8-Bit Serial Receiver Datasheet RX8 V 3.50

Copyright © 2002-2015 Cypress Semiconductor Corporation. All Rights Reserved.

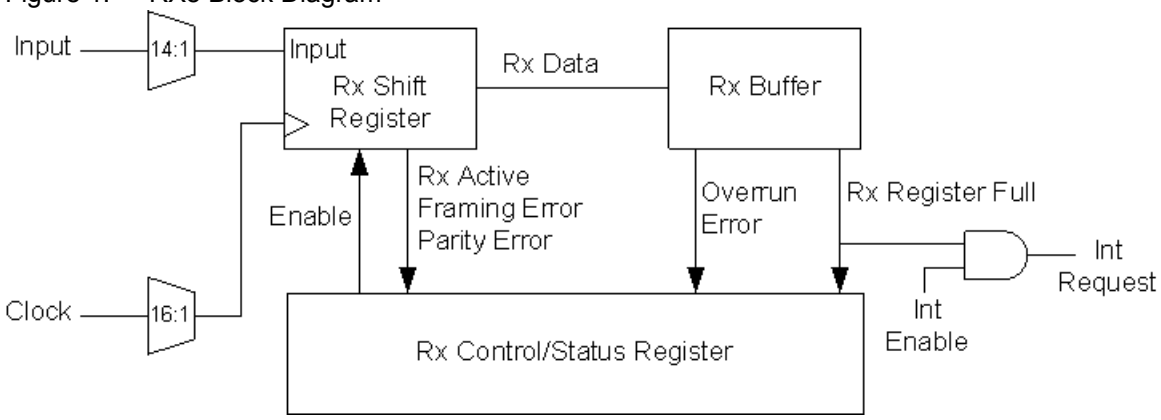
Resources	PSoC® Blocks			API Memory (Bytes)		Pins
	Digital	Analog CT	Analog SC	Flash	RAM	
CY8C29/27/24/22/21xxx, CY8C23x33, CYWUSB6953, CY8CLED02/04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8C22x45, CY8CTMA140, CY8CTST300, CY8CTMG300, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx, CY8C21x12, CY7C64215, CY7C603XX						
RxCmdBuf Enabled	1	0	0	273	3 + Buffer	1
RxCmdBuf Disabled	1	0	0	77	0	1

Features and Overview

- Burst rates up to 6 Mbits/second
- RS-232 data-format compliant with framing consisting of start, optional parity, and stop bits
- Serial data format with even, odd, or no parity
- Optional interrupt receive register full condition
- Automatic framing, overrun, and parity error detection

The RX8 User Module is a RS-232 data-format compliant 8-bit serial receiver with programmable clocking and selectable interrupt or polling control operation. The format of the received data consists of a start bit, an optional parity bit, and a trailing stop bit. Receiver firmware is used to initialize the device, read the received byte, and detect error conditions.

Figure 1. RX8 Block Diagram



Functional Description

The RX8 User Module implements a serial receiver. The RX8 maps onto a single PSoC Digital Communication block designated “RX” in the PSoC Designer Device Editor. It uses the Buffer, Shift and Control registers of a digital communications type PSoC block.

The Control register is initialized and configured using the RX8 User Module firmware Application Programming Interface (API) routines. Initialization of the RX8 consists of setting the parity, optionally enabling the interrupt on the Rx Register Full condition, and then enabling the receiver.

When a start bit is detected on the RX8 input, a divide-by-eight bit clock is started and synchronized to sample the data in the center of the received bits. On the rising edge of the next eight-bit clocks, the input data is sampled and shifted into the Shift register. If parity is enabled, the next bit clock samples the parity bit. The sampling of the stop bit, on the next clock, results in the received data byte transfer to the Buffer register and the triggering of one or more of the following events:

- Rx Register Full bit in the Control register is set, and if the interrupt for the RX8 is enabled, then the associated interrupt is triggered.
- If the stop bit is not detected at the expected bit position in the data stream, then the Framing Error bit in the Control register is set.
- If the Buffer register has not been read, before the stop bit of the currently received data, then the Overrun Error bit in the Control register is set.
- If a parity error was detected, then the Parity Error bit is set in the Control register.

For polling detection of a completely received data byte, the Rx Register Full bit in the Control register should be monitored. Data must be read out of the Buffer register, before the next byte is completely received, to prevent an overrun error condition.

