PORT_J (excluding BSP_GPIO_PORT_I,
             which is not defined for the Z8F6482 Series).

BitMask   Specifies the bit value of one or more pins within the port
             to be driven high (logic 1).

Return Value

None.

Description

The BSP_GPIO_Set macro is used to drive a logic 1 to the specified
GPIO port pin(s).

Correct Usage

The specified port pin(s) must have previously been configured for output
mode (see the BSP_GPIO_DD_Out API on page 60).

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An ◻IXYS Company

62

BSP_GPIO_CLEAR

Prototype

```
void BSP_GPIO_Clear( UINT16 Port, UINT8 BitMask );

#define BSP_GPIO_Clear( Port,BitMask ) \
{ \
 GPIO_PxOUT( (Port) ) &= ~(BitMask); \
}
```

Parameters

Port   Specifies the GPIO port whose pin(s) are to be driven to
            the logic 0 (clear) state. The value of the Port parameter
            must be between BSP_GPIO_PORT_A to
            BSP_GPIO_PORT_J (excluding BSP_GPIO_PORT_I
            which is not defined for the Z8F6482 Series).

BitMask   Specifies the bit value of one or more pins within the port
            to be driven low (logic 0).

Return Value

None.

Description

The BSP_GPIO_Clear macro is used to drive a logic 0 to the specified
GPIO port pin(s).

Correct Usage

The specified port pin(s) must have previously been configured for output
mode (see the BSP_GPIO_DD_Out API on page 60).

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

z i l o g
Embedded in Life
An ◘IXYS Company **63**

BSP_GPIO_TOGGLE

Prototype

```
void BSP_GPIO_Toggle( UINT16 Port, UINT8 BitMask );

#define BSP_GPIO_Toggle( Port,BitMask ) \
{ \
 GPIO_PxOUT( (Port) ) ^= (BitMask); \
}
```

Parameters

Port   Specifies the GPIO port whose pin(s) are to be inverted; i.e., port pin(s) currently driven to a logic 1 state will be driven to a logic 0 and vice versa. The value of the Port parameter must be between BSP_GPIO_PORT_A to BSP_GPIO_PORT_J (excluding BSP_GPIO_PORT_I which is not defined for the Z8F6482 Series).

BitMask   Specifies the bit value of one or more pins within the port to be toggled.

Return Value

None.

Description

The BSP_GPIO_Toggle macro is used to invert the state of the specified GPIO port pin(s).

Correct Usage

The specified port pin(s) must have previously been configured for output mode (see the BSP_GPIO_DD_Out API on page 60).

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

64

Inter-Integrated Circuit API

The BSP I²C API contains support for an I²C device acting either as a I²C Master or as an I²C Slave (but not as both simultaneously). The device can be configured to operate in Poll, Interrupt, or DMA modes.

I²C Functions in the BSP API

The I²C API implements the following functions:

- BSP_I2C_Init – see page 66

- BSP_I2C_Stop – see page 67

- I2C_Setup Functions – see page 68
    – I2C_Setup_Master_Polling
    – I2C_Setup_Slave_Polling
    – I2C_Setup_Master_Irq
    – I2C_Setup_Slave_Irq
    – I2C_Setup_Master_Dma
    – I2C_Setup_Slave_Dma

- I2C Transfer and Receive Functions – see page 69
    – I2C_Master_Transmit
    – I2C_Master_Receive
    – I2C_Slave_Transmit
    – I2C_Slave_Receive

- I2C_Set_Slave_Buffer(): – see page 71
    – I2C_Set_Slave_TxBuffer
    – I2C_Set_Slave_RxBuffer

- I2C_Brg – see page 72
    – UINT16 I2C_Brg

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

65

- [BSP_I2C_General_Call_Address](#) – see page 73

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

66

zilog
Embedded in Life
An ◘ IXYS Company

BSP_I2C_INIT

Prototype

```
BSP_STATUS BSP_I2C_Init (BSP_AES * pAES )
```

Parameter

pI2C   A pointer to a BSP_I2C structure that configures I²C
              operations; see the I2C Structures and Unions in the BSP
              API section on page 168.

Return Value

BSP_ERR_SUCCESS if no errors occur.

BSP_ERR_IN_USE if the I²C has already been initialized and
BSP_I2C_Stop() was not subsequently called; or if a DMA channel is
requested which was previously initialized.

BSP_ERR_INVALID_PARAM for some invalid configurations of the
BSP_I2C structure.

Description

Use to configure the I²C API before first use. May also be used to recon-
figure the I²C API for a different mode (Polling, Irq, or Dma), switch
between master and slave, or to change the baud rate.

Correct Usage

This function should not be called if the I²C has been previously initial-
ized; it should only be called after first calling BSP_I2C_Stop(); other-
wise, an error status will be returned.

Z8 Encore! XP® F6482 Series API Programmer's Reference Manual

zilog
Embedded in Life
An ∎IXYS Company

67

BSP_I2C_STOP

Prototype

```
BSP_STATUS BSP_I2C_Stop (void )
```

Parameters

n/a

Return Value

BSP_ERR_SUCCESS if no errors occur.

BSP_ERR_BUSY if called while an transaction is in progress.

Description

This API can be used to shut down the I^2C gracefully when it is no longer required, and can also be used to undo a previous initialization prior to reconfiguring the I^2C.

Correct Usage

This function should not be called if an I^2C transaction is in progress.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An ◻IXYS Company

68

I2C_SETUP FUNCTIONS

Prototypes

```
reentrant BSP_STATUS I2C_Setup_Master_Polling(BSP_I2C *pI2c);
reentrant BSP_STATUS I2C_Setup_Slave_Polling(BSP_I2C *pI2c);

reentrant BSP_STATUS I2C_Setup_Master_Irq(BSP_I2C *pI2c);
reentrant BSP_STATUS I2C_Setup_Slave_Irq(BSP_I2C *pI2c);

reentrant BSP_STATUS I2C_Setup_Master_Dma(BSP_I2C *pI2c);
reentrant BSP_STATUS I2C_Setup_Slave_Dma(BSP_I2C *pI2c);
```

Description

The functions should never be called directly. Rather, set the fpSetup member of the appropriate structure in I2C_CFG union to select the desired mode of operation. Any error return value will itself be returned by BSP_I2C_Init().

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

z i l o g
Embedded in Life
An ◻ IXYS Company

69

I²C TRANSFER AND RECEIVE FUNCTIONS

Prototypes

```
BSP_STATUS I2C_Master_Transmit (UINT16 slaveAddress,
 UINT8 tenBitSlave,
 HANDLE txBuf,
 BSP_SIZE len);

BSP_STATUS I2C_Master_Receive (UINT16 slaveAddress,
 UINT8 tenBitSlave,
 HANDLE rxBuf,
 BSP_SIZE len);


BSP_STATUS I2C_Slave_Transmit(HANDLE txBuf,
 BSP_SIZE len);

BSP_STATUS I2C_Slave_Receive(HANDLE rxBuf,
 BSP_SIZE len);
```

Parameter

slaveAddress	For the master transmit and receive operations, the address of the I²C slave to use.
tenBitSlave	For the master transmit and receive operations, nonzero if the address is a 10 bit address.
txBuf, rxBuf	The address of the message to transmit or to place the message received.
len	The size in bytes of the message buffer.

Return Value

BSP_ERR_SUCCESS if no errors occur.

BSP_ERR_BUSY if called while an transaction is in progress.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An IXYS Company

70

> ▶ **Note:** Any error in the message transmission or reception is passed as the status argument to the `fpXferDone` callback function.

Description

These are the principal calls to initiate an I²C transfer. In all cases, when the call returns `BSP_ERR_SUCCESS` indicating that the transfer was accepted, the result of the transfer will be passed through the `fpXfer-Done` callback function passed in the `BSP_I2C` structure used to initiate the I²C API.

> ▶ **Note:** These are not actually functions but macros defined in `BSP_I2C.h`.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

z i l o g
Embedded in Life
An ◼ IXYS Company

71

I2C_SET_SLAVE_BUFFER();

Prototype

```
BSP_STATUS I2C_Set_Slave_Buffer(HANDLE Buf,
 BSP_SIZE len,
 UINT8 transmit);
```

Parameters

Buf   The buffer to transmit from or receive into.

len   The size of the buffer in bytes.

transmit   nonzero if Buf is a transmit buffer.

Return Value

BSP_ERR_SUCCESS if no errors occur.

BSP_ERR_BUSY if called while an transaction is in progress.

Description

Use this API on an I^{2}C slave application if you do not know whether the master will issue a transmit or a receive request first.

Correct Usage

The macros I2C_Set_Slave_TxBuffer and I2C_Set_Slave_RxBuffer are provided, which basically hide the transmit parameter in the function call.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

72

I2C_BRG

Prototype

```
UINT16 I2C_Brg(UINT32 clockRate, UINT32 baudRate);
```

Parameters

clockRate   The system clock rate (in ticks per second).

baudRate   The desired baud rate.

Return Value

The value to place in the BSP_I2C structure to obtain the desired baudrate.

> **Note:** I2C_Brg is actually a macro.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An◼IXYS Company

73

BSP_I2C_GENERAL_CALL_ADDRESS

Prototype

```
void BSP_I2C_General_Call_Address(UINT8 enable);
```

Parameter

enable   Nonzero to enable general call addresses in a slave.

Description

Use BSP_I2C_General_Call_Address() to enable receiving messages addressed to the I^2C General Call Address (7-bit address of 0) in an I^2C Slave application. By default, receiving such messages is disabled.

> **Note:** BSP_I2C_General_Call_Address is actually a macro.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

74

Interrupt Controller API Reference

The BSP interrupt controller module provides a set of macros to simplify interrupt configuration and processing. These macros are defined in the Z8F6482_IRQ_SFR.h and BSP_IRQ.h header files. The macro definitions in the Z8F6482_IRQ_SFR.h header file identify all of the Z8F6482 interrupt sources monitored by the interrupt controller. The macros defined in the BSP_IRQ.h header file are described in the remainder of this section.

IRQ Macros in the BSP API

The BSP_IRQ.h header file defines the following macros:

-
-
-
-
-
-
-
-
-
-
-
-
-

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An◻IXYS Company

75

BSP_IRQXEN_DBLD

Prototype

```
#define BSP_IRQxEN_DBLD( Base, Bit ) \
{ \
 IRQ_ENH((Base)) &= ~(Bit); \
 IRQ_ENL((Base)) &= ~(Bit); \
}
```

Parameters

Base   Specifies which bank of interrupt control registers contains
            the interrupt signal(s) specified by the Bit parameter. The
            value of Base must be one of BSP_IRQ0, BSP_IRQ3.

Bit   Specifies the bit value of one or more interrupt signals to
            be disabled within the bank specified by the Base
            parameter.

Return Value

None.

Description

This macro is used to disable one or more interrupt sources within the
specified bank. For example, to disable the UART0 transmit and receive
interrupt signals in bank 0, application programs may use the following
macro invocation:

```
BSP_IRQxEN_DBLD( BSP_IRQ0, IRQ_U0RX | IRQ_U0TX );
```

To learn more about the interrupt controller, refer to the Z8F6482 Series
Product Specification.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

76

Correct Usage

None.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An ◻ IXYS Company

77

BSP_IRQXEN_LO

Prototype

```
#define BSP_IRQxEN_LO( Base, Bit ) \
{ \
 IRQ_ENH((Base)) &= ~(Bit); \
 IRQ_ENL((Base)) |= (Bit); \
}
```

Parameters

Base   Specifies which bank of interrupt control registers contains
            the interrupt signal(s) specified by the Bit parameter. The
            value of Base must be one of BSP_IRQ0, BSP_IRQ3.

Bit   Specifies the bit value of one or more interrupt signals to
            be enabled at low priority within the bank specified by the
            Base parameter.

Return Value

None.

Description

This macro enables one or more interrupt sources within the specified
bank as a low priority interrupt(s). For example, to configure the UART0
transmit and receive interrupt signals as low priority interrupts, applica-
tion programs may use the following macro invocation:

BSP_IRQxEN_LO(BSP_IRQ0, IRQ_U0RX | IRQ_U0TX);

To learn more about the interrupt controller, refer to the Z8F6482 Series
Product Specification.

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An ∎IXYS Company

78

BSP_IRQXEN_NOM

Prototype

```
#define BSP_IRQxEN_NOM( Base, Bit ) \
{ \
 IRQ_ENH((Base)) |= (Bit); \
 IRQ_ENL((Base)) &= ~(Bit); \
}
```

Parameters

Base   Specifies which bank of interrupt control registers contains
            the interrupt signal(s) specified by the Bit parameter. The
            value of Base must be one of BSP_IRQ0, BSP_IRQ3.

Bit   Specifies the bit value of one or more interrupt signals to
            be enabled at nominal priority (between low and high
            priority) within the bank specified by the Base parameter.

Return Value

None.

Description

This macro enables one or more interrupt sources within the specified
bank as a medium (nominal) priority interrupt(s). For example, to config-
ure the UART0 transmit and receive interrupt signals as medium priority
interrupts, application programs may use the following macro invocation:

```
BSP_IRQxEN_NOM( BSP_IRQ0, IRQ_U0RX | IRQ_U0TX );
```

To learn more about the interrupt controller, refer to the Z8F6482 Series
Product Specification.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

79

Correct Usage

None.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

80

BSP_IRQXEN_HI

Prototype

```
#define BSP_IRQxEN_HI( Base, Bit ) \
{ \
 IRQ_ENH((Base)) |= (Bit); \
 IRQ_ENL((Base)) |= (Bit); \
}
```

Parameters

Base   Specifies which bank of interrupt control registers contains the interrupt signal(s) specified by the Bit parameter. The value of Base must be one of BSP_IRQ0, BSP_IRQ3.

Bit   Specifies the bit value of one or more interrupt signals to be enabled at high priority within the bank specified by the Base parameter.

Return Value

None.

Description

This macro enables one or more interrupt sources within the specified bank as a high priority interrupt(s). For example, to configure the UART0 transmit and receive interrupt signals as high priority interrupts, application program may use the following macro invocation:

```
BSP_IRQxEN_HI( BSP_IRQ0, IRQ_U0RX | IRQ_U0TX );
```

To learn more about the interrupt controller, refer to the Z8F6482 Series Product Specification.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

81

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

z i l o g
Embedded in Life
An ∎ IXYS Company

82

BSP_IRQX_CLR

Prototype

```
#define BSP_IRQx_CLR( Base, Bit ) \
{ \
 IRQ_REQ((Base)) &= ~(Bit); \
}
```

Parameters

Base   Specifies which bank of interrupt control registers contains
              the interrupt signal(s) specified by the Bit parameter. The
              value of Base must be one of BSP_IRQ0, BSP_IRQ3.

Bit   Specifies the bit value of one or more interrupt signals to
              be dismissed (i.e., cleared) within the bank specified by the
              Base parameter.

Return Value

None.

Description

This macro is used to dismiss an interrupt signal that has been latched in
the interrupt controller. After an interrupt signal has been latched within
the interrupt controller a vectored interrupt will occur (preempting the
foreground thread of execution) as soon as interrupts are enabled (both
global interrupts and the target interrupt signal). This macro is used to dis-
miss the specified interrupt signal(s). For example, to dismiss pending
UART0 transmit and receive interrupt signals, application programs may
use the following macro invocation:

```
BSP_IRQx_CLR( BSP_IRQ0, IRQ_U0RX | IRQ_U0TX );
```

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

83

To learn more about the interrupt controller, refer to the <u>Z8F6482 Series Product Specification</u>.

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

84

BSP_IRQ_ES_FALLING

Prototype

```
#define BSP_IRQ_ES_FALLING( Bit ) \
{ \
 IRQES &= ~(Bit); \
}
```

Parameters

Bit   Specifies the bit value of one or more interrupt signals to
             be configured for falling edge interrupts.

Return Value

None.

Description

Some GPIO port pins can be configured to generate either rising edge or
falling edge interrupts. This macro is used to select falling edge interrupt
generation for the set of interrupt sources corresponding to the Bit
parameter. For example, to configure PA4 and PD3 to generate falling
edge interrupts, application programs may use the following macro invo-
cation:

```
BSP_IRQ_ES_FALLING( IRQ_PAD4 | IRQ_PAD3 );
```

To learn more about the interrupt controller, refer to the Z8F6482 Series
Product Specification.

Correct Usage

None.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An∎IXYS Company

85

BSP_IRQ_ES_RISING

Prototype

```
#define BSP_IRQ_ES_RISING( Bit ) \
{ \
 IRQES |= (Bit); \
}
```

Parameters

Bit   Specifies the bit value of one or more interrupt signals to
be configured for rising edge interrupts.

Return Value

None.

Description

Some GPIO port pins can be configured to generate either rising edge or
falling edge interrupts. This macro is used to select rising edge interrupt
generation for the set of interrupt sources corresponding to the Bit
parameter. For example, to configure PA4 and PD3 to generate rising
edge interrupts, application programs may use the following macro invo-
cation:

```
BSP_IRQ_ES_RISING( IRQ_PAD4 | IRQ_PAD3 );
```

To learn more about the interrupt controller, refer to the Z8F6482 Series
Product Specification.

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An ◻ IXYS Company

86

BSP_IRQ_SS0_FIRST

Prototype

```
#define BSP_IRQ_SS0_FIRST( Bit ) \
{ \
 IRQSS0 &= ~(Bit); \
}
```

Parameters

Bit   Specifies the bit value of one or more interrupt signals with shared sources to be configured such that interrupts from the first source are selected and interrupts from the second source are ignored. Bit should be a combination of 1 or more of the following values: `IRQ_PA7LVD`, `IRQ_PA6C0`, `IRQ_PA5C1`, `IRQ_PAD4`, `IRQ_PAD3`, `IRQ_PAD2`, or `IRQ_PAD1`.

Return Value

None.

