8-Bit Serial Transmitter Datasheet TX8 V 3.50

Copyright © 2001-2015 Cypress Semiconductor Corporation. All Rights Reserved.

Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O and Clock)
	Digital	Analog CT	Analog SC	Flash	RAM	
CY8C29/27/24/22/21xxx, CYWUSB6953, CY8C23x33, CY8CLED02/04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8C22x45, CY8CTMA30xx, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx, CY8C21x12, CY7C64215, CY7C603XX						
	1	0	0	193	0	1

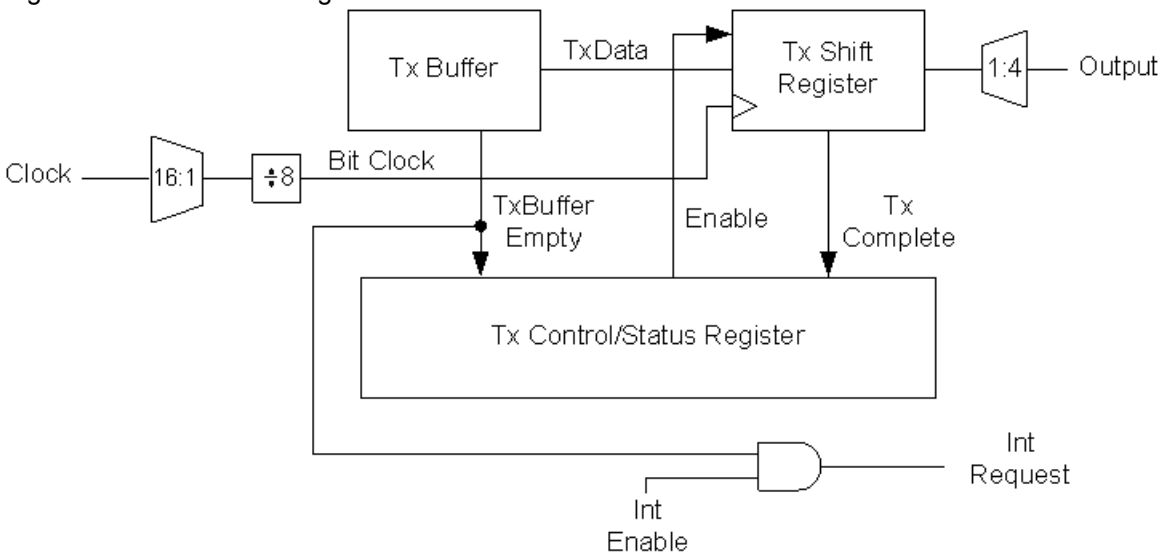
For one or more fully configured, functional example projects that use this user module go to www.cypress.com/psocexampleprojects.

Features and Overview

- 8-bit serial transmitter with selectable clocking to 48 MHz, yielding maximum 6-Mbit data rate
- Data framing consists of start, optional parity, and stop bits
- RS-232 serial-data compliant format with even, odd, or no parity
- Optional interrupt on transmit buffer empty condition

The TX8 User Module is an 8-bit RS-232 data-format compliant serial transmitter with programmable clocking and selectable interrupt or polling style operation. The data transmitted is framed with a leading start bit, an optional parity bit, and a stop bit. Transmitter firmware is used to initialize, start, stop, read status, and write data to the TX8.

Figure 1. TX8 Block Diagram



Functional Description

The TX8 User Module implements a serial transmitter. It uses the Buffer, Shift, and Control registers of a digital communications type PSoC block.

The Control register is initialized and configured, using the TX8 User Module firmware Application Programming Interface (API) routines. When the Enable bit in the Control register is set, an internal divide-by-eight bit clock is generated.

A data byte to transmit is written by an API routine into the Buffer register, clearing the Buffer Empty status bit in the Control register. This status bit can be used to detect and prevent transmit overrun errors.

The rising edge of the next bit clock transfers the data to the Shift register and sets the Buffer Empty bit of the Control register. If the interrupt enable mask is enabled, an interrupt will be triggered. This interrupt enables the queuing of the next byte to transmit, so that upon completion of transmission of the current data byte, the new byte will be transmitted on the next available transmit clock.