High-Level API

The high-level API adds additional firmware on top of the basic functions, to provide command and string level functions instead of character level. The Device Editor allows the user to set the size of the receiver command buffer, command terminator, parameter delimiter, and below what value the receiver should ignore characters.

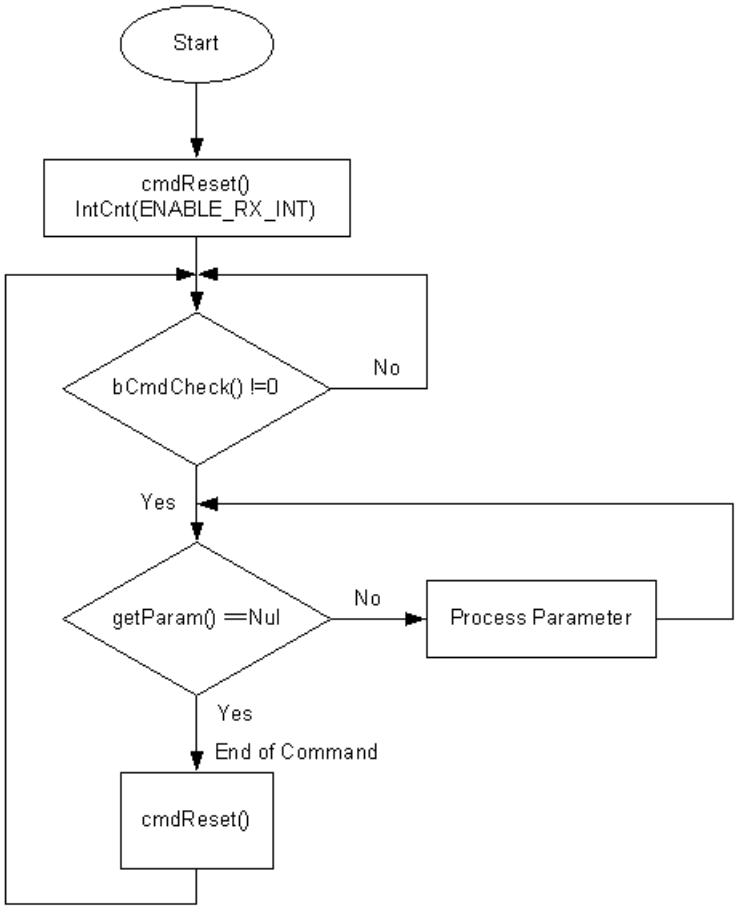
To make use of the high-level receiver functions, go to the Device Editor window, select the UART and select the “Enable” option for the “RxCmdBuffer” parameter. Next, select a “RxBufferSize” that is large enough to hold your largest command plus one. Select a command terminator character “CommandTerminator.” This is most often set to be either a carriage return (13), or a line feed (10). If your commands contain two or more parameters, select a parameter delimiter “Param_Delimiter.” Common command delineators are usually a space (32) or a comma (44) character. Control characters below a selected value may also be ignored. Most control characters are in the range between 0 and 31. Set “IgnoreCharsBelow” to 32 to ignore these characters. Set this parameter to ‘1‘ if all characters are valid. The character selected for the command terminator (CommandTerminator) is not affected by the “IgnoreCharsBelow” option. The flow chart below shows the proper sequence for basic operation of the command buffer functions.

The command buffer works in the UART RX interrupt service routine to collect the characters in a buffer until the command terminator character is received. At that time, a flag is set to signal that the command buffer is ready to be read. The receive buffer may be accessed directly by reading the array INSTANCE_NAME_aRxBuffer or by using either szGetParam() or szGetRestOfParams() functions. If more characters are received than the buffer_size -1, the subsequent characters will be ignored.

The command buffer will collect characters until the buffer is full or a command terminator is detected. Any characters received after either of these two conditions will be ignored, until the CmdReset command is

executed. Once the CmdReset command is executed, the RX ISR firmware will begin to collect characters once again.

Figure 2. Command Buffer Flow



Timing

Each received bit is sampled on the rising edge of a generated divide-by-eight bit clock that is synchronized with the center of the start bit.

If enabled, the RX8 interrupt occurs once per received byte. Enabling and disabling the interrupt is controlled through the API.

The following RX8 timing diagram illustrates the operation of the RX8 User Module.