Description

Some Z8F6482 interrupt signals are shared between two sources. This macro is used to select the first alternate interrupt source to drive the shared interrupt request signal. For example either GPIO pin PA4 or PD4 can drive the `IRQ_PAD4` interrupt request signal. To select PA4 as the interrupt source that drives the PAD4 interrupt request signal, application programs may use the following macro invocation:

```
BSP_IRQ_SS0_FIRST( IRQ_PAD4 );
```

To learn more about the interrupt controller, refer to the Z8F6482 Series Product Specification.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

87

Correct Usage

None.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

z i l o g
Embedded in Life
An ◻ IXYS Company

88

BSP_IRQ_SS0_SECOND

Prototype

```
#define BSP_IRQ_SS0_SECOND( Bit ) \
{ \
 IRQSS0 |= (Bit); \
}
```

Parameters

Bit   Specifies the bit value of one or more interrupt signals with shared sources to be configured such that interrupts from the second source are selected and interrupts from the first source are ignored. Bit should be a combination of 1 or more of the following values: `IRQ_PA7LVD`, `IRQ_PA6C0`, `IRQ_PA5C1`, `IRQ_PAD4`, `IRQ_PAD3`, `IRQ_PAD2`, or `IRQ_PAD1`.

Return Value

None.

Description

Some Z8F6482 interrupt signals are shared between two sources. This macro is used to select the second alternate interrupt source to drive the shared interrupt request signal. For example either GPIO pin PA4 or PD4 can drive the `IRQ_PAD4` interrupt request signal. To select PD4 as the interrupt source that drives the PAD4 interrupt request signal, application programs may use the following macro invocation:

```
BSP_IRQ_SS0_SECOND( IRQ_PAD4 );
```

To learn more about the interrupt controller, refer to the [Z8F6482 Series Product Specification](#).

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

89

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

z i l o g
Embedded in Life
An ∎IXYS Company

90

BSP_IRQ_SS1_FIRST

Prototype

```
#define BSP_IRQ_SS1_FIRST( Bit ) \
{ \
 IRQSS1 &= ~(Bit); \
}
```

Parameters

Bit   Specifies the bit value of one or more interrupt signals with
               shared sources to be configured such that interrupts from
               the first source are selected and interrupts from the second
               source are ignored.

Return Value

None.

Description

This macro is similar to the BSP_IRQ_SS0_FIRST macro but is to be
used when selecting the interrupt source for the IRQ_PC3DMA3 or
IRQ_PC2DMA2 interrupt signal. To learn more, see the description of the

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An ■ IXYS Company

91

BSP_IRQ_SS1_SECOND

Prototype

```
#define BSP_IRQ_SS1_SECOND( Bit ) \
{ \
 IRQSS1 |= (Bit); \
}
```

Parameters

Bit   Specifies the bit value of one or more interrupt signals with shared sources to be configured such that interrupts from the second source are selected and interrupts from the first source are ignored.

Return Value

None.

Description

This macro is similar to the BSP_IRQ_SS0_SECOND macro but is to be used when selecting the interrupt source for the IRQ_PC3DMA3 or IRQ_PC2DMA2 interrupt signal. To learn more, see the description of the BSP_IRQ_SS0_SECOND macro on page 88.

Correct Usage

None.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An ■ IXYS Company

92

BSP_IRQ_DISABLE

Prototype

```
#define BSP_IRQ_DISABLE( Flags ) \
{ \
 (Flags) = IRQCTL; \
 DI(); \
}
```

Parameters

Flags   An application-defined variable assigned the current value
of the interrupt control special function register (IRQCTL).
The Flags variable should be local to a function and reside
on the run time stack to allow nesting of interrupt disable/
enable macros.

Return Value

None. Global interrupts are disabled after execution of this macro.

Description

This macro is used to globally disable vectored interrupts. Applications
use this macro in conjunction with the BSP_IRQ_RESTORE macro to
implement a noninterruptible block of code (often referred to as a critical
section). Prior to disabling interrupts the state of the interrupt controller is
assigned to the Flags parameter allowing the BSP_IRQRESTOR macro to
restore the interrupt controller to the state it was in prior to executing the
BSP_IRQ_DISABLE macro.

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An◼IXYS Company   **93**

BSP_IRQ_RESTORE

Prototype

```
#define BSP_IRQ_RESTORE( Flags ) \
{ \
 IRQCTL = (Flags); \
}
```

Parameter:

Flags   Application defined variable whose current value is written
               to the global interrupt controller (IRQCTL). The Flags
               variable should be local to a function and reside on the run
               time stack to allow nesting of interrupt disable/enable
               macros. The value of the Flags variable should have
               previously been set by invoking the BSP_IRQ_DISABLE
               macro.

Return Value

None.

Description

This macro is used to return the interrupt controller to the state it was in
when the BSP_IRQ_DISABLE macro was invoked. If global interrupts
were enabled when the BSP_IRQ_DISABLE macro was invoked, this
macro will reenable global interrupts; otherwise global interrupts will
remain disabled after this macro is invoked.

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

94

Serial Peripheral Interface API Reference

The BSP SPI driver supports both the Serial Peripheral Interface (SPI) protocol and the Inter-IC Sound (I^2S) protocol. The SPI API includes support for Master, Multi-Master (only with the SPI protocol) and Slave modes of operation. The SPI driver can be configured to transfer data using CPU polling, interrupt control or DMA (only with the SPI protocol).

SPI Functions in the BSP API

The SPI API implements the following functions:

-
-
-
-
-

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An ∎IXYS Company

95

BSP_SPI_INIT

Prototype

```
BSP_STATUS BSP_SPI_Init( UINT8 Idx, BSP_SPI *pSpi );
```

Parameters

Idx   Specifies which SPI device is being initialized. Idx must
            be in the range of 0 to (BSP_NUM_SPI −1).

pSpi   Pointer to a BSP_SPI data structure that the application
            must initialize to effect a particular SPI (or I^{2}S) mode of
            operation. To learn more see the SPI Data Structures in the
            BSP API section on page 183.

Return Value

BSP_ERR_SUCCESS if no errors occur.

BSP_ERR_IN_USE if the SPI driver has already been initialized and
BSP_SPI_Stop() was not subsequently called; or if the SPI configura-
tion is requesting a DMA channel which is used by some other entity.

BSP_ERR_INVALID_PARAM is returned (with the debug version of the
BSP library) if the Idx parameter is larger than (BSP_NUM_SPI −1), if the
pSpi parameter is 0 or if pSpi references a configuration specifying an
invalid DMA channel.

▶   **Note:** Using the debug version of the BSP library will cause the USB driver to
            perform additional parameter validation which could result in a decrease in
            driver performance.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

z i l o g
Embedded in Life
An ◼ IXYS Company

96

Description

This routine must be called before applications call any other SPI API. After this API is called, it may not be called again until the BSP_SPI_Stop API is called. This routine will acquire and initialize the hardware resources (GPIO pins, interrupts, and DMA channels) necessary to enable the SPI controller in accordance with the configuration information supplied in the BSP_SPI structure.

Correct Usage

If the BSP_SPI configuration structure specifies that DMA is used for data transfer, be sure to call the BSP_DMA_Init API before calling this function.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

z i l o g
Embedded in Life
An ◼ IXYS Company

97

BSP_SPI_XFER

Prototype

```
BSP_STATUS BSP_SPI_Xfer
(
 UINT8 Idx,
 HANDLE hTxBuf,
 HANDLE hRxBuf,
 BSP_SIZE Len
);
```

Parameters

Idx	Specifies which SPI device to use for the data transfer operation. Idx must be in the range of 0 to (BSP_NUM_SPI –1).
hTxBuf	Application buffer containing data to be sent to the remote SPI device. For a receive-only transfer operation, set hTxBuf to NULLPTR (0).
hRxBuf	Application buffer to be filled with data received from the remote SPI device. For a transmit-only operation set hRxBuf to NULLPTR (0).
Len	Specifies the number of bytes of data to be transferred.

Return Value

BSP_ERR_BUSY is returned if the specified SPI device has not yet completed a previous transfer request.

BSP_ERR_INVALID_PARAM is returned (with the debug version of the BSP library) if the Idx parameter is out of range, both the hTxBuf and hRxBuf parameters are NULLPTR, if Len is 0 or both receive and transmit operations have been disabled on the SPI controller.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An∎IXYS Company

98

If there is no error, this API returns the number of bytes of data remaining in the transfer. For Poll Mode (synchronous) transfers, a return value other than 0 indicates the number of bytes that could not be transferred. For interrupt and DMA transfers (that complete asynchronously) the return value is typically equal to the Len parameter indicating there are Len bytes remaining in the data transfer operation.

Description

The SPI protocol allows for full duplex data transfer; i.e., one bit of information is received every time one bit of information is sent, because Rx and Tx data are transmitted over physically separate connections. Consequently, this API allows an application to request full-duplex data transfer. This API sends Len bytes of data from the application buffer referenced by hTxBuf to the remote SPI device and places Len bytes of data received from the remote SPI device into the application buffer specified by the hRxBuf parameter.

Applications may set either the hTxBuf parameter or the hRxBuf parameter to NULLPTR for half-duplex data transfer.

When the transfer operation complete the SPI driver will call the fpXferDone callback routine (part of the BSP_SPI structure passed to BSP_SPI_Init) to inform the application that the transfer has completed and report the final transfer status. If the fpXferDone member of the BSP_SPI structure is NULLPTR (0), then the SPI driver will not issue an application callback when the SPI transfer completes.

The function prototype of the transfer completion routine is shown the following code snippet:

```
reentrant void SpiXferDoneCallback
(
 UINT8 Idx,
 UINT8 Status,
 BSP_SIZE XferLen
);
```

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An▣IXYS Company

99

The `Idx` parameter indicates the SPI driver interface that is reporting the transfer completion. The value of `Idx` is between 0 and (BSP_NUM_SPI –1). The Status parameter reports the hardware status of the SPI controller at the time the transfer completed. The meaning of individual bits within the Status byte can be found in the description of the ESPI Status Register (ESPISTAT) in the [F6482 Series Product Specification (PS0294)](). The `XferLen` parameter reports the number of bytes in the original transfer that could not be completed because of the error indicated by the Status parameter. If the transfer completes without error the value of the `Xfer-Len` parameter is 0.

Correct Usage

SPI is a master-slave protocol; all data transfers must be initiated by the master. Consequently, if the BSP SPI driver is configured to operate in Slave Mode and the `BSP_SPI_Xfer` API is called to send and/or receive data, the actual data transfer cannot begin until the master initiates an SPI transfer. Similarly, an error-free slave data transfer operation will not terminate until exactly `Len` bytes have been transferred; such an issue can have a significant impact on the behavior of Slave Mode applications.

If the SPI driver is configured to operate in Slave Mode using CPU polling for data transfer, it is possible that this API will not return control back to the calling application for a significant (possibly unbound) amount of time. When using interrupts or DMA for slave data transfer the `BSP_SPI_Xfer` API will typically return immediately while the SPI transfer completes asynchronously in the background.

For all slave data transfer modes (polling, interrupt, and DMA), there could be a significant (possibly unbound) delay between the time the `BSP_SPI_Xfer` API is called to initiate an SPI data transfer until the application's `fpXferDone` callback routine is called. The length of the delay depends on how long the slave must wait for the master to send `Len` bytes of data. As an (exaggerated) example, if the slave expects 5 bytes of data and the master sends 1 byte every hour, then the slave transfer operation will complete in 5 hours.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

100

Applications are cautioned against using polling for slave-mode transfers unless the slave application knows that the master is about to initiate a transfer and knows exactly how much data should be transferred to/from the master. Otherwise SPI polling could consume the entire CPU bandwidth preventing the application from performing any other useful task. For this reason most slave applications will benefit from using either of the asynchronous data transfer methods (interrupt or DMA) which allow the application to use the CPU to perform application-specific tasks while waiting for the master to complete the SPI transfer.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An∎IXYS Company

101

BSP_SPI_RECEIVE

Prototype

```
BSP_STATUS BSP_SPI_Receive
(
 UINT8 Idx,
 HANDLE hBuf,
 BSP_SIZE Len
);

#define BSP_SPI_Receive( Idx, hBuf, Len ) \
BSP_SPI_Xfer( (Idx), NULLPTR, (hBuf), (Len) )
```

Parameters

Idx	Specifies which SPI device to use for the data transfer operation. Idx must be in the range of 0 to (BSP_NUM_SPI –1).
hBuf	Application buffer to be filled with data received from the remote SPI device.
Len	Specifies the maximum number of bytes to be received.

Return Value

Refer to the BSP_SPI_Xfer API for a description of the values returned by this macro.

Description

The BSP_SPI_Receive API is just a macro that calls the BSP_SPI_Xfer routine with the hTxBuf parameter set to NULLPTR (0). Refer to the BSP_SPI_Xfer API to learn more.

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

102

BSP_SPI_TRANSMIT

Prototype

```
BSP_STATUS BSP_SPI_Transmit
(
 UINT8 Idx,
 HANDLE hTxBuf,
 BSP_SIZE Len
);

#define BSP_SPI_Transmit( Idx, hBuf, Len ) \
BSP_SPI_Xfer( (Idx), (hBuf), NULLPTR, (Len) )
```

Parameters

Idx	Specifies which SPI device to use for the data transfer operation. Idx must be in the range of 0 to (BSP_NUM_SPI –1).
hBuf	Application buffer containing data to be sent to the remote SPI device.
Len	Specifies the number of bytes of data to be transmitted.

Return Value

Refer to the BSP_SPI_Xfer API for a description of the values returned by this macro.

Description

The BSP_SPI_Transmit API is just a macro that calls the BSP_SPI_Xfer routine with the hRxBuf parameter set to NULLPTR (0). Refer to the BSP_SPI_Xfer API to learn more.

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An ■IXYS Company

103

BSP_SPI_STOP

Prototype

```
BSP_STATUS BSP_SPI_Stop( UINT8 Idx );
```

Parameters

Idx   Specifies which SPI device is being returned to the system
                for subsequent reallocation. Idx must be in the range of 0
                to (BSP_NUM_SPI −1).

Return Value

BSP_ERR_SUCCESS if no errors occur.

BSP_ERR_IN_USE if an attempt is made to stop an SPI device that has
not yet completed a previous transfer request.

BSP_ERR_INVALID_PARAM is returned (with the debug version of the
BSP library) if the Idx parameter is out of range.

Description

If the SPI driver is idle, this API can be used to return the SPI controller to
a quiescent state. If this routine is successful, all BSP resources acquired
by the BSP_SPI_Init API are released and the BSP_SPI structure can
be modified to initiate a different SPI configuration by calling
BSP_SPI_Init.

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

104

Timer API Reference

The Timer module provides macros to start, stop, and read the count of one of the integrated timers. The BSP does not provide any services to configure timers, because this scenario typically only involves writing configuration values to the timer's special function registers. The `Z8F6482_TMR_SFR.h` header file contains macro definitions that can be used to configure timer registers instead of using numerical constants to improve software legibility.

TMR Macros in the BSP API

The `BSP_TMR.h` header file defines the following macros:

-
-
-

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

z i l o g
Embedded in Life
An ◼ IXYS Company

105

BSP_TMR_READ

Prototype

```
#define BSP_TMR_READ( Base, x ) \
{ \
 asm( "ATM" ); \
 x = TMR_CNT( (Base) ); \
}
```

Parameters

Base   Indicates which timer's 16-bit count register is being read.
            The value of Base must be one of BSP_TMR0, BSP_TMR1,
            or BSP_TMR2.

x   Application defined variable updated with the 16-bit count
            value of the specified timer.

Return Value

None.

Description

This macro provides an interrupt safe method of reading the 16-bit timer
count register while the timer is active. Reading the timer high byte regis-
ter latches the value of the timer low byte; however, if an interrupt occurs
between the time the high byte is read and the low byte is read, and if the
interrupt also reads the timer high and low byte registers, then the 16-bit
value obtained by the foreground task could be corrupted. The use of the
ATM assembly instruction in this macro ensures that the timer high and
low bytes are read in an atomic operation.

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

106

BSP_TMR_START

Prototype

```
#define BSP_TMR_START( Base ) \
{ \
 TMR_CTL1( (Base) ) |= TMR_TEN; \
}
```

Parameters

Base   Specifies which of the integrated timers is to be enabled.
                The value of Base must be one of BSP_TMR0, BSP_TMR1,
                or BSP_TMR2.

Return Value

None.

Description

This macro is used to enable the specified 16-bit reload timer. Depending
on how the timer was configured, the timer may or may not immediately
start counting. The application must configure the timer before using this
macro.

Correct Usage

None.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An ☐ IXYS Company

107

BSP_TMR_STOP

Prototype

```
#define BSP_TMR_START( Base ) \
{ \
 TMR_CTL1( (Base) ) &= ~TMR_TEN; \
}
```

Parameters

Base   Specifies which of the integrated timers is to be disabled.
              The value of Base must be one of BSP_TMR0, BSP_TMR1,
              or BSP_TMR2.

Return Value

None.

Description

This macro is used to disable (stop) the specified 16-bit reload timer.

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

108

Universal Asynchronous Receiver Transmitter API Reference

The BSP UART API includes support for standard UART mode, Multi-processor Mode, Local Interconnect Network (LIN) Mode, and DMX Mode. The UART driver can be configured to transfer data using CPU polling, interrupt control or DMA.

UART Functions in the BSP API

The UART API implements the following functions:

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

z i l o g
Embedded in Life
An❑IXYSCompany

109

BSP_UART_INIT

Prototype

```
BSP_STATUS BSP_UART_Init( UINT8 Idx, BSP_UART * pCfg );
```

Parameters

Idx   Specifies which UART device is being initialized. Idx
             must be in the range of 0 to (BSP_NUM_UART −1).

pCfg   Pointer to a BSP_UART data structure that the application
             must initialize to effect a particular UART mode of
             operation. To learn more, see the UART Data Structures in
             the BSP API section on page 190.

Return Value

BSP_ERR_SUCCESS if no errors occur.

BSP_ERR_IN_USE if the UART driver has already been initialized and
BSP_UART_Stop() was not subsequently called; or if the UART config-
uration is requesting a DMA channel which is used by some other entity.