The start bit is transmitted at the same time that the data byte is transferred from the Buffer register to the Shift register. Successive bit clocks shift a serial bit stream to the output. The stream is composed of each bit of the data byte, least significant bit first, an optional parity bit, and a final stop bit. Upon completion of transmission of the stop bit, the Control register's Tx Complete status bit is set. This bit will remain valid until read. If a new data byte has been written to the Buffer register, the data byte will be transferred to the Shift register and transmission of the data will begin on the next rising edge of the bit clock.

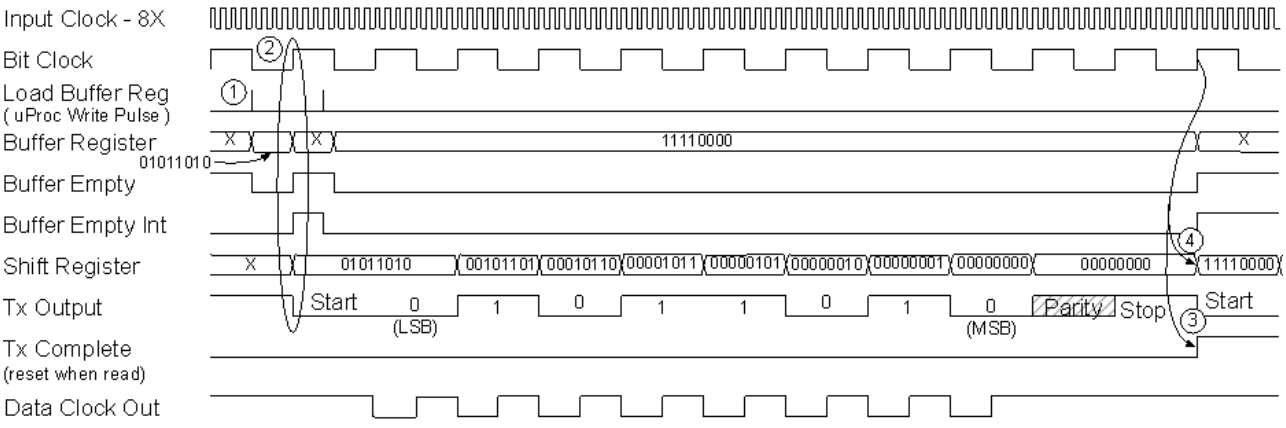
Timing

The clock rate must be set to eight times the desired bit transmit rate.

If enabled, the TX8 interrupt occurs on the Tx Buffer Empty event. This occurs when the data byte to be transmitted is transferred from the Buffer register to the Shift register. Enabling and disabling the interrupt is controlled through the API.

The following TX8 Timing diagram illustrates the operation of the TX8 User Module.

Figure 2. TX8 Timing Diagram



Communication System Accuracy

For reliable UART communication, the maximum deviation allowed in the clock source is ±4%. The IMO of the PSoC 1 has a maximum tolerance of 2.5% and, therefore, can be used. However, the 6-MHz SLIMO clock cannot be used because it has a tolerance of ±4.2%, which is not acceptable for a reliable UART communication.

Some of the PSoC 1 devices, such as CY24x94 and CY21x34, have an IMO with a tolerance of >=4%. When connected to the USB bus, the IMO is synced to the USB clock and, therefore, will become as accurate as the USB bus clock. When not connected to USB bus, it runs at ±4% tolerance. In these devices, when you place a TX8 User Module, the Design Rule Checker gives a warning. For example, when you place the TX8 User Module in CY8C24x94, the warning generated is “TX8 should not be used in the CY8C24x94 devices without connection to the USB bus”.

The system error, or the sum of the error at both ends of the communication link, should be less than 4% for the UART communication to work properly. See the device datasheets for more information about the accuracy of SysClk.

DC and AC Electrical Characteristics

Table 1. TX8 DC and AC Electrical Characteristics

Parameter	Conditions and Notes	Typical	Limit	Units
F_{max}	Maximum transmission frequency	--	6	Mbits

Placement

The TX8 User Module uses a single block, designated “TX”, which can map freely onto any of the Digital Communication PSoC blocks.

Parameters and Resources

Clock

TX8 is clocked by one of 16 possible sources. This parameter is set using the Device Editor in PSoC Designer. The Global I/O busses may be used to connect the clock input to an external pin or a clock function generated by a different PSoC block. When using an external digital clock for the block, the row input synchronization should be turned off for best accuracy, and sleep operation.The 48 MHz clock, the CPU_32 kHz clock, one of the divided clocks, 24V1 or 24V2, or another PSoC block output can be specified as the TX8 clock input.