Figure 3. RX8 Timing Diagram

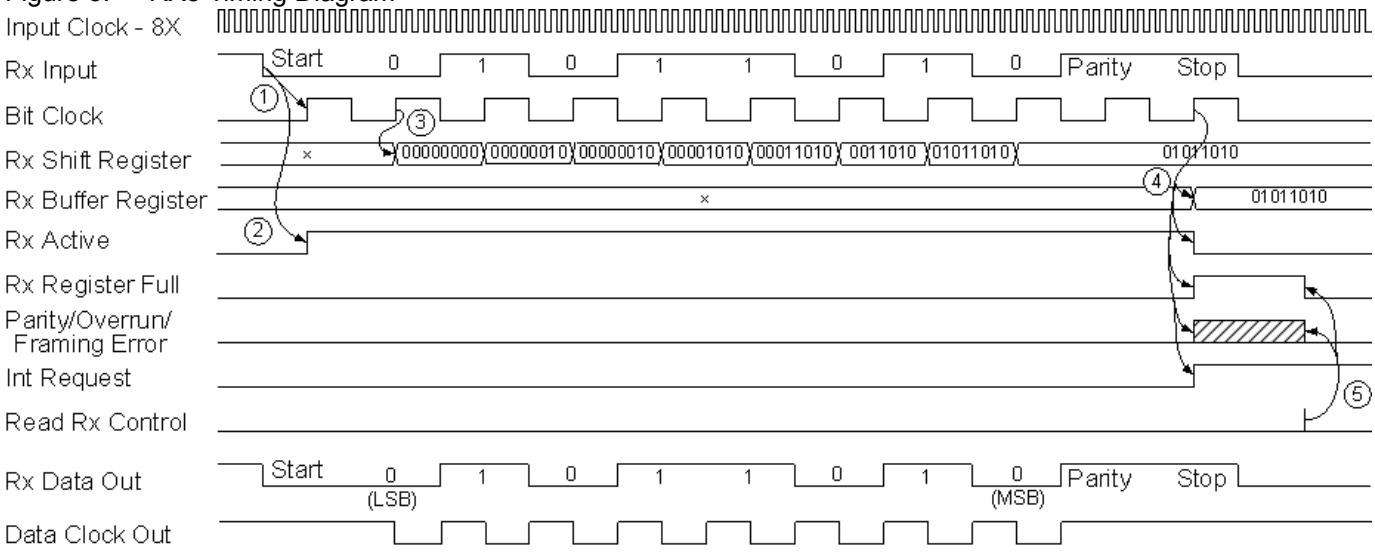


Figure Notes

1. The start bit is detected, causing the generation and synchronization of a divide-by-eight bit clock. The rising edge of the bit clock is used to sample the input bit stream.
2. The detection of the start bit causes the Rx Active status bit to be set in the Rx Control/Status register.
3. Beginning with the next bit clk rising edge and for the next 8 clocks, the input is sampled and shifted into the Rx Shift register.
4. When the stop bit is detected, data is transferred into the Rx Buffer, Rx Active is cleared, Rx Register Full is set, parity/overrun/framing error are computed, and the interrupt request is triggered if enabled.
5. A read of the Rx Control register puts the status data onto the data bus and clears all of the status bits. When parity or overflow errors are detected, the firmware should not need to perform any actions. If a framing error is detected, the framer will immediately begin looking for the next data byte. If there's a chance in the system that the RX input could be stuck at logic 0 for an extended period of time, the receiver should be stopped and the line should be polled for a return to logic 1 before the re-enabling of receiver. If the RX input is stuck at logic 0, this avoids repeated detection of 'false' start bits while the line is logic 0 and resulting framing errors when a logic 0 is also detected where the start bit should be.

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, becomes 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 RX8 User Module, the Design Rule Checker gives a warning. For example, when you place the RX8 User Module in CY8C24x94, the warning generated is “RX8 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. RX8 DC and AC Electrical Characteristics

Parameter	Conditions and Notes	Typical	Limit	Units
F_{max}	Maximum receive frequency		6	Mbits/s

Placement

The RX8 User Module may be placed in any of the Digital Communication blocks.

Parameters and Resources

Clock

RX8 is clocked by one of 16 possible sources. 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 RX8 clock input.

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

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

If the desired rate is 9600 Kbps, the clock to the RX8 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 RX8 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.

Input

As a general rule, input through desired bus option to a source of asynchronous data. Using a global bus, the input can be connected to one of the external pins.

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.

RX Output

This parameter allows the Input signal to be routed to one of the row busses. This signal along with the Data Clock Out can be used to facilitate data verification functions such as Cyclical Redundancy Checks using the CRC16 User Module.