BSP_ERR_INVALID_PARAM is returned (with the debug version of the
BSP library) if the Idx parameter is larger than (BSP_NUM_UART −1), if
the pCfg parameter is 0 or if pCfg references a configuration specifying
an invalid DMA channel.

▶ **Note:** Using the debug version of the BSP library will cause the UART driver to
perform additional parameter validation which could result in a decrease in
driver performance.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

110

Description

This routine must be called before applications call any other UART API. After this API is called, it may not be called again until the BSP_UART_Stop API is called. This routine will acquire and initialize the hardware resources (GPIO pins, interrupts, and DMA channels) necessary to enable the UART controller in accordance with the configuration information supplied in the BSP_UART structure.

Correct Usage

If the BSP_UART configuration structure specifies that DMA is used for data transfer, be sure to call the BSP_DMA_Init API before calling this function.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

z i l o g
Embedded in Life
An∎IXYS Company

111

BSP_UART_TRANSMIT

Prototype

```
BSP_STATUS
BSP_UART_Transmit
(
 UINT8 Idx,
 HANDLE hBuf,
 BSP_SIZE Len
);
```

Parameters

Idx	Specifies which UART device to use for the transmit operation. Idx must be in the range of 0 to (BSP_NUM_UART −1).
hBuf	Application buffer containing data to be transmitted.
Len	Specifies the number of bytes of data to be transmitted.

Return Value

BSP_ERR_SUCCESS is returned if no error occurs.

BSP_ERR_BUSY is returned if the specified UART device has not yet completed a previous transmit request.

BSP_ERR_INVALID_PARAM is returned (with the debug version of the BSP library) if the Idx parameter is out of range.

BSP_ERR_FAILURE is returned (with the debug version of the BSP library) if the BSP_UART structure was initialized with a value of NULLPTR (0).

Description

This API is used to transmit Len bytes of data from the application transmit buffer referenced by hBuf to the remote UART device(s). Data is

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

112

transmitted using the standard UART protocol or LIN Protocol Mode depending on the setting of the Mode member of the `BSP_UART` structure passed to the `BSP_UART_Init` API.

If the UART driver is configured to use CPU polling for data transfer and no errors occur, this API does not return control to the caller until all `Len` bytes have been transmitted.

If the UART driver is configured to use either Interrupt Mode or DMA Mode for data transfer and this API returns `BSP_ERR_SUCCESS`, data is transmitted in the background while the foreground task continues to run. When the transmission completes (either successfully or after a transmission error occurs) the (optional) application transmit complete callback handler is called.

The function prototype of the transmit completion routine is shown in the following code snippet:

```
reentrant void
UartTxCompleteCallback
(
 UINT8 Idx,
 HANDLE hBuf,
 BSP_SIZE Len,
 UINT8 Status
);
```

The `Idx` parameter indicates which UART device is reporting its transmit completion status. The value of `Idx` is between 0 and (`BSP_NUM_UART` –1).

The `hBuf` parameter references the buffer location immediately following the last byte of data that was transmitted. The `Len` parameter is the number of bytes of data (starting at `hBuf`) that were not transmitted. If the transmission completes successfully the value of `Len` is 0 and `hBuf` points to the byte after the last byte in the original transmit buffer.

The Status parameter contains the value of the UART Status 0 register (UxSTAT0) at the time the transmit operation ended.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

113

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An ◻ IXYS Company

114

BSP_UART_RECEIVE

Prototype

```
BSP_STATUS
BSP_UART_Receive
(
 UINT8 Idx,
 HANDLE hBuf,
 BSP_SIZE Len
);
```

Parameters

Idx   Specifies the UART device from which data is to be received.
        Idx must be in the range of 0 to (BSP_NUM_UART –1).

hBuf   Application buffer into which received data is placed.

Len   Specifies the size of the application receive buffer referenced
        by hBuf.

Return Value

BSP_ERR_INVALID_PARAM is returned (with the debug version of the
BSP library) if the Idx parameter is out of range.

If no error occurs, this routine returns the number of bytes of data that
were placed into the application receive buffer referenced by hBuf.

If the fpRxCfg member of the BSP_UART structure passed to the
BSP_UART_Init API was set to NULLPTR (0), then this API will
always return 0 to indicate that there is no receive data available.

Description

This API is used to obtain data received from the peer UART device
regardless of the setting of the Mode member of the BSP_UART structure
passed to the BSP_UART_Init API; i.e., all UART protocols use the

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An■IXYS Company

115

BSP_UART_Receive API to obtained data transmitted by other UART devices.

If the UART driver is configured to transfer data using CPU polling the BSP_UART_Receive API operates synchronously; i.e., control is not returned to the caller until Len bytes of data are transmitted by the peer device or until a receive error is detected.

If the UART driver is configured to transfer data using interrupts or DMA, then the UART driver buffers received data in the ring buffer passed to the BSP_UART_Init API through the pRxBuf structure member within the pRxCfg structure. In this instance, the BSP_UART_Receive API is used to copy data from the ring buffer into the application buffer which frees up space in the ring buffer.

Correct Usage

Applications are cautioned against using CPU polling for receiving UART data, because there could be a significant (possibly unbound) delay between the time the BSP_UART_Receive API is called and the time until the remote UART device transmits Len bytes of data. During this time the CPU will be consumed with polling for UART activity preventing other application tasks from running.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

116

BSP_UART_STOP

Prototype

```
BSP_STATUS BSP_UART_Stop( UINT8 Idx );
```

Parameters

Idx   Specifies which UART device is being returned to the system for subsequent reallocation. Idx must be in the range of 0 to (BSP_NUM_UART –1).

Return Value

BSP_ERR_SUCCESS if no errors occur.

BSP_ERR_IN_USE if an attempt is made to stop a UART device that has not yet completed a previous transfer request.

BSP_ERR_INVALID_PARAM is returned (with the debug version of the BSP library) if the Idx parameter is out of range.

Description

If the UART driver is idle, this API can be used to return the UART controller to a quiescent state. If this routine is successful, all BSP resources acquired by the BSP_UART_Init API are released and the BSP_UART structure can be modified to initiate a different UART configuration by calling BSP_UART_Init.

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An∎IXYS Company

117

BSP_MP_TRANSMIT

Prototype

```
BSP_STATUS BSP_MP_Transmit
(
 UINT8 Idx,
 UINT8 DestAddr,
 HANDLE hBuf,
 BSP_SIZE Len
);
```

Parameters

Idx	Specifies which UART device to use for the transmit operation. Idx must be in the range of 0 to (BSP_NUM_UART −1).
DestAddr	The address of the remote UART device to which data is transmitted.
hBuf	Application buffer containing data to be transmitted.
Len	Specifies the number of bytes of data to be transmitted

Return Value

BSP_ERR_SUCCESS is returned if no error occurs.

BSP_ERR_BUSY is returned if the specified UART device has not yet completed a previous transmit request.

BSP_ERR_INVALID_PARAM is returned (with the debug version of the BSP library) if the Idx parameter is out of range.

BSP_ERR_FAILURE is returned (with the debug version of the BSP library) if the BSP_UART structure was initialized with a value of NULLPTR (0).

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

118

Description

This API is used to transmit data to remote the remote UART device configured with the 8-bit address DestAddress. Data is transmitted using the multiprocessor protocol (also known as 9-Bit Mode). In this mode, the parity bit (MP bit) is used to specify whether the byte being transmitted is a multiprocessor address byte (MP=1) or a data byte (MP=0).

When the BSP UART driver is configured to operate in multiprocessor mode the driver only accepts data bytes that follow the MP address byte that matches the device's own address. Received data is obtained by calling the BSP_UART_Receive API.

Correct Usage

None.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

119

BSP_DMX_TRANSMIT

Prototype

```
BSP_STATUS BSP_DMX_Transmit
(
 UINT8 Idx,
 UINT8 StartCode,
 HANDLE hBuf,
 BSP_SIZE Len
);
```

Parameters

Idx	Specifies which UART device to use for the transmit operation. Idx must be in the range of 0 to (BSP_NUM_UART −1).
StartCode	Specifies the 8-bit start code to transmit before the DMX data buffer referenced by hBuf.
hBuf	Application buffer containing data to be transmitted.
Len	Specifies the number of bytes of data to be transmitted

Return Value

BSP_ERR_SUCCESS is returned if no error occurs.

BSP_ERR_BUSY is returned if the specified UART device has not yet completed a previous transmit request.

BSP_ERR_INVALID_PARAM is returned (with the debug version of the BSP library) if the Idx parameter is out of range.

BSP_ERR_FAILURE is returned (with the debug version of the BSP library) if the BSP_UART structure was initialized with a value of NULLPTR (0).

Description

This API is used to transmit a DMX frame that includes the break, mark after break, start code a variable number of data slots (as determined by the value of the Len parameter) and the mark after break sequences.

When the BSP UART driver is configured to operate in DMX mode the driver only accepts data bytes that begin in the slot that matches its slave DMX address (and all subsequent data slots in the DMX frame) as well as the start code that precedes the slot data. The slave DMX address is determined by the value of the Addr member of the BSP_UART structure passed to the BSP_UART_Init API. For example if the master transmit a DMX frame with 10 data slots, the DMX slave whose address is 0x03 will receive the start code and the 3rd through 10th data slot values transmitted by the DMX master. Received data is obtained by calling the BSP_UART_Receive API.

Correct Usage

None.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

121

Universal Serial Bus API Reference

The BSP USB API supports USB 2.0 full-speed endpoint data transfer and basic enumeration services for USB devices. The BSP USB API does not implement any host or USB On-The-Go (OTG) features.

USB Functions in the BSP API

The routines described in this section pertain to the Z8F6482 USB controller and not to individual endpoints.

-
-
-
-

Endpoint Functions in the BSP USB API

The routines described in this section pertain to a single USB endpoint as opposed to the USB controller. Before calling any of the following USB endpoint functions, application programs must first call the `BSP_USB_Init` API.

-
-
-
-
-

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

![zilog logo] Embedded in Life
An **IXYS** Company

122

BSP_USB_INIT

Prototype

```
BSP_STATUS BSP_USB_Init( BSP_USB * pUsb );
```

Parameters

pUsb   A pointer to a BSP_USB structure that configures the USB
controller; see the USB Data Structures in the BSP API
section on page 211.

Return Value

BSP_ERR_SUCCESS if no errors occur.

BSP_ERR_IN_USE if the USB driver has already been initialized and
BSP_USB_Stop() was not subsequently called; or if the USB configura-
tion is requesting a DMA channel which is used by some other entity.

BSP_ERR_INVALID_PARAM is returned (with the debug version of the
BSP library) if the pDeviceDesc or pCfgDesc members of the
BSP_USB structure are 0.

BSP_ERR_INVALID_STATE is returned if this API is called when the
device is physically disconnected from the USB.

> **▶ Note:** Using the debug version of the BSP library will cause the USB driver to
> perform additional parameter validation which could result in a decrease in
> driver performance.

Description

This routine must be called before applications attempt to configure or
transfer data over USB endpoints. After this API is called, it may not be

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An◼IXYS Company

123

called again until the `BSP_USB_Stop` API is called. This routine will acquire and initialize the hardware resources (GPIO pins, interrupts, and DMA channels) necessary to enable the USB controller in accordance with the configuration information supplied in the `BSP_USB` structure. If successful, this routine directs the USB controller to attach to the USB which will cause the host to initiate USB enumeration.

The `BSP_USB_Init` API is used to configure the following features:

Enumeration. The `BSP_USB` structure specifies the routine used to process standard USB requests as well as USB class or vendor specific requests.

Descriptors. The `BSP_USB` structure specifies the set of device, configuration, endpoint, class and/or vendor descriptors used during USB enumeration

Data Transfer Method. The BSP USB driver can transfer endpoint data using interrupts, DMA, or CPU polling.

Application Configuration Callback. When the USB host completes enumeration or otherwise modifies the current configuration, the BSP USB driver calls the (optional) application configuration callback to enable the application to configure the appropriate set of endpoints to service that selected configuration.

To learn more, refer to the `BSP_USB` structure.

Correct Usage

If the `BSP_USB` configuration structure specifies that DMA is used for endpoint data transfer be sure to call the `BSP_DMA_Init` API before calling this function.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An ◻ IXYS Company

124

BSP_USB_POLLEVENTS

Prototype

```
void BSP_USB_PollEvents( void );
```

Parameters

None

Return Value

None.

Description

Applications that configure the BSP_USB structure to use CPU Poll Mode for endpoint data transfer should periodically call this routine to allow the BSP USB driver to service USB events. Failure to do so could result in data loss and/or the USB host controller could suspend the USB device.

Correct Usage

This routine should only be used by applications that have configured the BSP USB driver to use CPU polling for endpoint data transfer. Applications that use DMA or interrupts for USB data transfer should not call this API under any circumstance.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An**IXYS**Company

125

BSP_USB_RESUME

Prototype

```
BSP_STATUS BSP_USB_Resume( UINT8 ForceWake );
```

Parameters

ForceWake   If nonzero, specifies that the BSP USB driver should initiate resume signaling even if the host has disabled this feature.

Return Value

BSP_ERR_SUCCESS is returned if no errors occur.

BSP_ERR_FAILURE is returned if the host has disabled USB remote wake-up (i.e., device-initiated resume) and the ForceWakeup parameter is 0.

Description

If the USB bus is idle for longer than 3 ms, the USB controller will suspend the device. The BSP USB driver does not issue a callback when the USB device is suspended. Typically, the USB host will resume USB activity before the device must perform any data transfer. If the device application must wake the host prior to host-initiated resume, the application can call this API.

The USB host can use the standard SET_FEATURE request to enable remote wake-up on the device or the CLEAR_FEATURE request to disable remote wake-up. If this API is called with the ForceWake parameter set to 0, then the device will only issue resume signaling if the device's remote wake-up feature has been enabled. If the ForceWake parameter is nonzero, then the device will initiate resume signaling to the host even if the host has explicitly disabled remote wake-up.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

126

Correct Usage

In suspend state a device that is powered from USB and configured for remote wake-up must not consume more than 2.5 mA of current. If the application requires more current in suspend state an external power source should be used.

Even if the device initiates resume signaling the host might not wake-up if the particular device class (or underlying host driver) does not support the remote wake-up feature. Similarly, the host's power management (or BIOS) configuration might dictate that it should ignore device-initiated USB wake-up events.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An ◼ IXYS Company

127

BSP_USB_STOP

Prototype

```
BSP_STATUS BSP_USB_Stop( void );
```

Parameters

None

Return Value

BSP_ERR_SUCCESS is returned if no errors occur.

BSP_ERR_IN_USE is returned if any endpoint is actively performing data transfer.

Description

If all endpoints are idle this API can be used to return the USB controller to a quiescent state. If this routine is successful, all BSP resources acquired by the BSP_USB_Init API are released and the BSP_USB structure can be modified to initiate a different USB configuration by calling BSP_USB_Init.

If an IN endpoint is actively sending data to the host this API will return BSP_ERR_IN_USE. This error code is also returned for an OUT endpoint that is waiting for data received from the host. If there are in-use endpoints the application should call BSP_USB_EpAbort API prior to calling BSP_USB_Stop.

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

128

BSP_USB_EPABORT

Prototype

```
BSP_STATUS BSP_USB_EpAbort( BSP_EP Ep );
```

Parameters

Ep   Specifies the endpoint being released. Valid values of Ep
            range from BSP_EP0_IN to MAX_USB_EP. Specifying a
            value of MAX_USB_EP causes all endpoints to be aborted.

Return Value

BSP_ERR_SUCCESS is returned if no errors occur.

BSP_ERR_INVALID_PARAM is returned (with the debug version of the
BSP library) if an application specifies an endpoint number larger than
MAX_USB_EP or tries to abort an endpoint before calling BSP_USB_Init.

Description

This API is used to release one or more endpoints that are no longer
required for data transfer. Unlike the BSP_USB_EpStop API which only
releases idle endpoints, the BSP_USB_EpAbort API will release the end-
point(s) back to the system even if they are actively transferring data.
After the endpoint(s) are released they can be reconfigured and reacquired
via the BSP_USB_EpInit API.

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An◼IXYS Company

129

BSP_USB_EPINIT

Prototype

```
BSP_STATUS BSP_USB_EpInit
(
 BSP_EP Ep,
 EP_BUF_SIZE BufSize,
 FP_EP_DONE fpXferDone
);
```

Parameters

Ep	Specifies the endpoint to be initialized. Valid values of Ep range from BSP_EP0_IN to BSP_EP3_OUT.
BufSize	Specifies the size of the largest packet transferred with the host. Must be one of the following values: EP_BUF_SIZE_8, EP_BUF_SIZE_16, EP_BUF_SIZE_24, EP_BUF_SIZE_32, or EP_BUF_SIZE_64.
fpXferDone	Application provided callback function that the USB driver calls when an IN or OUT data transfer operation completes (only used in IRQ and DMA modes for IN endpoints).

Return Value

BSP_ERR_SUCCESS is returned if no errors occur.

BSP_ERR_IN_USE is returned if the target endpoint has already been initialized.

BSP_ERR_INVALID_PARAM is returned (with the debug version of the BSP library) if an application specifies an endpoint number larger than BSP_EP3_OUT or tries to initialize an endpoint before calling BSP_USB_Init, or specifies a buffer size larger than EP_BUF_SIZE_64.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

130

Description

The `BSP_USB_EpInit` API is used to configure and initialize an endpoint for data transfer. Typically, applications call this API from within their configuration callback routine for each endpoint in the selected configuration. The configuration callback routine is specified in the `fpUserConfig` member of the `BSP_USB` structure specified in the call to `BSP_USB_Init`.

The endpoint data transfer direction is determined by the value of the Ep parameter. IN endpoints are used to send data from the device to the host. OUT endpoints are used to transfer data from the host to the device.