The clock rate must be set to eight times the desired bit transmit rate. One data bit is transmitted out every 10 input clocks.

The following examples show how you can set a different transmit rate.

If the desired rate is 9600 Kbps, the clock to the TX8 User Module should be 8 x 9600 = 76.8 KHz. To get a frequency of 76.8 KHz from 24 MHz, the required divider is 312.5. Unfortunately, we cannot have a fraction in the divider and have to round off to 312 or 313. Some of the options to generate this divider are:

- SysClk = 24 MHz, VC1 divider = 8, VC3 Source = VC1, and VC3 divider = 39.
- SysClk = 24 MHz, VC1 divider = 2, VC2 divider = 4, VC3 Source = VC2, and VC3 divider = 39.
- SysClk = 24 MHz, VC1 divider = 2, VC3 Source = VC1, and VC3 divider = 156.

In all the above cases, the Clock parameter is set to VC3.

In another example, the desired rate is 19200 Kbps. Here, the clock to the TX8 User Module should be 8 x 115200 = 153.6 KHz. To get a frequency of 153.6 KHz from 24 MHz, the required divider is 156.25. In this case, the divider will be rounded off to 156. Some of the options to generate this divider are:

- SysClk = 24 MHz, VC1 divider = 4, VC3 Source = VC1, and VC3 divider = 39
- SysClk = 24 MHz, VC1 divider = 2, VC2 divider = 2, VC3 Source = VC2 and VC3 divider = 39.
- SysClk = 24 MHz, VC3 Source = SysClk/1 and VC3 divider = 156.

In all the above cases, the Clock parameter is set to VC3.

Output

The output of the transmitter can be routed to the Global Output Bus. The Global Output Bus can then be connected to an external pin or to another PSoC block for further processing.

TX Interrupt Mode

This option determines when an interrupt will be generated for the TX block. The “TxRegEmpty” option causes an interrupt to be generated, as soon as the data has been transfered from the Data register to the Shift register. Choosing the second option, “TxComplete,” delays the interrupt until the last bit is shifted out the Shift register. This second option is useful when it is important to know that the character has been completely sent. The first option, “TxRegEmpty,” is best used to maximize the output of the transmitter. It allows a byte to be loaded while the previous byte is being sent. In the interrupt service routine, the TX_CONTROL_REG should be read to enable subsequent interrupts.

ClockSync

In the PSoC devices, digital blocks may provide clock sources in addition to the system clocks. Digital clock sources may even be chained in ripple fashion. This introduces skew with respect to the system clocks. These skews are more critical in the CY8C29/27/24/22/21xxx and CY8CLED04/08/16 PSoC device families because of various data-path optimizations, particularly those applied to the system busses. This parameter may be used to control clock skew and ensure proper operation when reading and writing PSoC block register values. Appropriate values for this parameter should be determined from the following table: .

ClockSync Value	Use
Sync to SysClk	Use this setting for any 24 MHz (SysClk) derived clock source that is divided by two or more. Examples include VC1, VC2, VC3 (when VC3 is driven by SysClk), 32KHz, and digital PSoC blocks with SysClk-based sources. Externally generated clock sources should also use this value to ensure that proper synchronization occurs.
Sync to SysClk*2	Use this setting for any 48 MHz (SysClk*2) based clock unless the resulting frequency is 48 MHz (in other words, when the product of all divisors is 1).
Use SysClk Direct	Use when a 24 MHz (SysClk/1) clock is desired. This does not actually perform synchronization but provides low-skew access to the system clock itself. If selected, this option overrides the setting of the Clock parameter, above. It should always be used instead of VC1, VC2, VC3 or Digital Blocks where the net result of all dividers in combination produces a 24 Mhz output.
Unsynchronized	Use when the 48 MHz (SysClk*2) input is selected. Use when unsynchronized inputs are desired. In general this use is advisable only when interrupt generation is the sole application of the Counter.

Data Clock Out

The Data Clock Out signal is a clock signal that corresponds to the Clock input divided by 8. This clock is active only during the data bits of the transmit and will be held high all other times. The rising edge of the clock corresponds to the time when the data is stable and should be sampled. The Data Clock Out signal can be used to facilitate data checking functions such as Cyclical Redundancy Checks.

Interrupt Generation Control

The following two parameters InterruptAPI and IntDispatchMode are only accessible by setting the Enable **Interrupt generation control** check box in PSoC Designer. This is available under **Project > Settings > Chip Editor**.

InterruptAPI

The InterruptAPI parameter allows conditional generation of a User Module's interrupt handler and interrupt vector table entry. Select "Enable" to generate the interrupt handler and interrupt vector table entry. Select "Disable" to bypass the generation of the interrupt handler and interrupt vector table entry. If the Receive Command Buffer is to be used then the InterruptAPI parameter should be set to "Enable". Properly selecting whether an Interrupt API is to be generated is recommended particularly with projects that have multiple overlays where a single block resource is used by the different overlays. By selecting Interrupt API generation only when it is necessary the need to generate an interrupt dispatch code might be eliminated, thereby reducing overhead.

IntDispatchMode

The IntDispatchMode parameter is used to specify how an interrupt request is handled for interrupts shared by multiple user modules existing in the same block but in different overlays. Selecting "ActiveStatus" causes firmware to test which overlay is active before servicing the shared interrupt request. This test occurs every time the shared interrupt is requested. This adds latency and also produces a nondeterministic procedure of servicing shared interrupt requests, but does not require any RAM. Selecting "OffsetPreCalc" causes firmware to calculate the source of a shared interrupt request only when an overlay is initially loaded. This calculation decreases interrupt latency and produces a deterministic procedure for servicing shared interrupt requests, but at the expense of a byte of RAM.

Application Programming Interface

The Application Programming Interface (API) routines are provided as part of the user module to allow the designer to deal with the module at a higher level. This section specifies the interface to each function together with related constants provided by the "include" files.

Note

In this, as in all user module APIs, the values of the A and X register may be altered by calling an API function. It is the responsibility of the calling function to preserve the values of A and X prior to the call if those values are required after the call. This "registers are volatile" policy was selected for efficiency reasons and has been in force since version 1.0 of PSoC Designer. The C compiler automatically takes care of this requirement. Assembly language programmers must ensure their code observes the policy, too. Though some user module API function may leave A and X unchanged, there is no guarantee they will do so in the future.

For Large Memory Model devices, it is also the caller's responsibility to preserve any value in the CUR_PP, IDX_PP, MVR_PP, and MVW_PP registers. Even though some of these registers may not be modified now, there is no guarantee that will remain the case in future releases.

The API routines allow programmatic control of the TX8 User Module. The following tables list the low level and high level TX8 supplied API functions.

Table 2. Low Level TX8 API

Function	Description
void TX8_Start(BYTE bParity)	Enable user module and set parity.
void TX8_Stop(void)	Disable user module.
void TX8_EnableInt(void)	Enable TX interrupts.
void TX8_DisableInt(void)	Disable TX interrupts.
void TX8_SetTxIntMode(BYTE bTxIntMode)	Set the source of the TX interrupt.
void TX8_SendData(BYTE bTxData)	Send byte without checking TX status.
BYTE TX8_bReadTxStatus(void)	Return status of TX Status register.

Table 3. High Level TX8 API

Function	Description
void TX8_PutString(char * szStr)	Send NULL terminated string out TX8 port.
void TX8_CPutString(const char * azStr)	Send NULL terminated constant (ROM) string out TX8 port.
void TX8_PutChar(char bData)	Send character to TX8 port when TX register is empty. Function will not return until TX Data register can be written to without a data overrun error.
void TX8_Write(char * aStr, BYTE bCnt)	Send bCnt bytes from aStr array to TX8 port.
void TX8_CWrite(const char * aStr, int iCnt)	Send iCnt bytes from constant aStr array to TX8 port.
void TX8_PutSHexByte(BYTE bValue)	Send a two character hex repressentation of bValue to the TX8 port.
void TX8_PutSHexInt(int iValue)	Send a four character hex representation of iValue to the TX8 port.
void TX8_PutCRLF(void)	Send a carriage return (0x0D) and a line feed (0x0A) to the TX8 port.

TX8_Start

Description:

Sets the parity of the TX8 transmitter and enables the TX8 module, by setting the Tx Enable bit of the Control register. Once enabled, transmission occurs on the next Bit Clock after the Buffer register is written.

C Prototype:

```
void  TX8_Start(BYTE bParitySetting)
```

Assembly:

```
mov   A, TX8_PARITY_NONE
lcall  TX8_Start
```

Parameters:

bParitySetting: One byte that specifies the transmit parity. Symbolic names provided in C and assembly, and their associated values, are given in the following table.