`BSP_EP0_OUT` is reserved for processing USB requests (both standard and class/vendor specific requests). `BSP_EP0_IN` is not used by the BSP USB library but is reserved for processing class/vendor requests. Endpoint 0 IN and OUT form the default control pipe which the BSP USB driver always enables for processing host requests. The majority of these requests occur during enumeration, but the host may issue USB (device/class/vendor) requests at any time. Therefore, when an application calls this API targeting either `BSP_EP0_IN` or `BSP_EP0_OUT` the USB driver only modifies the address of the endpoint's data transfer complete callback.

The `BufSize` parameter is used to configure the maximum packet size for data sent to the host (for IN endpoints) and the expected packet size for data received from the host. The value used should correspond to the buffer size specified in the endpoint's descriptor (see the **BSP_USB** structure on page 214 to learn more). The value of the `BufSize` parameter is ignored for `BSP_EP0_IN` and `BSP_EP0_OUT`; which always use a buffer size of 64 bytes.

When the BSP USB driver is configured to use polling for endpoint data transfer, the BSP USB driver does not issue a transmit callback even if the application specifies a nonzero value for the `pXferDone` parameter (i.e., the transmission is synchronous). Consequently, the `fpXferDone` parameter should be set to 0 for IN endpoints. In Poll Mode, OUT endpoints

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

131

should specify a valid `fpXferDone` callback routine, because the
`BSP_USB_PollEvents` API will issue callbacks for data received from
the host.

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An ☐ IXYS Company

132

BSP_USB_EPSTOP

Prototype

```
BSP_STATUS BSP_USB_EpStop( BSP_EP Ep );
```

Parameters

Ep   Specifies the endpoint being released. Valid values of Ep
            range from BSP_EP0_IN to BSP_EP3_OUT.

Return Value

BSP_ERR_SUCCESS is returned if no errors occur.

BSP_ERR_BUSY is returned if the target endpoint is actively transferring
data (IN endpoint) or has posted a receive buffer (OUT endpoint) for the
host to fill.

BSP_ERR_INVALID_PARAM is returned (with the debug version of the
BSP library) if an application specifies an endpoint number larger than
BSP_EP3_OUT or tries to stop an endpoint before calling
BSP_USB_Init.

Description

This API is used to release an idle endpoint that is no longer required for
data transfer. However, an error is returned if the endpoint is actively
transferring data. After the endpoint is released, it can be reconfigured
and reacquired via the BSP_USB_EpInit API.

Endpoints BSP_EP0_OUT and BSP_EP0_IN are used for the default con-
trol pipe. As such the BSP USB driver does not actually disable those
endpoints when targeted by this API and the BSP driver will continue to
process USB requests on EP 0. When this API is called for either
BSP_EP0_OUT or BSP_EP0_IN and the target endpoint is idle, the BSP

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

133

driver only stops calling the `fpXferDone` callback passed to the `BSP_USB_EpInit` API.

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An ∎IXYS Company

134

BSP_USB_EPTRANSMIT

Prototype

```
BSP_STATUS BSP_USB_EpTransmit
(
 BSP_EP Ep,
 HANDLE hBuf,
 BSP_SIZE Len
);
```

Parameters

Ep	Specifies which IN endpoint to use for sending data to the host. Must be one of BSP_EP0_IN to BSP_EP3_IN.
hBuf	References a buffer containing the data to be sent to the host.
Len	Specifies the length of the data buffer referenced by hBuf.

Return Value

BSP_ERR_SUCCESS is returned if no errors occur.

BSP_ERR_BUSY is returned if the target endpoint is actively transferring data (i.e., the target endpoint is currently sending data to the host).

BSP_ERR_INVALID_PARAM is returned (with the debug version of the BSP library) if an application specifies an endpoint number larger than BSP_EP3_IN or tries to transmit data on an uninitialized endpoint.

BSP_ERR_INVALID_STATE is returned if an attempt is made to transmit on a suspended endpoint.

Description

This API is used to post a transmit buffer to the BSP USB driver. The USB driver copies the transmit data from the application buffer (referenced by hBuf) into endpoint buffer RAM within the USB controller.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An ◾IXYS Company

135

Because the BSP USB driver only provides device support, the data is not actually transmitted to the host until the host issues an IN token. After the host issues an IN token to the endpoint targeted by the Ep parameter, the USB controller will transmit the application data to the host. After all the data has been transferred, the BSP USB driver will call the fpXferDone callback routine specified in the call to BSP_USB_EpInit (only if USB is configured to use DMA or interrupts for endpoint data transfer and the fpXferDone parameter is non-zero).

When the USB controller is configured to use DMA or interrupts for end-point data transfer, a return code of BSP_ERR_SUCCESS does not mean that the data has been transmitted to the host. It only means that the BSP USB driver has accepted the application data for transmission. Between the time this API returns BSP_ERR_SUCCESS and until the data is actually transmitted (and the optional fpXferDone callback is called), subsequent calls to this API will return a status of BSP_ERR_BUSY.

When the USB controller is configured to use polling for endpoint data transfer, this routine does not return until the data is successfully transmitted to the host or some other error occurs. In this instance, a return code of BSP_ERR_SUCCESS does indicate that the data has successfully been sent to the host. Since data is transmitted synchronously in poll mode, the BSP USB driver does not call the fpXferDone callback when the transfer completes.

The application data buffer can be of any size; it is not constrained by the size of the BufSize parameter passed to the BSP_USB_EpInit API. If the buffer passed to this routine is larger than the BufSize parameter passed to BSP_USB_EpInit, then this routine will internally subdivide the application buffer into packets of up to BufSize bytes. The last packet will contain between 1 and (BufSize -1) bytes if the Len parameter is not an integer multiple of BufSize. The (nonzero) fpXferDone callback function will only be called after the entire application buffer has been transmitted regardless of the size of hBuf.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

136

Correct Usage

When using interrupt and/or DMA for endpoint data transfer applications should not modify the contents of hBuf until the transmit operation completes. Consequently, hBuf should only be modified after the endpoint's fpXferDone callback function is called.

When transmitting a ZLP in Interrupt Mode or DMA Mode, the EP_SEND_ZLP bit in the Status member of the EP_CB_INFO structure passed to the fpXferDone callback will be set to 1 if the BSP USB driver did not attempt to send a ZLP to the host. This occurs if the host begins a new control transfer before the application is able to request transmission of a ZLP in the previous transfer. When polling is used for data transfers, this routine does not return until the ZLP is transmitted (bulk and control endpoints) or is aborted (only applicable to control endpoints).

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An∎IXYS Company

137

BSP_USB_EPRECEIVE

Prototype

```
BSP_STATUS BSP_USB_EpReceive
(
 BSP_EP Ep,
 HANDLE hBuf,
 BSP_SIZE Len
);
```

Parameters

Ep	Specifies which OUT endpoint to use for receiving data from the host. Must be one of BSP_EP0_OUT to BSP_EP3_OUT.
hBuf	References an application buffer into which data from the host will be placed.
Len	Specifies the length of the data buffer referenced by hBuf.

Return Value

BSP_ERR_SUCCESS is returned if no errors occur.

BSP_ERR_BUSY is returned if the target endpoint is actively transferring data (i.e., a receive buffer has already been posted for the target endpoint).

BSP_ERR_INVALID_PARAM is returned (with the debug version of the BSP library) if an application specifies an endpoint number smaller than BSP_EP0_OUT, larger than BSP_EP3_OUT, tries to transmit data on an uninitialized endpoint or specifies a Len parameter of 0.

BSP_ERR_INVALID_STATE is returned if an attempt is made to receive data on a suspended endpoint.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

z i l o g
Embedded in Life
An ◼ IXYS Company

138

Description

This API is used to post a receive buffer (referenced by hBuf) to the BSP USB driver. After the host issues an OUT transaction to the endpoint targeted by the Ep parameter, the data from the host is copied into the receive buffer and the (optional) fpEpXfer done callback routine (a parameter to the BSP_USB_EpInit API) is called.

A return code of BSP_ERR_SUCCESS does not mean that there is data available in hBuf; it only indicates that the BSP USB driver has accepted the receive buffer and that the endpoint has been enabled for data reception. Between the time this API returns BSP_ERR_SUCCESS and until the data is actually received (and the optional fpXferDone callback is called), subsequent calls to this API will return a status of BSP_ERR_BUSY.

The application receive buffer can be of any size; it is not constrained by the size of the BufSize parameter passed to the BSP_USB_EpInit API. The Len parameter determines the maximum amount of data that can be received from the host in a single transfer (that can span several USB transactions) while the BufSize parameter passed to the BSP_USB_EpInit API determines the maximum packet size expected within a single USB transaction. As long as OUT transactions contain a full sized data packet (equal to BufSize bytes), the host is sending data within the same transfer. The host transmits a packet of less than Buf-Size bytes to mark the end of the current transfer.

Regardless of whether the host has finished the current OUT transfer the fpXferDone API is called when the receive buffer is full. The fpXfer-Done routine is also called when the host finishes the OUT transfer (as determined by the reception of a packet of length < BufSize).

Between the time the fpXferDone API is called and the time when the application calls this routine to post another receive buffer, the USB controller can buffer up to BufSize bytes of data for the endpoint. Subsequent OUT transactions will be NAKed by the USB controller until the application calls this API.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

z i l o g
Embedded in Life
An ■ IXYS Company

139

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

140

Appendix A. Data Structures

This appendix describes the data structures (C structures, unions, and enumerations) that apply to the BSP API.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

z i l o g
Embedded in Life
An∎IXYS Company

141

AES Structures and Unions in the BSP API

This section presents the following data structures, which are used by the BSP AES driver.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

142

BSP_AES

Definition

```
typedef struct BSP_AES_s
{UINT8 Mode;
 UINT8 Key[ AES128_BLOCK_SIZE ];
 UINT8 Decrypt;
 AES_CFG * pAesCfg;
}BSP_AES;
```

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An◼IXYS Company

143

Members

Mode   The encryption mode; must be one of AES_MODE_ECB, AES_MODE_OFB, AES_MODE_CBC, or AES_MODE_KEYD.

Key   The 16-byte encryption or decryption key. The ECB encryption mode uses separate keys for encryption and decryption; the decryption key must be derived from the encryption key using the AES_MODE_KEYD operation. The same is true of CBC Encryption Mode, with the further requirement that the encryption mode must be set to AES_MODE_ECB when decrypting messages encrypted with AES_MODE_CBC. In OFB Encryption Mode, the same key is used for both encryption and decryption.

Decrypt   Set to AES_ENCRYPT to encrypt a message or derive a decryption key. Set to AES_DECRYPT to decrypt a message (except when using OFB encryption mode).

pAesCfg   A pointer to an AES_CFG union that configures the data transfer mode (Poll, Irq, or Dma). In defining the BSP_AES structure, define this pointer as being to an object which is one of the types in the AES_CFG union, but cast it to (AES_CFG *), as shown in the following code fragment:

```
BSP_AES AesConfig =
{
 AES_MODE_ECB,
 {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15},
 AES_ENCRYPT,
 (AES_CFG *) &IrqCfg
};
```

Correct Usage

n/a

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

144

AES_CFG

Definition

```
typedef union AES_CFG_u
{                   AES_POLL_CFG Poll;
AES_IRQ_CFG Irq;
AES_DMA_CFG Dma;
} AES_CFG;
```

Members

Poll	A struct of type AES_POLL_CFG, if you want to use polling for data transfers to and from the AES accelerator.
Irq	A struct of type AES_IRQ_CFG, if you want to use interrupt-driven data transfers.
Dma	A struct of type AES_DMA_CFG, if you want to use DMA for data transfers.

Correct Usage

You must define a struct of one of these three types, then use a pointer to that struct, cast to (AES_CFG *), in declaring a BSP_AES structure. Refer to the structure defined in the to learn more.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

z i l o g
Embedded in Life
An◻IXYS Company

145

AES_POLL_CFG

Definition

```
typedef struct AES_POLL_CFG__s
{              FP_AES_SETUP fpSetup;
} AES_POLL_CFG;
```

Members

fpSetup   A function pointer used in initializing the AES; must be set
to AES_PollSetup.

Correct Usage

If fpSetup is not defined properly, runtime errors will result.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

146

AES_IRQ_CFG

Definition

```
typedef struct AES_IRQ_CFG_s
{                   FP_AES_SETUP fpSetup;
FP_AES_DONE fpAesDone;
} AES_IRQ_CFG;
```

Members

fpSetup   A function pointer used in initializing the AES; must be set
            to AES_IrqSetup.

fpAesDone   A function pointer to a callback function, which will be
            called when the AES transform is complete. This operation
            is necessary because AES transfers are nonblocking in this
            data transfer mode.

Correct Usage

If fpSetup and fpAesDone are not defined properly, runtime errors will
result.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An∎IXYS Company

147

AES_DMA_CFG

Definition

```
typedef struct AES_DMA_CFG_s
{                    FP_AES_SETUP fpSetup;
FP_AES_DONE fpAesDone;
UINT16 InDMABase;
UINT16 OutDMABase;
} AES_DMA_CFG;
```

Members

fpSetup	A function pointer used in initializing the AES; must be set to AES_DmaSetup.
fpAesDone	A function pointer to a callback function, which will be called when the AES transform is complete. This operation is necessary because AES transfers are nonblocking in this data transfer mode.
InDMABase	The DMA channel to be used for transferring data to the AES accelerator. Must be one of: BSP_DMA0, BSP_DMA1, BSP_DMA2, or BSP_DMA3.
OutDMABase	The DMA channel to be used for extracting data from the AES accelerator. Must be one of: BSP_DMA0, BSP_DMA1, BSP_DMA2, or BSP_DMA3, and must be different from InDMABase.

Correct Usage

If fpSetup and fpAesDone are not defined properly; or if InDMABase and OutDMABase are not defined properly, or are not different from each other; or if the DMA channel corresponding to either InDMABase or Out-DMABase is already in use by another peripheral, then runtime errors will result.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

148

CLKS Data Structure in the BSP API

This section presents the data structure ("C" structure, union and enumeration) used by the BSP clock system driver.

-

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An ☐ IXYS Company

149

BSP_CLKS

Definition

```
typedef struct          BSP_CLKS_s
{
   UINT8                SCK_Ctrl;
   UINT8                PCK_Ctrl;
   UINT8                HFXO_Ctrl;

   struct
   {
      UINT8             Ctrl;
      UINT8             RdivOdiv;
      UINT8             Ndiv;
   }                    PLL;

   struct
   {
      UINT8             Ctrl;
      UINT16            FLL_Div;
      UINT8             CWH;
      UINT8             CWL;
   }                    DCO;
} BSP_CLKS;
```

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

150

Members

SCK_Ctrl   Specifies the system clock control value to be written to the CLKCTL0 Special Function Register. If the value of the SCK_Ctrl structure member includes the CLKS_CSTAT flag, access to the system clock registers will remain unlocked after the BSP_CLKS_Config routine returns to the caller. If the CLKS_CSTAT flag is not specified, then access to the clock system registers will be locked. To learn how to initialize the SCK_Ctrl structure member, see the description of the CLKCTL0 Register in the Z8F6482 Series Product Specification.

PCK_Ctrl   Specifies the peripheral clock (PCLK) control value to be written to the CLKCTL1 Special Function Register. Be sure to specify the CLKS_IPOEN if the internal precision oscillator is selected as the PCLK source. Similarly, the CLKS_LFXOEN flag should be used if the low-frequency crystal oscillator is selected as the PCLK source. To learn how to initialize the PCK_Ctrl structure member, see the description of the CLKCTL1 Register in the Z8F6482 Series Product Specification.

HFXO_Ctrl   Specifies the high frequency crystal oscillator (HFXO) control value to be written to the CLKCTL2 Special Function Register. If the HFXO is used as either the PLL or system clock source, the value of the HFXO_Ctrl structure member should include the CLKS_HFXOEN flag. To learn how to initialize the HFXO_Ctrl structure member, see the description of the CLKCTL2 Register in the Z8F6482 Series Product Specification.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An◻IXYS Company

151

PLL   The PLL member of the BSP_CLKS structure is also a structure that contains the parameters used to initialize the phase locked loop clock (PLLClk). The PLL.Ctrl, PLL.RdivOdiv, and PLL.Ndiv structure members are written to the CLKCTLA, CLKCTLB, and CLKCTLC special function registers. Typically, the PLL is configured to produce a 48MHz output clock used by the integrated USB controller; the PLL can also be used as the system clock. If the PLL is not being used, the PLL.Ctrl structure member should be set to 0. To learn how to configure the PLL, see the descriptions of the CLKCTLA, CLKCTLB, and CLKCTLC registers in the Z8F6482 Series Product Specification.

DCO   The DCO member of the BSP_CLKS structure is also a structure that contains the parameters used to initialize the digitally-controlled oscillator (DCO). The DCO.Ctrl, DCO.FLL_Div, DCO.CWH, and DCO.CWL structure members are written to the CIKCTL3 Register through the CLKCTL7 special function registers. If used, the DCO can be configured to free run off the high and low control words (CWH and CWL) or can be locked to a multiple of PCLK, as determined by the value of the frequency-locked loop divider (FLL_Div). If the DCO is configured to free run, then the CLKS_FLLEN flag is not required to be specified in the value of the DCO.Ctrl structure member. If the DCO is not used, set the value of the DCO.Ctrl structure member to 0. To learn how to configure the DCO, see the description of the CLKCTL3 Register in the Z8F6482 Series Product Specification.