Symbolic Name	Value
TX8_PARITY_NONE	0x00
TX8_PARITY_EVEN	0x02
TX8_PARITY_ODD	0x06

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C27x66, CY8C29xxx, and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

TX8_Stop

Description:

Disables the TX8 module, by clearing the Control register Tx Enable bit.

C Prototype:

```
void  TX8_Stop(void)
```

Assembly:

```
lcall  TX8_Stop
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C27x66, CY8C29xxx, and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

TX8_EnableInt

Description:

Enables the TX8 interrupt on the Tx Buffer Empty condition, by setting the appropriate enable bit in the Digital PSoC Block Interrupt Mask Register. The placement location of the TX8 determines the specific interrupt vector and priority. For details, see the PSoC datasheet for the specific part chosen.

C Prototype:

```
void  TX8_EnableInt(void)
```

Assembly:

```
lcall  TX8_EnableInt
```

Parameters:

None

Return Value:

None

Side Effects:

If an interrupt is pending and this API is called, the interrupt will be triggered immediately. This call should be made prior to calling Start(). The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C27x66, CY8C29xxx, and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

TX8_DisableInt

Description:

Disables the TX8 Interrupt on TxBuffer Empty, by clearing the appropriate enable bit in the Digital PSoC Block Interrupt Mask Register.

C Prototype:

```
void  TX8_DisableInt(void)
```

Assembly:

```
lcall  TX8_DisableInt
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C27x66, CY8C29xxx, and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

TX8_SetTxIntMode

Description:

Set the source of the interrupt.

C Prototype:

```
void  TX8_SetTxIntMode(BYTE bTxIntMode)
```

Assembly:

```
lcall  TX8_SetTxIntMode
```

Parameters:

bTxIntMode: One byte that specifies the interrupt mode. Symbolic names provided in C and assembly, and their associated values are given in the following table.

TX Interrupt Mode	Value
TX8_INT_MODE_TX_REG_EMPTY	0x00
TX8_INT_MODE_TX_COMPLETE	0x01

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C27x66, CY8C29xxx, and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

TX8_SendData

Description:

Initiates data transmission by loading the Buffer register with specified data to transmit. The TX Complete status bit, in the Control register, should be monitored to make sure transmission was initiated.

C Prototype:

```
void  TX8_SendData(BYTE bTxData)
```

Assembly:

```
mov   A, bTxData
lcall  TX8_SendData
```

Parameters:

bTxData: Data to be loaded into the TX Buffer register. Passed in the Accumulator.

Return Value:

None

Side Effects:

If the interrupt on the Tx Buffer Empty condition is set up, an interrupt will be generated when bTxData is transferred from the Buffer register to the Shift register. The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C27x66, CY8C29xxx, and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

TX8_bReadTxStatus

Description:

Returns the contents of the Control register.

C Prototype:

```
BYTE TX8_bReadTxStatus(void)
```

Assembly:

```
lcall TX8_bReadTxStatus
and  A, TX8_TX_COMPLETE
jnz  TxIsComplete
```

Parameters:

None

Returns:

Returns status byte read. Utilize defined masks to test for specific status conditions. Note that masks can be OR'ed together to check for multiple conditions.

RX Status Masks	Value
TX8_TX_COMPLETE	0x20
TX8_TX_BUFFER_EMPTY	0x10

Side Effects:

Status bits are cleared after performing the read. The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C27x66, CY8C29xxx, and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

TX8_PutString

Description:

Sends a null terminated (RAM) string to the TX8 port.

C Prototype:

```
void TX8_PutString(char * szRamString)
```

Assembler:

```
mov  A,>szRamString     ; Load MSB part of pointer to RAM based null
                        ; terminated string.
mov  X,<szRamString     ; Load LSB part of pointer to RAM based null
                        ; terminated string.
lcall TX8_PutString      ; Call function to send string out TX8 port
```

Parameters:

char * aRamString: Pointer to the string to be sent to the TX8 port. MSB is passed in the Accumulator and LSB is passed in X register.

Return Value:

None

Side Effects:

Program flow stays in this function until the last character is loaded into the TX8 transmit buffer. The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C27x66, CY8C29xxx, and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the IDX_PP page pointer register is modified.

TX8_CPutString

Description:

Sends constant (ROM), null terminated string out the TX8 port.

C Prototype:

```
void TX8_CPutString(const char * szROMString)
```

Assembler:

```
mov  A,>szRomString   ; Load MSB part of pointer to ROM based null
                      ; terminated string.
mov  X,<szRomString   ; Load LSB part of pointer to ROM based null
                      ; terminated string.
lcall TX8_CPutString   ; Call function to send constant string out
                      ; TX8 port
```

Parameters:

const char * szROMString: Pointer to string to be sent to the TX8 port. MSB of string pointer is passed in the Accumulator and LSB of the pointer is passed in the X register.

Return Value:

None

Side Effects:

Program flow stays in this function, until the last character is loaded into the TX8 transmit buffer. The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C27x66, CY8C29xxx, and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

TX8_PutChar

Description:

Writes single character to TX8 port when the port buffer is empty.

C Prototype:

```
void TX8_PutChar(CHAR cData)
```

Assembler:

```
mov  A,0x33           ; Load ASCII character "3" in A
lcall TX8_PutChar      ; Call function to send single character to
                      ; TX8 port.
```

Parameters:

CHAR cData: Character to be sent to the TX8 port. Data is passed in the Accumulator.

Return Value:

None

Side Effects:

Program flow stays in this function until the data can be written to the TX8 buffer. The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C27x66, CY8C29xxx, and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

TX8_Write

Description:

Sends ‘n‘ characters (RAM) to TX8 port.

C Prototype:

```
void TX8_Write(char * szRamString, BYTE bCount)
```

Assembler:

```
mov   A,20              ; Load string/array count
push  A
mov   A,>szRamString    ; Load MSB part of pointer to RAM string
push  A
mov   A,<szRamString    ; Load LSB part of pointer to RAM string
push  A
mov   X,SP              ; Set X register to point to variables
dec   X
lcall  TX8_Write         ; Make call to function
add   SP,253            ; Reset stack pointer to original position
```

Parameters:

CHAR * szRamString: Pointer to the string to be sent to the TX8 port.

BYTE bCount: Number of characters to be sent to the TX8.

Return Value:

None

Side Effects:

Program flow stays in this function until the last character is loaded into the TX8 buffer. The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C27x66, CY8C29xxx, and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the IDX_PP page pointer register is modified.

TX8_CWrite

Description:

Sends n characters (ROM) to the TX8 port.

C Prototype:

```
void TX8_CWrite(const char * szRomString, INT iCount)
```

Assembler:

```
mov   A,0                ; Load MSB of string/array count
push  A
mov   A,20               ; Load LSB of string/array count
push  A
mov   A,>szRomString     ; Load MSB part of pointer to ROM string
push  A
mov   A,<szRomString     ; Load LSB part of pointer to ROM string
push  A
mov   X,SP               ; Set X register to point to variables
dec   X
lcall  TX8_CWrite         ; Make call to function
add   SP,252             ; Reset stack pointer to original position
```

Parameters:

const char * szRomString: Pointer to the string to be sent to the TX8 port.

int iCount: Number of characters to be sent to the TX8 port.

Return Value:

None

Side Effects:

Program flow stays in this function until the last character is loaded into the TX8 buffer. The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C27x66, CY8C29xxx, and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

TX8_PutSHexByte

Description:

Sends two byte ASCII Hex representation of data to the TX8 port.

C Prototype:

```
void TX8_PutSHexByte(BYTE bData)
```

Assembler:

```
mov  A,0x33              ; Load data to be sent to TX8
lcall TX8_PutSHexByte     ; Call function to output hex
                         ; representation of data.  The output
                         ; for this value would be “33”.
```

Parameters:

BYTE bData: Byte to be converted to ASCII string.

Return Value:

None

Side Effects:

Program flow stays in this function, until the last character is loaded into the TX8 buffer. The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C27x66, CY8C29xxx, and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

TX8_PutSHexInt

Description:

Sends four byte ASCII Hex representation of data to the TX8 port.

C Prototype:

```
void TX8_PutSHexInt(INT iData)
```

Assembler:

```
mov  A,0x34           ; Load LSB in A
mov  X,0x12           ; Load MSB in X

lcall TX8_PutSHexInt   ; Call function to output hex
                      ; representation of data.  The output
                      ; for this value would be “1234”.
```

Parameters:

int iData: Integer to be converted to ASCII string.