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

152

DAC Structures and Unions in the BSP API

This section presents the following data structures, which are used by the BSP DAC driver.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

153

BSP_DAC

Definition

```
typedef struct          BSP_DAC_s
{
 UINT8                  dacCtrlReg;
 UINT8                  inpDataTwoBytes;
 UINT16                 initVal;
 DAC_BUF_OUTPUT_CFG     * pBufOpCfg;
} BSP_DAC;
```

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

154

Members

dacCtrlReg The DAC hardware is configured by copying this value to the SFR DACCTL. Individual fields within this byte control the power level at which the DAC will operate; the DAC voltage reference; whether input data are treated as signed or unsigned; whether direct operation or Event System triggering is used to drive conversions; and the left or right justification of the data in the 2 8-bit DAC data registers. The justification, in combination with inpDataTwoBytes, in essence tells the DAC what type of data you will be providing for conversion. You should select right justification if your input data are already constrained to a 12-bit range, matching the output range of the DAC. Use left justification if your input data will be 8 or 16 bits and you want the DAC hardware to scale those values into its output range.

inpDataTwoBytes If TRUE, the data presented to the DAC must be two bytes per value, either 12 or 16 bits depending on the justification. If FALSE, the input data must be 8 bit values.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An IXYS Company

155

initVal	An initial value that will be written to the DAC data registers before the DAC is enabled. When the DAC is enabled by calling BSP_DAC_Init(), conversion and output of this value will begin immediately, and this value will remain in the DAC output until you request output of either a single value or a buffer. This value is a literal value that will be written to the DAC data registers. When the DAC is enabled, the value initVal will be interpreted and converted in the way determined by your setup of the DAC hardware via dacCtrlReg. See the code comments in BSP_DAC.h for detailed examples.
pBufOpCfg	A pointer to a DAC_BUF_OUTPUT_CFG union that configures the data transfer mode (Irq or Dma). This pointer applies only to the case, in which a buffer of data values is to be output, driven by the Event System. For direct operation of the DAC, set this pointer to NULLPTR. For Event System operation, define it as a pointer to an object which is one of the types in the DAC_BUF_OUTPUT_CFG union, but cast it to DAC_BUF_OUTPUT_CFG *. See the DAC sample programs for examples.

Correct Usage

Application code is responsible for ensuring that the type (signed or unsigned; 8, 12, or 16-bit data range) of all data which the DAC is asked to convert does in fact match the hardware setup of the DAC that has been specified in this structure. If not, the output levels of the DAC will not be as expected.

Some of the possible DAC voltage reference selections set by dacCtrlReg require external hardware, or impose requirements on the power supply voltage, or interact with settings of the Analog to Digital Converter

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

156

(ADC). It is your responsibility to ensure that the software configuration defined by dacCtrlReg is consistent with your hardware.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An◻IXYS Company

157

DAC_ BUF_OUTPUT_CFG

Definition

```
typedef union DAC_ BUF_OUTPUT_CFG_u
{ DAC_BUF_OUTPUT_IRQ Irq;
DAC_BUF_OUTPUT_DMA Dma;
} DAC_ BUF_OUTPUT_CFG;
```

Members

Irq   A struct of type DAC_BUF_OUTPUT_IRQ, if you want to
             use interrupt-driven data transfers to output a buffer to the
             DAC.

Dma   A struct of type DAC_ BUF_OUTPUT_DMA, if you want to
             use DMA for data transfers to output a buffer to the DAC.

Correct Usage

You must define a struct of one of these 2 types, then use a pointer to that
struct, cast to (DAC_ BUF_OUTPUT_CFG *), in declaring a BSP_DAC
structure if your intended operation is to output a buffer to the DAC. See
the DAC sample programs for examples.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An ∎IXYS Company

158

DAC_BUF_OUTPUT_IRQ

Definition

```
typedef struct DAC_ BUF_OUTPUT_IRQ_s
{                    FP_DAC_SETUP     fpSetup;
FP_DAC_DONE            fpDacDone;
UINT8                              stopInstantly;
} DAC_ BUF_OUTPUT_IRQ;
```

Members

fpSetup	A function pointer used in initializing the DAC; must be set to DAC_IrqOutputBufSetup.
fpDacDone	A function pointer to a callback function, which will be called when all data transfers to the DAC are complete. This operation is necessary because transfers of data buffers to the DAC are nonblocking, being under the control of the Event System.
stopInstantly	If FALSE, the completion callback function fpDacDone will not be called until the final value in the input buffer has been driven on the DAC output pin for the normal convert-and-hold time. You may prefer this option if your use of the DAC will end, at least for the near term, after the current buffer has been output (possibly repeatedly). The goal in this case is to make the DAC output hardware treat the final value in the input buffer consistently with all the other buffer elements.

If stopInstantly is TRUE, fpDacDone will be called as soon as the data transfer to the DAC data registers is complete.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

159

> ▶ **Note:** At this point, the final value transferred to the DAC has not yet begun to be driven on the DAC output, because on every Event System signal, the DAC drives the previous contents of the data registers on its output, then requests a new data value from the interrupt service or DMA.

The `stopInstantly` = TRUE option is provided for the use of application code when the goal is to transition as fast as possible from output of one buffer to another, for example if streaming data continuously to the DAC from external hardware. Making a smooth transition from one buffer output to another and managing the DAC output timing for such applications is the responsibility of the application code.

Correct Usage

If `fpSetup` and `fpDacDone` are not defined properly, runtime errors will result.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

160

DAC_BUF_OUTPUT_DMA

Definition

```
typedef struct DAC_ BUF_OUTPUT_DMA_s
{                   FP_DAC_SETUP    fpSetup;
FP_DAC_DONE            fpDacDone;
UINT8                           stopInstantly;
UINT16                          dmaBase;
} DAC_ BUF_OUTPUT_DMA;
```

Members

fpSetup	A function pointer used in initializing the DAC; must be set to DAC_DmaOutputBufSetup.
fpDacDone	A function pointer to a callback function, which will be called when all data transfers to the DAC are complete. This operation is necessary because transfers of data buffers to the DAC are nonblocking, being under the control of the Event System.
stopInstantly	If FALSE, the completion callback function fpDacDone will not be called until the final value in the input buffer has been driven on the DAC output pin for the normal convert-and-hold time. You may prefer this option if your use of the DAC will end, at least for the near term, after the current buffer has been output (possibly repeatedly). The goal in this case is to make the DAC output hardware treat the final value in the input buffer consistently with all the other buffer elements.

If stopInstantly is TRUE, fpDacDone will be called as soon as the data transfer to the DAC data registers is complete.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An ◼ IXYS Company

161

> **Note:** At this point, the final value transferred to the DAC has not yet begun to be driven on the DAC output, because on every Event System signal, the DAC drives the previous contents of the data registers on its output, then requests a new data value from the interrupt service or DMA.

The `stopInstantly` = TRUE option is provided for the use of application code when the goal is to transition as fast as possible from output of one buffer to another, for example if streaming data continuously to the DAC from external hardware. Making a smooth transition from one buffer output to another and managing the DAC output timing for such applications is the responsibility of the application code.

dmaBase   The DMA channel to be used for transferring data to the DAC; it must be one of: `BSP_DMA0`, `BSP_DMA1`, `BSP_DMA2`, or `BSP_DMA3`.

Correct Usage

If `fpSetup`, `fpDACDone`, and `dmaBase` are not defined properly; or if the DMA channel corresponding to `dmaBase` is already in use by another peripheral, then runtime errors will result.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

162

DMA Data Structures in the BSP API

This section presents the following data structure, which is used by the BSP DMA driver.

- <u>DMA_DESC</u> – see page 163

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

z i l o g
Embedded in Life
An ■ IXYS Company

163

DMA_DESC

Definition

```
typedef struct DMA_DESC_s
{
 UINT16 Src;
 UINT16 Dst;
 UINT16 Cnt;
 UINT8 Ctl0;
 UINT8 Ctl1;
} DMA_DESC;
```

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

164

Members

`Src`	Address of the (RAM) memory buffer containing the data to be transferred or the Special Function Register (SFR) address of a peripheral device from which data is to be extracted.
`Dst`	Address of the (RAM) memory buffer into which data is transferred or the Special Function Register (SFR) address of a peripheral device to which data is to be transferred.
`Cnt`	Specifies the number of bytes of data to be transferred.
`Ctl0`	Specifies the value to be written to the DMA channel's DMAxCTL0 Special Function Register. To learn more about the meaning of this value, refer to the [Z8F6482 Series Product Specification](#).
`Ctl1`	Specifies the value to be written to the DMA channel's DMAxCTL1 Special Function Register. To learn more about the meaning of this value, refer to the [Z8F6482 Series Product Specification](#).

Correct Usage

Linked list DMA transfers require the application to provide an array of descriptors that configure the DMA channel's special function registers to perform a transfer composed of one or more stages. Each descriptor in the array must be of type DMA_DESC. Applications are responsible for ensuring that the array of descriptors is located at an address in RAM that is evenly divisible by 8 (i.e., the descriptor array must be 8-byte aligned).

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

165

GPIO Data Structures in the BSP API

This section presents the following data structure, which is used by the GPIO module.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

166

BSP_GPIO_CFG

Definition

```
typedef struct BSP_GPIO_CFG_s
{
 IOReg8 Port;
 UINT8 AF_Mask;
 UINT8 AFS1_Mask;
 UINT8 AFS2_Mask;
} BSP_GPIO_CFG;
```

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An∎IXYS Company

167

Members

Port   Specifies the GPIO port whose pins are being configured
            for alternate function mode. The value of the Port structure
            member must be between BSP_GPIO_PORT_A to
            BSP_GPIO_PORT_J (excluding BSP_GPIO_PORT_I
            which is not defined for the Z8F6482 Series).

AF_Mask   The alternate function mask identifies the set of pins within
            the port being configured for the same alternate
            subfunction mode. This value can be expressed as an 8-bit
            number (e.g., 0x30) or constructed using macros from the
            BSP.h header file (e.g., BIT5 | BIT4) or the
            Z8F6482_GPIO_SFR.h header fie (e.g., GPIOA_TXD0 |
            GPIOA_RXD0).

AFS1_Mask   Specifies whether the AFS1 subfunction selection bit(s)
            should be set or cleared for the specified set of GPIO
            pin(s). If the value of the AFS1_Mask structure member is
            0, then the bit(s) corresponding to the AF_Mask structure
            member in the GPIO AFS1 subregister are cleared to 0. If
            the value of the AFS1_Mask structure member is nonzero,
            then the bit(s) corresponding to the AF_Mask structure
            member in the GPIO AFS1 subregister are set to 1.

AFS2_Mask   Specifies whether the AFS2 subfunction selection bit(s)
            should be set or cleared for the specified set of GPIO
            pin(s). If the value of the AFS2_Mask structure member is
            0, then the bit(s) corresponding to the AF_Mask structure
            member in the GPIO AFS2 subregister are cleared to 0. If
            the value of the AFS2_Mask structure member is nonzero,
            then the bit(s) corresponding to the AF_Mask structure
            member in the GPIO AFS2 subregister are set to 1.

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

168

I²C Structures and Unions in the BSP API

This section presents the following data structures, which are used by the BSP I²C driver.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An ◼ IXYS Company

169

BSP_I2C

Definition

```
typedef struct BSP_I2C_s
{
 FP_I2C_DONE fpXferDone;
 I2C_CFG* pCfg;
} BSP_I2C;
```

Members

fpXferDone   The callback function called at the end of each I^{2}C transfer.

pCfg   Pointer to an I2C_CFG containing configuration information.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

170

I2C_CFG

Definition

```
typedef union I2C_CFG_u
{
 I2C_COMMON_CFG common;
 I2C_MASTER_POLLING master_poll;
 I2C_MASTER_IRQ master_irq;
 I2C_MASTER_DMA master_dma;
 I2C_SLAVE_POLLING slave_poll;
 I2C_SLAVE_IRQ slave_irq;
 I2C_SLAVE_DMA slave_dma;
} I2C_CFG
```

Members

See the descriptions of the individual structures in this union to learn more.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An∎IXYS Company

171

I2C_COMMON_CFG

Definition

```
typedef struct I2C_COMMON_CFG_s
{
 FP_I2C_SETUP fpSetup;
 UINT16 Brg;
 rom BSP_GPIO_CFG *pI2cCfg;
}I2C_COMMON_CFG;
```

Members

fpSetup	A function pointer used in initializing the I^2C. Each of the other members of the I2C_CFG union has a specific value this must be set to.
Brg	The value to place in the I2CBR Register to set the baudrate. Use the I2C_Brg macro to compute this.
pI2CCfg	A pointer to an array in ROM to initialize the I^2C ports. The following arrays are provided in the BSP library: I2C_Pins_A6_A7: A6 is the SCL and A7 is SDA. I2C_Pins_A6_C5: A6 is the SCL and C5 is SDA. I2C_Pins_C4_A7: C4 is the SCL and A7 is SDA. I2C_Pins_C4_C5: 4 is the SCL and C5 is SDA.

Description

I2C_COMMON_CFG is a common front end to all the other members of union I2C_CFG.

I2C_MASTER_POLLING

Definition

```
typedef struct I2C_MASTER_POLLING_s
{
 FP_I2C_SETUP fpSetup;
 UINT16 Brg;
 rom BSP_GPIO_CFG *pI2cCfg;
} I2C_MASTER_POLLING;
```

See the I2C_COMMON_CFG structure on page 171 for the definition of
these members. `fpSetup` must be set to
`I2C_Setup_Master_Polling`.

Correct Usage

Use this structure to configure I^{2}C to act as a master in Poll Mode.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An ■IXYS Company

173

I2C_MASTER_IRQ

Definition

```
typedef struct I2C_MASTER_IRQ_s
{
 FP_I2C_SETUP fpSetup;
 UINT16 Brg;
 rom BSP_GPIO_CFG *pI2cCfg;
} I2C_MASTER_IRQ;
```

See the I2C_COMMON_CFG structure on page 171 for the definition of these members. fpSetup must be set to I2C_Setup_Master_Irq.

Correct Usage

Use this structure to configure I²C to act as a master in Interrupt Mode.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

174

I2C_MASTER_DMA

Definition

```
typedef struct I2C_MASTER_DMA_s
{
 FP_I2C_SETUP fpSetup;
 UINT16 Brg;
 rom BSP_GPIO_CFG *pI2cCfg;
 UINT16 DmaBase;
} I2C_MASTER_DMA;
```

Members

See the I2C_COMMON_CFG structure on page 171 for the definition of the first three members. `fpSetup` must be set to `I2C_Setup_Master_Dma`.

`DmaBase`   The special function register to use as the base for DMA operations.

Correct Usage

Use this structure to configure I^{2}C to act as a master in Interrupt Mode with transfers other than the address part of a message handled by DMA.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An **■**IXYS Company

175

I2C_SLAVE_POLLING

Definition

```
typedef struct I2C_SLAVE_POLLING_s
{
 FP_I2C_SETUP fpSetup;
 UINT16 Brg;
 rom BSP_GPIO_CFG *pI2cCfg;
 UINT16 address;
 UINT8 tenBitAddress;
} I2C_SLAVE_POLLING;
```

Members

See the I2C_COMMON_CFG structure on page 171 for the definition of the first three members. fpSetup must be set to I2C_Setup_Slave_Polling.

address	The slave address of the device.
tenBitAddress	Nonzero if the device will have a 10-bit I^2C address.

Correct Usage

Use this structure to configure I^2C to act as a slave in Poll Mode.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

176

I2C_SLAVE_IRQ

Definition

```
typedef struct I2C_SLAVE_IRQ_s
{
 FP_I2C_SETUP fpSetup;
 UINT16 Brg;
 rom BSP_GPIO_CFG *pI2cCfg;
 UINT16 address;
 UINT8 tenBitAddress;
} I2C_SLAVE_IRQ;
```

Members

See the I2C_COMMON_CFG structure on page 171 for the definition of the first three members. `fpSetup` must be set to `I2C_Setup_Slave_Irq`.

address	The slave address of the device.
tenBitAddress	Nonzero if the device will have a ten bit I^2C address.

Correct Usage

Use this structure to configure I^2C to act as a slave in Interrupt Mode.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An ∎IXYS Company

177

I2C_SLAVE_DMA

Definition

```
typedef struct I2C_SLAVE_DMA_s
{
 FP_I2C_SETUP fpSetup;
 UINT16 Brg;
 rom BSP_GPIO_CFG *pI2cCfg;
 UINT16 address;
 UINT8 tenBitAddress;
 UINT16 DmaBase;
} I2C_SLAVE_DMA;
```

Members

See the I2C_COMMON_CFG structure on page 171 for the definition of the first three members. fpSetup must be set to I2C_Setup_Master_Dma.

address	The slave address of the device.
tenBitAddress	Nonzero if the device will have a ten bit I^{2}C address.
DmaBase	The special function register to use as the base for DMA operations.

Correct Usage

Use this structure to configure I^{2}C to act as a slave in Interrupt Mode with data transfers handled by DMA.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

178

I2C_STATUS

Definition

```
typedef unsigned char enum I2C_Status
{
 I2C_SUCCESS,
 I2C_ERR_NCKI,
 I2C_ERR_NO_SUCH_SLAVE,
 I2C_ERR_SHORT_MESSAGE,
 I2C_ERR_OVERFLOW,
 I2C_ERR_ARBLST,
} I2C_STATUS;
```

Correct Usage

An I2C_Status is passed to the `fpXferDone` callback function to indicate the ultimate result of the requested operation.

Members

`I2C_SUCCESS`	The transfer was a success.
`I2C_ERR_NCKI`	The transmission was aborted because of a nonacknowledgment by the recipient on other than the last character to be transmitted. This member normally means that the recipient's buffer was full, but could result from a hardware failure.
`I2C_ERR_NO_SUCH_SLAVE`	The message was not sent because no slave responded to the address given.
`I2C_ERR_SHORT_MESSAGE`	Less than the full length of the message was sent or received for reasons other than `I2C_ERR_NCKI`. Depending on the protocol implemented between master and slave, this could simply because the buffer passed was large enough to accommodate any message, or it could indicate some sort of error.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

179

| I2C_ERR_ OVERFLOW | More than the specified number of bytes were sent or received. (This issue should never occur.) |
| I2C_ERR_ARBLST | The message was aborted because of a loss of arbitration on the I^2C bus. |

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

180

I2C_STATE

Definition

```
typedef unsigned char enum I2C_State
{
 I2CIdle,
 I2CReceive,
 I2CTransmit,
} I2C_STATE;
```

Correct Usage

An I2C_State is passed to the `fpXferDone` callback function to indicate whether the message completed was a transfer or a receive.