Return Value:

None

Side Effects:

Program flow stays in this function, until the last character is loaded into the TX8 buffer. The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C27x66, CY8C29xxx, and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

TX8_PutCRLF

Description:

Function to send out carriage return (0x0D) and line feed (0x0A) out of the TX8 port.

C Prototype:

```
void TX8_PutCRLF(void)
```

Assembler:

```
lcall TX8_PutCRLF          ; Send a carriage return and line feed out TX
```

Parameters:

None

Return Value:

None

Side Effects:

Program execution stays in this function, until all characters have been written to the TX8 buffer. The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C27x66, CY8C29xxx, and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

Sample Firmware Source Code

The following code is an example of a function that transmits a RAM resident zero-terminated string using the TX8 module. The code sample in assembly language is:

```
;
;  This sample shows how to transmit the data byte without checking TX status.
;
; OVERVIEW:
;
;  The TX8 output can be routed to any pin.
;  In this example the TX8 output is routed to P0[0].
;
;The following changes need to be made to the default settings in the Device Editor:
;
;   1. Select TX8 user module.
;   2. The User Module will occupy the space in dedicated system resources.
;   3. Rename User Module's instance name to TX8.
;   4. Set TX8's Clock Parameter to VC2.
;   5. Set TX8's Output Parameter to Row_0_Output_0.
;   6. Set TX8's TX Interrupt ModeParameter to TXComplete.
;   8. Set TX8's ClockSync Parameter to SyncSysClk.
;   6. Set TX8's Data Clock Out Parameter to None.
;   7. Click on Row_0_Output_0 and connect Row_0_Output_0 to GlobalOutEven_0.
;   8. Select GlobalOutEven_0 for P0[0] in the Pinout.
;
; CONFIGURATION DETAILS:
;
;  1. The UM's instance name must be shortened to TX8.
;
; PROJECT SETTINGS:
;
;     CPU_Colck  = 1.5_MHz(SysClk/16)     System clock is set to 1.5MHz
;     VC1=SysClk/N  = 16 (default)
;     VC2=VC1/N  = 16 (default)
;
; USER MODULE PARAMETER SETTINGS:
;
;  ------------------------------------------------------------------------
;  UM        Parameter          Value                 Comments
;  ------------------------------------------------------------------------
;  TX8       Name               TX8                  UM's instance name
;            Clock              VC2
;            Output             Row_0_Output_0
;            TX Interrupt Mode  TXComplete
```

```
;             ClockSync          SyncSysClk
;             Data Clock Out   None
;
;  ------------------------------------------------------------------------

; Code begins here

include "m8c.inc"       ; part specific constants and macros
include "memory.inc"    ; Constants & macros for SMM/LMM and Compiler
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

export _main

AREA bss (RAM, REL)
bTxData:          blk 1

AREA text(ROM,REL,CON)

_main:
    M8C_EnableGInt ; Enable Global Interrupts

    mov    A, TX8_PARITY_NONE
    lcall  TX8_Start
    mov [bTxData], 0xAA
    mov A, [bTxData]
    lcall  TX8_SendData  ; Send data
.terminate:
    ; Insert your main assembly code here.
    jmp .terminate
```

The same code written in C is:

```
//
//  This sample shows how to transmit the data buffer without checking TX status.
//
// OVERVIEW:
//
//  The TX8 output can be routed to any pin.
//  In this example the TX8 output is routed to P0[0].
//
//The following changes need to be made to the default settings in the Device Editor:
//
//   1. Select TX8 user module.
//   2. The User Module will occupy the space in dedicated system resources.
//   3. Rename User Module's instance name to TX8.
//   4. Set TX8's Clock Parameter to VC2.
//   5. Set TX8's Output Parameter to Row_0_Output_0.
//   6. Set TX8's TX Interrupt ModeParameter to TXComplete.
//   8. Set TX8's ClockSync Parameter to SyncSysClk.
//   6. Set TX8's Data Clock Out Parameter to None.
//   7. Click on Row_0_Output_0 and connect Row_0_Output_0 to GlobalOutEven_0.
//   8. Select GlobalOutEven_0 for P0[0] in the Pinout.
//
// CONFIGURATION DETAILS:
//
```

```
//  1. The UM's instance name must be shortened to TX8.
//
// PROJECT SETTINGS:
//
//     CPU_Colck  = 1.5_MHz(SysClk/16)     System clock is set to 1.5MHz
//     VC1=SysClk/N  = 16 (default)
//     VC2=VC1/N  = 16 (default)
//
// USER MODULE PARAMETER SETTINGS:
//
//  ------------------------------------------------------------------------------
//  UM        Parameter          Value                 Comments
//  ------------------------------------------------------------------------------
//  TX8       Name               TX8                  UM's instance name
//            Clock              VC2
//            Output             Row_0_Output_0
//            TX Interrupt Mode  TXComplete
//            ClockSync          SyncSysClk
//            Data Clock Out   None
//
//  ------------------------------------------------------------------------------

/* Code begins here */

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules
BYTE TxDataBuffer[8] = {0,1,2,3,4,5,6,7};

void TxZeroTerminatedRamString( BYTE *pbStrPtr )
{
    /* check for the end condition, before sending the next byte */
    while( *pbStrPtr != 0 )
    {
        /* send the next byte */
        TX8_SendData( *pbStrPtr );

        /* Wait for the data to start transmitting */
        while( !( TX8_bReadTxStatus() & TX8_TX_BUFFER_EMPTY ) );

        pbStrPtr++;
    }
}

void main(void)
{
    M8C_EnableGInt ; // Enable Global Interrupts

    TX8_EnableInt(); // Enable TX interrupts
    TX8_Start(TX8_PARITY_NONE); // Enable user module and set parity

    TxZeroTerminatedRamString(TxDataBuffer); // Send data buffer
    while(1)
    {
    // Insert your main routine code here.
    }
```

```
}
```

Configuration Registers

The Digital Communication Type A PSoC block registers used to configure a TX8 User Module are described below. Only the parameterized symbols are explained.

Table 4. Block TX, Register: Function

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	1	1	0	1

This register defines the personality of this Digital Communications Type ‘A’ Block to be a TX8 User Module.

Table 5. Block TX, Register: Input

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	Clock			

Clock is the selected clock to drive the transmitter timing. It is set using the Device Editor during parameter selection.

Table 6. Block TX, Register: Output

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	TX Output Bus		

Output Enable and Select specifies the output of the TX8. It is set using the Device Editor during parameter selection.

Table 7. Block TX, Data Shift Register: DR0

Bit	7	6	5	4	3	2	1	0
Value	TX8 Shift Register							

TX8 Shift Register is controlled by the TX8 state machine hardware to shift the contents and transmit the least significant bit. Data is loaded into this register from the DR1 Data register, by the TX8 state machine hardware.

Table 8. Block TX, Data Buffer Register: DR1

Bit	7	6	5	4	3	2	1	0
Value	TX8 Buffer Register							

TX Buffer Register is used to transmit data that is written to this register by the user module APIs. The data loaded into this register is transferred to the TX Shift register by the TX state machine.

Table 9. Block TX, Data Register: DR2

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0

This register is not used.

Table 10. Block TX, Control/Status Register: CR0

Bit	7	6	5	4	3	2	1	0
Value	Rsvd	Rsvd	TX Complete	TX Reg Empty	Rsvd	Parity Type	Parity Enable	TX Enable

Tx Complete is a flag that indicates if a data byte is in the process of being transmitted. This bit is reset when the register is read. It can be accessed by way of one of the API functions. Tx Reg Empty is a flag that indicates when the Buffer register is empty. It can be accessed by way of one of the API functions. Parity Type is the type of parity to compute. This bit is a “don’t care if Parity Enable bit is not set.” It can be set by way of one of the API functions or from the Device Editor. Parity Enable enables or disables the computation and transmission of a parity bit with data byte. Parity is selected by setting the Parity Type bit. It can be set by way of one of the API functions or from the Device Editor. Tx Enable enables or disables the TX transmitter. It can be set by way of one of the API functions.

Version History

Version	Originator	Description
3.4	DHA	Updated Clock description to include: When using an external digital clock for the block, the row input synchronization should be turned off for best accuracy, and sleep operation. Added DRC that informs the user not to use the 32 kHz option unless an external crystal is used.
3.50	DHA	Added support for CY8C21x12 devices.

Note PSoC Designer 5.1 introduces a Version History in all User Module datasheets. This section documents high level descriptions of the differences between the current and previous user module versions.

Copyright © 2001-2015 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer™ and Programmable System-on-Chip™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.