Members

I2CIdle	This value is never actually passed to the callback function. Within the I^{2}C code, it indicates that the system is idle.
I2CReceive	The message was a receive.
I2CTransmit	The message was a transmit.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An ◼ IXYS Company

181

I²C Callback Functions in the BSP API

This section presents the following I²C callback function, which is used by the BSP API.

- FP_I2C_DONE (fpXferDone) – see page 182

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

182

FP_I2C_DONE (FPXFERDONE)

Definition

```
typedef void (* FP_I2C_DONE)(I2C_STATE state,
 I2C_STATUS status,
 HANDLE message,
 BSP_SIZE len);

FP_I2C_DONE fpXferDone; // (Member of BSP_I2C structure)
```

Parameters

state	An I2C_STATE indicating whether the operation was a transmit or a receive.
status	An I2C_STATUS indicating whether the operation was a success.
message	The buffer originally passed to one of the I^2C transfer functions.
len	The length in bytes of the message transmitted or received.

> **Note:** The fpXferDone callback function is called at the end of each I^2C transfer to indicate the ultimate result of the request. Unless you are using polling for I^2C, this function is called out of an interrupt routine, and should normally save the data and return.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

z i l o g
Embedded in Life
An ◼ IXYS Company

183

SPI Data Structures in the BSP API

This section presents the following data structure, which is used by the BSP SPI driver.

- <u>BSP_SPI</u> – see page 184

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

184

BSP_SPI

Definition

```
typedef struct BSP_SPI_s
{
 UINT8 Ctrl;
 UINT8 Mode;
 UINT16 Brg;

 rom BSP_GPIO_CFG * pGpioCfg;

 FP_SPI_SETUP fpSetup;
 FP_SPI_DONE fpXferDone;

 UINT16 TxDmaBase;
 UINT16 RxDmaBase;

 UINT8 NullTxChar;

 UINT8 I2S_WordSize;

 BOOL Use_nSS;
} BSP_SPI;
```

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An ◼ IXYS Company

185

Members

Ctrl   During initialization the value of the Ctrl structure member
              is written to the ESPICTL Register*. The SPI driver
              ignores the value of the DIRQS specified in the Ctrl byte.

Mode   During initialization the value of the Mode structure
              member is written to the ESPIMODE Register*.

Brg   During initialization the value of the Brg structure member
              is written to the ESPIBRH and ESPIBRL registers*.

pGpioCfg   References a BSP_GPIO_CFG structure that specifies the
              GPIO port pin configuration applicable to the target SPI
              controller. The structure referenced by the pGpioCfg
              pointer should be located in Flash (the compiler's ROM
              memory space). For example, the default GPIO port pin
              configuration for SPI0 on the Z8F6482 controller is shown
              in the following code snippet:

Note: *For a description of these registers, refer to the Z8F6482 Series Product Specification.

```
rom BSP_GPIO_CFG Spi0GpioCfg[ 2 ] =
{
 {
 BSP_GPIO_PORT_C,
 (GPIOC_SCK0 | GPIOC_MOSI0 |GPIOC_MISO0 | GPIOC_nSS0),
 0, 0
 },
 {0,0,0,0}
};
```

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An IXYS Company

186

To learn more, refer to the BSP_GPIO API reference.

fpSetup   A function pointer that references the hardware
               specific initialization routine that determines the data
               transfer method to be used (Poll Mode, Interrupt
               Mode or DMA Mode) when the BSP_SPI_Xfer
               routine is called. The BSP library includes the
               following default setup routines that can be used to
               initialize the fpSetup member of the BSP_SPI
               structure:

```
SPI_PollSetup    // For Poll mode on
                 // SPI0 or SPI1
SPI0_IrqSetup    // For Interrupt mode
                 // on SPI0
SPI0_DmaSetup    // For DMA Mode on SPI0
SPI1_IrqSetup    // For Interrupt mode
                 // on SPI1
SPI1_DmaSetup    // For DMA Mode on SPI1
I2S_PollSetup    // For I2S-Poll mode on
                 // SPI0 or SPI1
I2S0_IrqSetup    // For I2S-Interrupt
                 // mode on SPI0
I2S1_IrqSetup    // For I2S-Interrupt
                 // mode on SPI1
```

fpXferDone   Optional application callback that the SPI driver calls
               when a data transfer operation completes (or aborts).
               After a successful call to the BSP_SPI_Xfer API,
               the SPI driver will return a status of BSP_ERR_BUSY
               if the BSP_SPI_Xfer API is called again before the
               driver calls the application's fpXferDone callback.
               Only one callback is made to the fpXferDone
               callback regardless of whether the application has
               requested a half-duplex or full-duplex transfer.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An◻IXYS Company

187

TxDmaBase   if the `fpSetup` member of the `BSP_SPI` structure references the `SPI0_DmaSetup` routine or the `SPI1_DmaSetup` routine and the application must perform transmit operations, then the `TxDmaBase` structure member should be set to the BSP address of the DMA channel reserved for SPI0/SPI1 data transmission. In all other instances, the `TxDmaBase` structure member should be set to 0. The only valid nonzero values for this structure member are `BSP_DMA0` to `BSP_DMA3`.

RxDmaBase   if the `fpSetup` member of the `BSP_SPI` structure references the `SPI0_DmaSetup` routine or the `SPI1_DmaSetup` routine and the application must perform receive operations, then the `RxDmaBase` structure member should be set to the BSP address of the DMA channel reserved for SPI0/SPI1 data reception. In all other instances, the `RxDmaBase` structure member should be set to 0. The only valid nonzero values for this structure member are `BSP_DMA0` to `BSP_DMA3`.

NullTxChar   When the SPI driver performs half-duplex receive transfers, it can be configured to send an arbitrary data value (the value of the NullTxChar structure member) to the remote SPI device. To enable this mode of operation the `SPI_ESPIEN1` (i.e., transmit enable) bit must be set in the Mode structure member as well as the `SPI_ESPIEN0` (i.e., receive enable) bit. If the `SPI_ESPIEN1` bit is set to 0 in the Mode structure member, then transmission is disabled at the hardware level. In this instance, the remote SPI device receives a stream of 0xFF characters.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

188

I2S_WordSize	When the slave select mode (SSMD) field in the Mode structure member is set to SPI_SSMD_I2S this structure member specifies the number of bytes per I^2S datum. In I^2S Master Mode, the SPI driver toggles nSS after each I^2S datum is transferred.
Use_nSS	If nonzero, specifies that the SPI driver should assert nSS during data transfers. This structure member should be set to FALSE (0) when the SPI driver is configured to operate in Slave or Multi-Master modes. If the SPI driver is the only master on the bus, then the application can set this structure member to TRUE (1) if the BSP driver should control nSS or FALSE (0) if the application controls nSS (or the external GPIO pin(s) used to enable specific slaves).

Correct Usage

Applications using the BSP SPI driver must declare a variable of type BSP_SPI, initialize it with valid values and pass the address of the variable to the BSP_SPI_Init API before attempting to call any other SPI API.

When configuring the SPI driver to operate in I^2S Mode, be aware of the following limitations:

- DMA data transfer is not supported in I^2S Mode. In I^2S Mode, only polling or interrupt driver data transfers can be used. Consequently, there are no I^2S DMA setup functions provided for the fpSetup structure member.

- When operating as I^2S master at high baud rates, software may not be able to toggle the slave select signal fast enough causing extra output bits in the data stream. If this occurs reduce the data rate via the Brg structure member.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An ◼ IXYS Company

189

- The I^{2}S master toggles the SPI nSS signal between left and right data words. Therefore, the BSP I^{2}S master should be configured for Single Master Mode (SSIO=1 in the mode structure member).

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

190

UART Data Structures in the BSP API

This section presents the following data structures, which are used by the BSP UART driver.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

191

BSP_UART

Definition

```
typedef struct BSP_UART_s
{
 UINT8 Mode;
 UINT8 Ctl0;
 UINT8 Ctl1;
 UINT16 Brg;
 UINT8 Addr;

 rom BSP_GPIO_CFG * pGpioCfg;

 UART_TX_CFG * pTxCfg;
 UART_RX_CFG * pRxCfg;
} BSP_UART;
```

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

192

Members

Mode   During initialization the value of the Mode structure member is written to the UxMDSTAT Register*. The upper 3 bits of the Mode structure member are used to select the UART protocol mode and must be set to one of the following values: UART_MSEL_NORMAL_MP (for standard UART or Multiprocessor Protocol Mode), UART_MSEL_LIN (for LIN Protocol Mode) or UART_MSEL_DMX (for DMX Protocol Mode). The BSP UART driver does not support DALI Mode.

Ctl0   During initialization the value of the Ctl0 structure member is written to the UxCTL0 Register*. This structure member is used to enable hardware transmit flow control (CTSE), parity enable (PEN), Parity Select (PSEL) and the number of stop bits (STOP). The setting of all other U0CTL0 control bits are ignored (TEN, REN, SBRK, and LBEN).

CTL1   During initialization the value of the Ctl1 structure member is written to the UxCTL1 Register*. The meaning of the Ctl1 control byte is dependent upon the UART protocol mode specified by the value of the Mode structure member.

Brg   During initialization the value of the Brg structure member is written to the UxBRH and UxBRL registers*.

Addr   During initialization the value of the Addr structure member is written to the UxADDR Register. This structure member is only applicable to Multiprocessor (MP) and DMX Protocol modes.

Note: *To learn more about these registers, refer to the F6482 Series Product Specification (PS0294).

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An ■IXYS Company

193

pGpioCfg	References a `BSP_GPIO_CFG` structure that specifies the GPIO port pin configuration applicable to the target UART controller. The structure referenced by the pGpioCfg pointer should be located in Flash (the compiler's ROM memory space). To provide an example, the default GPIO port pin configuration for UART0 on the Z8F6482 controller is shown in the following code snippet:

```
rom BSP_GPIO_CFG U0GpioCfg[ 2 ] =
{
 {BSP_GPIO_PORT_A, (GPIOA_TXD0 |
GPIOA_RXD0), 0, 0},
 {0,0,0,0}
};
```

pTxCfg	Optional pointer to a UART transmit configuration structure used to specify parameters required for data transmission. If the pTxCfg structure is set to `NULLPTR` `(0)`, then the application should not call the `BSP_UART_Transmit` API (i.e., the application only performs UART data reception).
pRxCfg	Optional pointer to a UART receive configuration structure used to specify parameters required for data reception. If the pRxCfg structure is set to `NULLPTR (0)`, then the application should not call the `BSP_UART_Receive` API (i.e., the application only performs UART data transmission).

Note: *To learn more about these registers, refer to the F6482 Series Product Specification (PS0294).

Correct Usage

Applications using the BSP UART driver must declare a variable of type `BSP_UART`, initialize it with valid values and pass the address of the variable to the `BSP_UART_Init` API before attempting to call any other UART API.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

194

UART_TX_CONFIG

Definition

```
typedef union UART_TX_CFG_u
{
 UART_TX_POLL Poll;
 UART_TX_IRQ Irq;
 UART_TX_DMA Dma;
} UART_TX_CFG;
```

Members

Poll	UART_TX_POLL data structure containing configuration parameters for data transmission using CPU polling.
Irq	UART_TX_IRQ data structure containing configuration parameters for data transmission using interrupt control.
Dma	UART_TX_DMA data structure containing configuration parameters for data transmission using DMA.

Correct Usage

Before calling the BSP_UART_Transmit API, applications must initialize a variable of type UART_TX_POLL, UART_TX_IRQ, or UART_TX_DMA to specify whether the BSP UART driver will transmit data using CPU polling, interrupt control or DMA. The address of this variable is then used to initialize the fpTxSetup member of the BSP_UART structure. The assignment will require the variable to be cast to (UART_TX_CONFIG *).

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An ■ IXYS Company

195

UART_RX_CONFIG

Definition

```
typedef union UART_RX_CFG_u
{
 UART_RX_POLL Poll;
 UART_RX_IRQ Irq;
 UART_RX_DMA Dma;
} UART_RX_CFG;
```

Members

Poll	UART_RX_POLL data structure containing configuration parameters for data reception using CPU polling.
Irq	UART_RX_IRQ data structure containing configuration parameters for data reception using interrupt control.
Dma	UART_RX_DMA data structure containing configuration parameters for data reception using DMA.

Correct Usage

Before calling the BSP_UART_Receive API, applications must initialize a variable of type UART_RX_POLL, UART_RX_IRQ, or UART_RX_DMA to specify whether the BSP UART driver will receive data using CPU polling, interrupt control or DMA. The address of this variable is then used to initialize the fpRxSetup member of the BSP_UART structure. The assignment will require the variable to be cast to (UART_RX_CONFIG *).

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

196

UART_TX_POLL

Definition

```
typedef struct UART_TX_POLL_s
{
 FP_TX_SETUP fpTxSetup;
} UART_TX_POLL;
```

Members

fpTxSetup A function pointer that references the hardware specific
           initialization routine that enables Poll Mode data
           transmission. Applications should set the fpTxSetup
           structure member to reference the
           BSP_UART_PollTxInit routine to enable Poll Mode data
           transmission on either UART0 or UART1.

Correct Usage

None.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An ☐IXYS Company

197

UART_TX_IRQ

Definition

```
typedef struct UART_TX_IRQ_s
{
 FP_TX_SETUP fpTxSetup;
 FP_UART_TXC fpTxC;
} UART_TX_IRQ;
```

Members

fpTxSetup   A function pointer that references the hardware specific initialization routine that enables interrupt driven data transmission. Applications should set the fpTxSetup structure member to either reference the BSP_UART0_IrqTxInit routine to enable interrupt driven data transmission on UART0 or the BSP_UART1_IrqTxInit routine to enable interrupt driven data transmission on UART1.

fpTxC   Specifies the address of the application's transmit complete callback routine. Use of the fpTxC callback is optional when using interrupt driven data transmission. If the fpTxC structure member is nonzero, the BSP UART driver calls the application callback handler routine referenced by fpTxC after the transmission completes (or aborts in error). If the fpTxC structure member is NULLPTR (0), then the BSP UART driver will not issue an application callback when the UART data transmission operations complete.

Refer to the BSP_UART_Transmit API on page 111 for the function prototype of the transmit complete callback and a description of the callback parameters.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

198

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

199

UART_TX_DMA

Definition

```
typedef struct UART_TX_DMA_s
{
 FP_TX_SETUP fpTxSetup;
 FP_UART_TXC fpTxC;
 UINT16 TxDmaBase;
} UART_TX_DMA;
```

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

200

Members

fpTxSetup A function pointer that references the hardware specific
initialization routine that enables data transmission with
DMA. Applications should set the fpTxSetup structure
member to either reference the BSP_UART0_DmaTxInit
routine to enable data transmission with DMA on UART0
or the BSP_UART1_DmaTxInit routine to enable data
transmission with DMA on UART1.

fpTxC Specifies the address of the application's transmit complete
callback routine. Use of the fpTxC callback is optional
when using data transmission with DMA. If the fpTxC
structure member is nonzero, the BSP UART driver calls
the application callback handler routine referenced by
fpTxC after the transmission completes (or aborts in error).
If the fpTxC structure member is NULLPTR (0), then the
BSP UART driver will not issue an application callback
when the UART data transmission operations complete.

TxDmaBase Specifies the DMA channel the UART driver should use
for data transmission. The value of this structure member
must be between BSP_DM0 and BSP_DMA3 and the
specified channel must not be used by any other BSP
peripheral.

Note: Refer to the BSP_UART_Transmit API on page 111 for the function prototype
of the transmit complete callback and a description of the callback parameters.

Correct Usage

None.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An ◻ IXYS Company

201

UART_RX_POLL

Definition

```
typedef struct UART_RX_POLL_s
{
 FP_RX_SETUP fpRxSetup;
} UART_RX_POLL;
```

Members

fpRxSetup A function pointer that references the hardware specific
initialization routine that enables Poll Mode data reception.
Applications should set the fpRxSetup structure member
to reference the BSP_UART_PollRxInit routine to enable
Poll Mode data reception on either UART0 or UART1.

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

202

UART_RX_IRQ

Definition

```
typedef struct UART_RX_IRQ_s
{
 FP_RX_SETUP fpRxSetup;
 FP_UART_RXC fpRxC;
 UINT8 * pRxBuf;
 BSP_SIZE RxBufSize;
} UART_RX_IRQ;
```

Members

fpRxSetup A function pointer that references the hardware specific initialization routine that enables interrupt-driven data reception. Applications should set the fpRxSetup structure member to either reference the BSP_UART0_IrqRxInit routine to enable interrupt driven data reception on UART0 or the BSP_UART1_IrqRxInit routine to enable interrupt driven data reception on UART1.

fpRxC Specifies the address of the application's receive callback routine. Use of the fpRxC callback is optional when using interrupt driven data reception. If the fpRxC structure member is nonzero, the BSP UART driver calls the application callback handler routine referenced by fpRxC as UART data is received. If the fpTxC structure member is NULLPTR (0), then the BSP UART driver will not issue an application callback when UART data is received.

The BSP UART driver calls the application callback when any of the following events occur:

- A character is received and UART driver's receive buffer (referenced by the pRxBuf member of the UART_IRQ_RX structure) is empty

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

z i l o g
Embedded in Life
An ◼ IXYS Company

203

- A character is received and transferred to the UART driver's receive buffer which completely fills the receive buffer

- A character is received but discarded because the UART driver's receive buffer is full

- A character could not be received due to a receive error detected by the UART controller

The function prototype of the receive callback is:

```
reentrant void RxCallback
(
 UINT8 Idx,
 UINT8 HwStatus,
 UINT8 State
);
```

The `Idx` parameter indicates which UART device has received data. This parameter will be in the range of 0 (BSP_NUM_UART −1).

The HwStatus parameter is the value of the UxSTAT0 Register at the time of the receive callback. Refer to the Z8F6482 Series product specification to learn more regarding the UxSTAT0 Register.

The State parameter is a bit field composed of the following flags:

BSP_UART_RX_ OVFL	This flag is set when the UART receive buffer overflows (i.e., one or more bytes of received data were discarded because there was no free space in the UART receive buffer). This bit is cleared when the BSP_UART_Receive API is called to remove data from the UART driver's receive buffer.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

204

BSP_UART_RX_FULL	This flag is set when the UART receive buffer contains exactly RxBufSize bytes of data. The value of RxBufSize is determined by the value of the corresponding member in the UART_RX_IRQ structure.
BSP_UART_RX_FLOW_OFF	This flag is not used with interrupt driven data reception.
BSP_UART_RX_AVAIL	This flag is set whenever there is at least 1 byte of data available in the UART receive buffer. Use the BSP_UART_Receive API to retrieve the Rx data.
BSP_UART_TX_BUSY	This flag is set during data transmission; reset after the transfer completes. If an application calls the BSP_UART_Transmit API while this flag is set the BSP_ERR_BUSY error code is returned to the caller.
pRxBuffer	This structure member should reference an application defined buffer that the BSP UART driver uses to hold receive data until the application calls the BSP_UART_Receive API.
RxBufSize	The size (in bytes) of the receive buffer referenced by the pRxBuffer structure member. Applications can adjust the size of the UART driver's receive buffer based on the amount of data expected between calls to the BSP_UART_Receive API.

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

205

UART_RX_DMA

Definition

```
typedef struct UART_RX_DMA_s
{
 FP_RX_SETUP fpRxSetup;
 FP_UART_RXC fpRxC;

 UINT8 * pRxBuf;
 BSP_SIZE RxBufSize;

 UINT16 RxDmaBase;

 BSP_SIZE OffThresh;
 IOReg8 RxFcPort;
 UINT8 RxFcPin;
 UINT8 RxFcIdle;
} UART_RX_DMA;
```

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

206

Members

fpRxSetup A function pointer that references the hardware specific initialization routine that enables data reception using DMA. Applications should set the fpRxSetup structure member to either reference the BSP_UART0_DmaRxInit routine to enable data reception via DMA on UART0 or the BSP_UART1_DmaRxInit routine to enable data reception via DMA on UART1.

fpRxC Specifies the address of the application's receive callback routine. Use of the fpRxC callback is optional when using data reception with DMA. If the fpRxC structure member is nonzero, the BSP UART driver calls the application callback handler routine referenced by fpRxC as UART data is received. If the fpTxC structure member is NULLPTR (0), then the BSP UART driver will not issue an application callback when UART data is received.

The BSP UART driver calls the application callback when any of the following events occur:

- A character is received and UART driver's receive buffer (referenced by the pRxBuf member of the UART_IRQ_RX structure) is empty

- A character is received and transferred to the UART driver's receive buffer which completely fills the receive buffer

- If Rx flow control is enabled (i.e., only in DMA Mode) and the RxFcPin is asserted

- A character is received but discarded because the UART driver's receive buffer is full

- A character could not be received due to a receive error detected by the UART controller

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An◻IXYS Company

207

The function prototype of the receive callback is:

```
reentrant void RxCallback
(
 UINT8 Idx,
 UINT8 HwStatus,
 UINT8 State
);
```

The `Idx` parameter indicates which UART device has received data. This parameter will be in the range of 0 (`BSP_NUM_UART` –1).

The HwStatus parameter is the value of the UxSTAT0 Register at the time of the receive callback. Refer to the Z8F6482 Series product specification to learn more regarding the UxSTAT0 Register.

The State parameter is a bit field composed of the following flags:

`BSP_UART_RX_ OVFL`	This flag is set when the UART receive buffer overflows (i.e., one or more bytes of received data were discarded because there was no free space in the UART receive buffer). This bit is cleared when the `BSP_UART_Receive` API is called to remove data from the UART driver's receive buffer.
`BSP_UART_RX_ FULL`	This flag is set when the UART receive buffer contains exactly `RxBufSize` bytes of data. The value of `RxBufSize` is determined by the value of the corresponding member in the `UART_RX_IRQ` structure.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

208

BSP_UART_RX_ FLOW_OFF	This flag is set when the BSP UART driver asserts the receive flow control pin that causes the remote UART device to stop transmitting. When this flag is set applications should call the BSP_UART_Receive API to empty the UART driver's receive buffer which will cause the driver to release the receive flow control pin and allow the peer UART device to resume data transmission.
BSP_UART_RX_ AVAIL	This flag is set whenever there is at least 1 byte of data available in the UART receive buffer. Use the BSP_UART_Receive API to retrieve the Rx data.
BSP_UART_TX_ BUSY	This flag is set during data transmission; reset when the transfer completes. If an application calls the BSP_UART_Transmit API while this flag is set the BSP_ERR_BUSY error code is returned to the caller.
pRxBuffer	This structure member should reference an application defined buffer that the BSP UART driver uses to hold receive data until the application calls the BSP_UART_Receive API.
RxBufSize	The size (in bytes) of the receive buffer referenced by the pRxBuffer structure member. Applications can adjust the size of the UART driver's receive buffer based on the amount of data expected between calls to the BSP_UART_Receive API.
RxDmaBase	Specifies the DMA channel the UART driver should use for data reception. The value of this structure member must be between BSP_DM0 and BSP_DMA3 and the specified channel must not be used by any other BSP peripheral.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An◨IXYS Company

209

OffThresh	If this structure member is nonzero and DMA is used for UART data reception, then the BSP UART driver uses flow control to reduce the likelihood of losing receive data due to a full receive buffer. The BSP UART driver activates the RxFcPin when there are fewer than OffThresh bytes of free space in the buffer referenced by pRxBuf. The RxFcPin is deactivated when the number of free bytes in the receive buffer goes above 0.5 * RxBufSize. If the transmitting device checks whether or not the RxFcPin is active before transmitting (and the BSP UART driver is able to activate the RxFcPin before the receive buffer fills) then data loss will be prevented.
	Note that the value of OffThresh is used to set the WMCNT field of the 16-bit DMA transfer count as specified in the Z8 Encore! XP F6482 Series Product Specification. Therefore, only bits 12 through 15 of OffThresh should be nonzero. To simply setting the flow control threshold, use one of the DMA_WMCNT_x macro values defined in Z8F6482_DMA_SFR.h shifted left 8 bits. For example, to set the Rx DMA flow control threshold to 20 bytes, set OffThresh equal to (DMA_WMCNT_20 << 8). The size of the receive buffer (RxBufSize) should be at least 4x the flow off threshold.
RxFcPort	Specifies the GPIO port of the pin to be used with receive flow control. This structure member should be set to a valid GPIO port in the range of BSP_GPIO_PORT_A to BSP_GPIO_PORT_J (excluding BSP_GPIO_PORT_I which is not defined for the Z8F6482 Series). This structure member is ignored if OffThresh is 0.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

210

RxFcPin	Specifies the bit position of the GPIO port pin to be used with receive flow control. This structure member should be set to a value between BIT0 and BIT7. This structure member is ignored if OffThresh is 0.
RxFcIdle	Specifies the value of the RxFcPin when it is inactive (i.e., the remote UART device is allowed to transmit). If the RxFcPin idles low set this structure member to 0; otherwise set this structure member to the same value as RxFcPin. This structure member is ignored if OffThresh is 0.

Correct Usage

None.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

211

USB Data Structures in the BSP API

This section presents the following data structures, which are used by the BSP USB driver.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

212

BSP_EP

Definition

```
typedef unsigned char enum BSP_EP_e
{
 BSP_EP0_IN,
 BSP_EP1_IN,
 BSP_EP2_IN,
 BSP_EP3_IN,
 BSP_EP0_OUT,
 BSP_EP1_OUT,
 BSP_EP2_OUT,
 BSP_EP3_OUT
} BSP_EP;
```

Usage

The first parameter of all USB endpoint functions (those that begin with BSP_USB_EpXxx) must be one of the BSP_EP enumerated constants shown above. IN endpoints are used to send data from the device to the host while OUT endpoints are used to receive data sent from the host to the device.

BSP_EP0_IN and BSP_EP0_OUT form the default control pipe. All USB devices must implement the default control pipe to process requests from the host. The BSP USB driver includes a module to process standard USB requests. Consequently, the BSP USB API does not allow the default control pipe to be disabled.

Applications typically use endpoints 1 to 3 to implement their device function.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An∎IXYS Company

213

EP_BUF_SIZE

Definition

```
typedef unsigned char enum EP_BUF_SIZE_e
{
 EP_BUF_SIZE_8,
 EP_BUF_SIZE_16,
 EP_BUF_SIZE_32,
 EP_BUF_SIZE_64,
} EP_BUF_SIZE;
```

Usage

The second parameter to the `BSP_USB_EpInit` API must be one of the `EP_BUF_SIZE`-enumerated constants shown above. The value chosen must match the buffer size specified in the endpoint's descriptor, which is included with the configuration descriptor of the `BSP_USB` structure passed to the `BSP_USB_Init` API.

The endpoint buffer size specifies the maximum size of a data packet sent or received in a single IN or OUT transaction. Applications using the BSP USB API can still send or receive larger buffers within transfers that include multiple USB transactions.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

214

BSP_USB

Definition

```
typedef struct BSP_USB_s
{
 FP_USB_DEF_ENUM fpEnum;
 FP_USB_SETUP fpSetup;

 UINT16 DmaCh[ MAX_USB_DMA ];

 /*
 * Enumeration Information
 * The first byte in a descriptor contains the length
of
 * the descriptor. A Configuration descriptor includes
 * the length of all interface, endpoint, and any class
 * or vendor specific descriptors.
 */
 USB_DEVICE_DESC * pDevDesc;
 USB_CONFIG_DESC * pCfgDesc;
 USB_STRING_DESC * pStrDesc;
 UINT8 StringsPerLangID;
 FP_USB_USR_ENUM fpUserEnum;
 FP_USB_USR_CFG fpUserConfig;
} BSP_USB;
```

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

z i l o g
Embedded in Life
An □IXYS Company

215

Members

`fpEnum`	A function pointer specifying the routine which the BSP USB driver calls to process standard USB requests (as described in Chapter 9 of the USB 2.0 Specification). Servicing these requests forms the basis of USB device enumeration. Applications that wish to use the default BSP request handler should use a value of `BSP_USB_Request` when specifying the fpEnum structure member. Alternatively applications can implement their own routine to service USB requests and use the name of that function when specifying the fpEnum member of the `BSP_USB` structure.
`fpSetup`	A function pointer that references the hardware specific initialization routine that determines the endpoint data transfer method (Poll Mode, Interrupt Mode or DMA Mode). The BSP library includes the following default initialization routines to configure Poll Mode, Interrupt Mode and DMA Mode endpoint data transfer: `USB_PollSetup`, `USB_IrqSetup`, and `USB_DmaSetup`. Applications that configure the BSP USB driver to use Poll Mode data transfer must periodically call the `BSP_USB_PollEvents` API to allow processing of USB events.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

z i l o g
Embedded in Life
An ⬛IXYS Company

216

DmaCh	The Z8F6482 USB controller can be configured to use up to 2 DMA channels for endpoint data transfer. The DmaCh structure member is an array of MAX_USB_DMA (currently defined equal to 2) used to specify which BSP DMA channel(s) are reserved for use by the USB controller. If the fpSetup function pointer references either of the USB_PollSetup or USB_IrqSetup routines both entries in the DmaCh array should be set to 0 to indicate no DMA channels are reserved for USB. If the fpSetup function pointer references the USB_DmaSetup routine, then at least one of the DmaCh entries should be set to a nonzero value to indicate which DMA channel(s) have been reserved for use by the USB controller. In this instance DmaCh array entries must be in the range of BSP_DMA0 to BSP_DMA3. If all DmaCh array entries are 0 it will not be possible to transfer endpoint data to/from the host.
pDevDesc	References the application's USB device descriptor that contains general information about the device. After the USB device attaches to the bus the host begins the process of enumeration. One of the first items the host requests during enumeration is the device descriptor. To learn more, refer to the description of the USB_DEVICE_DESC structure and the USB 2.0 Specification.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

217

pCfgDesc | References a byte array containing the concatenation of all configuration, interface, and endpoint descriptors (along with any class or vendor specific descriptors) used by all configurations within the USB device. During enumeration the host issues requests for the device's configuration descriptor(s). When returning the configuration descriptor, the USB device also returns the set of interface and endpoint descriptors used by that configuration in a single USB transfer. To learn more, refer to the description of the USB_CONFIG_DESC, USB_INTERFACE_DESC, and USB_ENDPOINT_DESC structure definitions and the USB 2.0 Specification.

pStrDesc | References an optional table of string descriptors. USB strings are composed of 16-bit UNICODE characters that can be displayed to an end user in a human readable format. The use of strings is optional within a USB device. If strings are not used, then the string index of all USB descriptors should be set to 0. To learn more, refer to the description of the USB_STRING_DESC structure and the USB 2.0 Specification.

StringsPerLangID | Strings within the string descriptor table are grouped according to their language identifier with each grouping containing StringsPerLangID strings. For example the screen descriptor table might contain 3 groups of string descriptors for the German, English, and French translations of five UNICODE strings. In this instance the StringsPerLangID member of the BSP_USB stricture would be set to 5.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

218

`fpUserEnum`	The BSP USB library includes the `BSP_USB_Request` routine that is capable of servicing standard USB device, interface, and endpoint requests. However this default enumerator is not able to service class or vendor specific requests. If the device application must support class or vendor requests than the application must specify a nonzero value for the `fpUserEnum` function pointer that references the application provided routine to service the application-specific USB class and/or vendor requests. If a non standard (or class or vendor) request is received and the `fpUserEnum` structure member is NULL the request is stalled.
	If the `fpUserEnum` function pointer is non-zero, it must reference an application-defined routine that accepts a pointer to a `USB_DEV_REQUEST` structure and return a `BSP_STATUS` value indicating wether the request was successfully processed (`BSP_ERR_SUCCESS`) or could not be completed (any other value). For more information, please refer to the `USB_DEV_REQUEST` structure definition.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An ◼ IXYS Company

219

fpUserConfig   After the USB host successfully completes enumeration it will activate one of the device configurations (or the only device configuration) referenced by the pCfgDesc structure member. As a result the BSP USB driver will call the application callback referenced by fpUserConfig (only if the fpUserConfig structure member is nonzero). This application provided callback routine should be used to initialize the set of endpoint descriptors required to implement the configuration selected by the host.

Correct Usage

Applications using the BSP USB driver must declare a variable of type BSP_USB and initialize it with valid values and pass the address of the structure to the BSP_USB_Init API before attempting to call any other USB API.

The application should specify an fpUserConfig parameter that references an application-defined routine with the following function prototype:

BSP_STATUS UserConfigFunc(UINT8 Config, UINT8 Interface, UINT8 AltSetting);

When the host application issues the standard USB Set Configuration request, the BSP USB driver calls the function referenced by the fpUserConfig member of the BSP_USB structure passed to the BSP_USB_Init API. In this instance, the Interface and AltSetting parameters will be set to 0 and the Config parameter will be a non-zero value that corresponds to one of the (or the only) configuration descriptor referenced by the fpCfgDesc member of the BSP_USB structure.

Typically, applications will need to save the values of the Config, Interface, and AltSetting parameters in global variables so that the application can detect when the host application is activating (or deacti-

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

220

vating) a particular configuration and when the host is switching between alternate settings (if applicable) within a specific interface. Applications that define only one configuration that does not include any alternate settings for any of the configuration's concurrent interfaces only need to activate all endpoints (via calls to the `BSP_USB_EpInit API`) within all interfaces when a non-zero `Config` parameter is specified.

When the host application switches configurations (or activates the only configuration defined by the device), it is implicitly activating alternate setting 0 of all interface(s) within that configuration. In this instance, the application should activate the set of endpoints specified in the endpoint descriptor(s) corresponding to alternate setting 0 (the default alternate settings) of each interface within the configuration.

If the host application issues a Set Interface request to activate one of the (mutually exclusive) alternate settings for a particular interface, the BSP USB driver will issue another call to the `fpUserConfig` routine with the same value of the `Config` parameter but with (possibly) different `Interface` and/ or `AltSetting` parameters depending on which alternate setting the host selected for a particular interface. It is up to the application to determine what, if any, modification is required to the set of endpoints previously enabled within the configuration callback routine specified by the `fpUserConfig` member of the `BSP_USB` structure.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

z i l o g
Embedded in Life
An ◻IXYS Company

221

USB_DEVICE_DESC

Definition

```
typedef struct USB_DEVICE_DESC_s
{
 UINT8 bLength;
 UINT8 bDescriptorType;
 USB_WORD bcdUSB;
 UINT8 bDeviceClass;
 UINT8 bDeviceSubClass;
 UINT8 bDeviceProtocol;
 UINT8 bMaxPacketSize;
 USB_WORD idVendor;
 USB_WORD idProduct;
 USB_WORD bcdDevice;
 UINT8 iManufacturer;
 UINT8 iProduct;
 UINT8 iSerialNumber;
 UINT8 bNumConfigurations;
} USB_DEVICE_DESC;
```

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

222

Members

bLength	Length of the descriptor in bytes. Use a value of sizeof(USB_DEVICE_DESC).
bDescriptorType	Code that identifies this as a device descriptor. Use a value of USB_DESC_DEVICE.
bcdUSB	The 16-bit little-endian, USB defined vendor identification code, a USB specification to which the device descriptor conforms. Use a value of USB_UINT16 (0x200) to indicate that this device descriptor is USB version 2.0.
bDeviceClass	Indicates the USB-defined class implemented by this device.
bDeviceSubClass	Indicates the USB-defined subclass implemented by this device. The particular USB class specification defines the set of permissible subclass codes.
bDeviceProtocol	Identifies the device-wide protocol for the implemented device class and subclass. The USB class and subclass specification define the set of supported protocols that may be used either on a device wide or per-interface basis.
bMaxPacketSize	Defines the maximum data packet size that will be used on the default control pipe (i.e., endpoint 0). Valid values of the bMaxPacketSize structure member are: 8, 16, 32, or 64 bytes. The BSP USB driver supports a maximum packet size of 64 bytes.
idVendor	16-bit little-endian, USB defined vendor identification code. For example Zilog's USB vendor ID is 1251 so the default USB demo program set the idVendor structure member to USB_UINT16(1251).

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

z i l o g®
Embedded in Life
An ￭IXYS Company

223

idProduct	16-bit little-endian, vendor/manufacturer defined code that identifies this USB product.
bcdDevice	16-bit little-endian, vendor/manufacturer code that identifies the (hardware/firmware) version of this USB device.
iManufacturer	Optional index of the USB string descriptor for the device manufacturer. If this implementation is not using string descriptors, set the value of iManufacturer to 0. To learn more, refer to the description of the USB_STRING_DESC.
iProduct	Optional index of the USB string descriptor that describes this product. If this implementation is not using string descriptors, set the value of iProduct to 0.
iSerialNumber	Optional index of the USB string descriptor containing the serial number for this device. If this implementation is not using string descriptors, set the value of iSerialNumber to 0.
bNumConfigurations	The number of configuration descriptors in the byte array referenced by the pCfgDesc of the BSP_USB structure. This value of bNumConfigurations must be at least 1.

Correct Usage

All USB devices must provide a valid device descriptor. Set the pDev-Desc member of the BSP_USB structure to reference the application's device descriptor.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

224

USB_CONFIG_DESC

Definition

```
typedef struct USB_CONFIG_DESC_s
{
 UINT8 bLength;
 UINT8 bDescriptorType;
 USB_WORD wTotalLength;
 UINT8 bNumInterfaces;
 UINT8 bConfigurationValue;
 UINT8 iConfiguration;
 UINT8 bmAttributes;
 UINT8 bMaxPower;
} USB_CONFIG_DESC;
```

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

z i l o g
Embedded in Life
An ∎IXYS Company

225

Members

bLength	Length of the descriptor in bytes. Use a value of sizeof(USB_CONFIG_DESC).
bDescriptorType	Code that identifies this as a configuration descriptor. Use a value of USB_DESC_CONFIGURATION.
wTotalLength	The 16-bit little-endian combined length of this configuration and the set of all interface-, endpoint-, and class/vendor-specific descriptors included within this configuration. Class and/or vendor specific descriptors follow the standard descriptor that they qualify. The placement of class/vendor descriptors is typically specified in the class or vendor specification.
	If a device supports more than one configuration, the set of descriptors for the second configuration (assigned a different nonzero configuration value) will immediately follow the first configuration. Similarly, the set of descriptors for configuration (n+1) immediately follow the set of descriptors for configuration (n).
bNumInterfaces	The number of interfaces supported by this configuration.
bConfigurationValue	A nonzero value the host uses to select this configuration using the Set Configuration request. If the host selects configuration 0, it is indicating that the device should disable the current configuration and return to the USB address state.
iConfiguration	Optional index of the USB string descriptor describing this configuration. If this implementation is not using string descriptors, set the value of iConfiguration to 0.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

226

bmAttributes		Bit field describing the characteristics of this configuration. The meaning assigned to each set of bits (BIT7 is the most significant bit in the byte):
	BIT7	Reserved; must be set to 1.
	BIT6	A value of 1 indicates the device is self-powered (or partially self-powered and partially bus-powered); 0 indicates the device is totally bus-powered
	BIT5	A value of 1 indicates that the device supports remote wake-up; 0 indicates remote wake-up is not supported
	BIT3 .. BIT0	Reserved; must be set to 00000'b.
bmAttributes (cont'd.)	bMaxP ower	Indicates the amount of current the bus-powered (or partially bus-powered, partially self-powered) device consumes from the USB. The current consumption is specified in units of 2 mA.

Correct Usage

All USB devices must define at least 1 configuration descriptor that includes at least 1 interface descriptor and possibly 1 or more endpoint descriptors.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An ◼IXYS Company

227

USB_INTERFACE_DESC

Definition

```
typedef struct USB_INTERFACE_DESC_s
{
 UINT8 bLength;
 UINT8 bDescriptorType;
 UINT8 bInterfaceNumber;
 UINT8 bAlternateSetting;
 UINT8 bNumEndpoints;
 UINT8 bInterfaceClass;
 UINT8 bInterfaceSubClass;
 UINT8 bInterfaceProtocol;
 UINT8 iInterface;
} USB_INTERFACE_DESC;
```

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

228

Members

`bLength`	Length of the descriptor in bytes. Use a value of sizeof(`USB_INTERFACE_DESC`).
`bDescriptorType`	Code that identifies this as an interface descriptor. Use a value of `USB_DESC_INTERFACE`.
`bInterfaceNumber`	Zero-based index of this interface. All interfaces within a configuration must have a unique bInterfaceNumber (and operate concurrently). All alternate interface descriptors have the same bInterfaceNumber (and are mutually exclusive).
`bAlternateSetting`	Zero-based index of this (alternate) interface. If a configuration only contains one interface with no alternate settings, then the bInterfaceNumber and bAlternateSetting structure members are both set to 0. The first alternate interface descriptor would also have a bInterfaceNumber value of 0 but the bAlternateSetting structure member would be set to 1. When the host selects a particular configuration, the default alternate interface (`bAlternetSetting` = 0) should be activated on all concurrent interfaces.
`bNumEndpoints`	The number of endpoints (other than endpoint 0 IN and OUT) used by this interface. The set of endpoint descriptors immediately follow their (alternate) interface descriptor.
`bInterfaceClass`	The USB defined interface class to which this interface conforms.
`bInterfaceSubclass`	The USB defined interface subclass to which this interface conforms.
`bInterfaceProtocol`	The USB defined protocol code supported by this interface.

Z8 Encore! XP® F6482 Series API Programmer's Reference Manual

zilog
Embedded in Life
An◼IXYS Company

229

iInterface   Optional index of the USB string descriptor
                    describing this interface. If this implementation is
                    not using string descriptors, set the value of
                    iInterface to 0.

Correct Usage

None.

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

230

USB_ENDPOINT_DESC

Definition

```
typedef struct USB_ENDPOINT_DESC
{
 UINT8 bLength;
 UINT8 bDescriptorType;
 UINT8 bEndpointAddress;
 UINT8 bmAttributes;
 USB_WORD wMaxPacketSize;
 UINT8 bInterval;
} USB_ENDPOINT_DESC;
```

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An ⬛IXYS Company

231

Members

bLength   Length of the descriptor in bytes. Use a value of
                 sizeof(USB_ENDPOINT_DESC).

bDescriptorType Code that identifies this as an endpoint descriptor.
                 Use a value of USB_DESC_ENDPOINT.

bEndpointAddress A bit field encoding the USB endpoint address.
                 The meaning assigned to each set of bits is:

    BIT7   Endpoint direction. A value of 1 indicates
                        an IN endpoint (data sent from device to
                        host). A value of 0 indicates an OUT
                        endpoint (data sent from host to device).

    BIT6   Reserved; must be set to 000'b.
    ..
    BIT4

    BIT3   Endpoint number (0 to 15).
    ..
    BIT0

    bmAtt Bitfield describing the characteristics of
    ribut this endpoint. The meaning assigned to
    es   each set of bits is:

    BIT7   Reserved; must be set to 0.
    ..
    BIT6

For nonisochronous endpoints:

    BIT5   Reserved; must be set to 0.
    ..
    BIT2

For Isochronous endpoints (not supported by the
BSP USB controller or driver):

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

232

BIT5 .. BIT4	Endpoint usage. Encodes one of the following values: Data (00'b), Feedback (01'b), Implicit feedback data (10'b) or Reserved (11'b).
BIT3 .. BIT2	Endpoint synchronization type. Encodes one of the following values: None (00'b), Asynchronous (01'b), Adaptive (10'b) or Synchronous (11'b).

For all endpoints:

BIT1 .. BIT0	Endpoint type. Encodes one of the following values: Control (00'b), Isochronous (01'b), Bulk (10'b) or Interrupt (11'b).
wMaxPacketSize	Defines the maximum data packet size for this endpoint. Valid values of the bMaxPacketSize structure member are: 8, 16, 32, or 64 bytes. The BSP USB driver supports a maximum packet size of 64 bytes.
bInterval	Specifies the interval used for polling the endpoint. For a full-speed interrupt endpoint the bInterval must be between 1 and 255 and specifies the maximum number of milliseconds between host transactions. The host may choose to use a polling interval shorter than bInterval.

Correct Usage

The BSP USB driver does not support isochronous endpoints and the only control endpoints that are supported are endpoint 0 OUT (and IN). Because there is never an endpoint descriptor for the default control pipe, application endpoint descriptors should only use a bmAtrributes value of 2 or 3 to indicate a bulk or interrupt endpoint respectively.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An◼IXYS Company

233

USB_STRING_DESC

Definition

```
typedef struct USB_STRING_DESC_s
{
 UINT8 bLength;
 UINT8 bDescriptorType;
 /*
 * Although the wDat structure member is declared as a
 * single 16-bit value, the actual number of 16-bit
 * characters in the wDat array is: (bLength -2) / 2.
 *
 * Typically, application code will not declare and
 * initialize a variable of type USB_STRING_DESC; but
 * will access an array of bytes as a USB_STRING_DESC
 * through a pointer.
 */
 USB_WORD wDat[1];
} USB_STRING_DESC;
```

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

234

Members

bLength	Length of the string descriptor in bytes. The bLength structure member must include the length (in bytes) of the bLength and bDescriptorType fields (a total of 2 bytes). When counting the length of the actual string characters remember that each UNICODE character is 2 bytes long.
bDescriptorType	Code that identifies this as a device descriptor. Use a value of USB_DESC_STRING.
wDat	An array of UNICODE characters containing the message string. The structure definition indicates the wDat is 2 bytes long but the actual length of the string descriptor is determined by the value of the bLength structure member. Typically, application is declare a byte-array containing the string descriptor(s) and cast the pStrDesc member of the BSP_USB structure to reference the byte array containing the actual set of string descriptors.

Correct Usage

Strings within the USB string descriptor table are grouped according to their language identifier. Each grouping contains the same number of strings that are typically a direct translation of each other. Other descriptors use a (nonzero) 1-byte identifier to select a particular string within each grouping. String index 0 is common to all language groupings and identifies the set of language identifiers within the descriptor table. The language identifiers are 16-bit little-endian values as defined by the USB Implementers Forum.

Typically, BSP USB applications use a table of 8-byte values to initialize the USB string descriptor as shown in the following example. The sample string table contains 2 string descriptors (that other USB descriptors would reference as string ID 1 and string ID 2) in 2 languages; US English (language ID 0x0409) and Standard French (language ID 0x040C).

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

235

The English representations of the strings are *one* and *two,* while the French representations are *un* and *deux*.

```
UINT8 StringDescTable[] =
{
 // String index 0 contains the supported language IDs
 6,USB_DESC_STRING, 0x09, 0x04, 0x0C, 0x04,

 // US English versions of string index 1 and 2 follow
 2+2*(3), USB_DESC_STRING, 'o',0,'n',0,'e',0,
 2+2*(3), USB_DESC_STRING, 't',0,'w',0,'o',0,

 // Std French versions of string index 1 and 2 follow
 2+2*(2), USB_DESC_STRING, 'u',0,'n',0,
 2+2*(4), USB_DESC_STRING, 'd',0,'e',0,'u',0,'x',0
};
```

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

236

USB_DEV_REQUEST

Definition

```
typedef struct USB_DEV_REQUEST_s
{
 UINT8 bmRequestType;
 UINT8 bRequest;
 USB_WORD wValue;
 USB_WORD wIndex;
 USB_WORD wLength;
} USB_DEV_REQUEST;
```

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An **IXYS** Company

237

Members

bmRequest Type	Bitfield that identifies the type of request being issued. The meaning assigned to each set of bits is:

BIT7	Direction of data transfer for this request. A value of 0 indicates a transfer from host to device; a value of 1 indicates a transfer from device to host.
BIT5 .. BIT5	Request type. Valid values for these bits are USB_REQ_TYPE_STD (00'b), USB_REQ_TYPE_CLASS (01'b), and USB_REQ_TYPE_VENDOR (10'b). Standard requests (00'b) are serviced by the BSP USB driver if the fpEnum member of the BSP_USB structure is set to BSP_USB_Request. Class and vendor requests are passed to the application provided request handler specified by the fpUserEnum member of the BSP_USB structure if the value of fpUserEnum structure member is nonzero. Otherwise the BSP USB driver returns an error to the host (protocol stall on EP0) for all class and vendor requests.
BIT4 .. BIT0	Entity targeted by this request. Valid values for these bits are: Device (0), Interface (1), Endpoint (2), and Other (3). All other values are reserved.
bRequest	USB defined code that identifies the standard device request being issued (e.g., USB_GET_DESCRIPTOR or USB_SET_CONFIGURATION). The USB Implementers Forum also defines class specific requests.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

238

wValue: 16-bit little-endian vale whose meaning is
            determine by the value of the bRequest (and
            bmRequestType) structure members.

wIndex: Additional 16-bit value whose meaning is
            determined by the value of the previous
            members of the device request.

wLength: If the specified request involves the use a
            data phase in the control transaction the
            wLength structure member typically
            identifies the (maximum) expected size of the
            data.

Correct Usage

Applications will typically just use the BSP supplied routine to service
standard USB device requests (as identified in Chapter 9 of the USB 2.0
specification) by setting the fpEnum member of the BSP_USB structure to
reference the BSP_USB_Request routine.

Applications that are not required to process class or vendor-specific
request should set the fpUserEnum member of the BSP_USB structure to
0. In this instance the BSP_USB_Request default handler for standard
USB requests will return an error to the host (protocol stall) if the host
issues a class or vendor request to this device.

Application that must service class or vendor-specific requests should
implement a handler for those requests, then set the fpUserEnum func-
tion pointer in the BSP_USB structure to reference the application pro-
vided handler. The function prototype of the request handler is shown in
the following code snippet:

```
BSP_STATUS UserEnumHandler( USB_DEV_REQUEST * pReq );
```

If the device request is successfully processed by the application, then the
routine should return BSP_ERR_SUCCESS. If the routine returns any other

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

zilog
Embedded in Life
An ∎IXYS Company

239

value, the `BSP_USB_Request` routine will return an error to the host. Refer to the `USB_Demo` sample project for an example of how to implement a class-specific request handler. This particular sample program services select Communication Device Class (CDC) requests for the purpose of implementing a USB virtual COM port.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

240

EP_CB_INFO

Definition

```
typedef struct EP_CB_INFO_s
{
 BSP_EP EpNum;
 EP_STATUS Status;
 UINT8 * pDat;
 BSP_SIZE Len;
} EP_CB_INFO;
```

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An◻IXYS Company

241

Members

EpNum BSP defined endpoint number for which the *endpoint transfer done* callback is being called. Valid values for the EpNum field range from `BSP_EP0_IN` to `BSP_EP3_OUT`.

Status Bitfield encoding the status of the BSP endpoint. The meaning assigned to each bit is:

BIT7 `EP_VALID.` A value of 1 indicates that the endpoint is valid. If the endpoint is no longer valid (BIT7=0) the application should not call the `BSP_USB_EpTrasnmit` or `BSP_USB_EpReceive` APIs.

BIT6 `EP_BUSY`. This bit gets set to 1 when the endpoint has data to send to the host or is waiting for the host to fill its receive buffer. Typically, this bit is 0 when the *endpoint transfer done* callback routine is called.

BIT5 `EP_STALL`. This bit is set if the endpoint has been stalled. The host can cause an endpoint stall by issuing a standard USB request (and can issue a different request to clear the endpoint stall; possibly after user-intervention). If a severe error occurs a USB device may also stall and endpoint (via calling the undocumented `SetEpStall` API) which is likely to require operator intervention on the host to clear (even if the undocumented `ClearEpStall` API is called).

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An **□IXYS** Company

242

BIT4 EP_SEND_ZLP. This bit is set to 1 for IN endpoint when the BSP USB driver must send a zero length packet to the host. By default the current implementation of the BSP USB driver will send a zero length packet on all IN endpoints (not just EP0 IN) when the device requests a transfer length that is an integer multiple of the endpoint maximum packet size.

pDat A pointer to an application supplied buffer that contains data received from the host (for an OUT endpoint) or points to the transmit buffer location from which point the IN transfer could not complete. For a successful IN transfer, pDat points to the byte that follows the last byte in the original transit buffer.

Len The number of bytes in the buffer referenced by pDat. For an OUT transfer pDat contains the Len bytes of data received from the host. For a successful IN transfer Len is 0; if the transfer did not complete Len is the number of bytes at the end of the application transmit buffer that were not buffered by the USB driver for transmission to the host.

Correct Usage

The fpXferDone parameter passed to the BSP_USB_EpInit API references the application callback routine that the BSP USB driver calls at the conclusion of the endpoint data transfer operation. The function the application provides for the callback should use the function prototype shown in the following code snippet:

```
reentrant void EpXferDoneCB
(
```

**Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual**

zilog
Embedded in Life
An ◻ IXYS Company

243

```
 EP_CB_INFO * pEpInfo
);
```

All OUT endpoints should specify a nonzero value for the fpXferDone
parameter so the application gets notified when data is received from the
host. IN endpoints are not required to specify an fpXferDone parameter
if it is not necessary to be notified of when the device-to-host data transfer
completes. When Poll Mode is used for endpoint data transfers, the BSP
USB driver does not issue callbacks when an IN transfer completes
(regardless of the value of the fpXferDone parameter passed to
BSP_USB_EpInit), because Poll Mode IN transfers complete before
returning from the BSP_USB_EpTransmit API.

Z8 Encore! XP® F6482 Series API
Programmer's Reference Manual

244

Customer Support

To share comments, get your technical questions answered, or report issues you may be experiencing with our products, please visit Zilog's Technical Support page at http://support.zilog.com.

To learn more about this product, find additional documentation, or to discover other facets about Zilog product offerings, please visit the Zilog Knowledge Base or consider participating in the Zilog Forum.

This publication is subject to replacement by a later edition. To determine whether a later edition exists, please visit the Zilog website at http://www.zilog.com.