Data Clock Out

This parameter allows the bit clock in SPI Mode 3 to be routed to one of the row busses. The bit clock is the Clock input divided by eight. The rising edge of the Data Clock Out signal corresponds to the time when the data is stable and should be sampled. Use the signal along with the RX Output to use data verification functions such as Cyclical Redundancy Checks using the CRC16 User Module.

RxCmdBuffer

This parameter enables the receive command buffer and firmware used for command processing. The command buffer works in the UART RX interrupt service routine to collect the characters in a buffer until the command terminator character is received. After the terminator character is received, a flag is set to signal that the command buffer is ready to be read. The UART RX interrupt must be enabled for the command buffer to operate.

RxBufferSize

This parameter determines how many RAM locations are reserved for the receive buffer. The largest command that can be received is one less than the buffer size selected, since the string must be null terminated. This parameter is only valid when the RxCmdBuffer is enabled and the UART RX interrupt is enabled.

InvertInput

This parameter allows the user to invert the RX input signal. This option may be used for certain RS232 transceivers that produce an inverted output.

CommandTerminator

This parameter selects the character that signals the end of a command. When received, a flag is set signaling a complete command has been received. Once this flag is set, additional characters are no longer accepted until the cmdReset() function is called.

Param_Delimiter

This parameter selects the character used to delimit the command and parameters in the command receiver buffer. For example, if the Param_Delimiter is set to a space character (32), each substring separated by a space would be a parameter. Given the string ‘cmd foo bar c‘, the parameters would be ‘cmd‘, ‘foo‘, ‘bar‘, and ‘c‘. Each call of szGetParam() returns a pointer to the next substring in order of placement from left to right as a null terminated string.

IgnoreCharsBelow

This parameter enables characters below a set value to be ignored by the receive buffer. The characters will be received, but will not be added to the receive buffer. This parameter is only valid when the RxCmdBuffer is enabled and the UART RX interrupt is active.

Interrupt Generation Control

There are two additional parameters that become available when the **Enable interrupt generation control** check box in PSoC Designer is checked. This is available under **Project > Settings > Chip Editor**. Interrupt Generation Control is important when multiple overlays are used with interrupts shared by multiple user modules across overlays:

- Interrupt API
- IntDispatchMode

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 RX8 User Module. The following tables list the low level and high level RX8supplied API functions.

Table 2. Low Level RX8 API

Function	Description
void RX8_Start(BYTE parity)	Enable user module and set parity.
void RX8_Stop(void)	Disable user module.
void RX8_EnableInt(void)	Enable interrupts.
void RX8_DisableInt(void)	Disable interrupts.
BYTE RX8_bReadRxData(void)	Return data in RX Data register without checking status of character is valid.
BYTE RX8_bReadRxStatus(void)	Check status of RX Status register.

Table 3. High Level RX8 API

Function	Description
char RX8_cGetChar(void)	Return character from RX Data register when valid data is available. Function will not return until character is received.
char RX8_cReadChar(void)	Read RX Data register immediately. If valid data not available, return 0, otherwise ASCII char between 1 and 255 is returned.
int RX8_ iReadChar(void)	Read Rx Data register immediately. If data is not available or an error condition exists, return an error status in the MSB. The received char is returned in the LSB.
void RX8_CmdReset(void)	Reset Rx command buffer.
BYTE RX8_bCmdCheck(void)	Returns a non-zero value if a valid command terminator has been received.
BYTE RX8_bCmdLength(void)	Returns the current command length.
char * RX8_szGetParam(void)	Return pointer to next parameter in RX buffer.
char * RX8_szGetRestOfParams(void)	Return pointer to remaining parameter string.
BYTE RX8_bErrCheck(void)	Return command buffer error status.

RX8_Start

Description:

Sets the parity of the RX8 receiver and enables the RX8 module, by setting the Rx Enable bit of the Control register.

C Prototype:

```
void  RX8_Start(BYTE bParitySetting)
```

Assembly:

```
mov   A, RX8_PARITY_NONE
lcall  RX8_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
RX8_PARITY_NONE	0x00
RX8_PARITY_EVEN	0x02
RX8_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 (CY8C29xxx and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

RX8_Stop

Description:

Disables the RX8 module by clearing the Control register Enable bit.

C Prototype:

```
void  RX8_Stop(void)
```

Assembly:

```
lcall RX8_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 (CY8C29xxx and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

RX8_EnableInt

Description:

Enables the RX8 interrupt on the Receive Register Full condition by setting the appropriate enable bit in the Digital PSoC Block Interrupt Mask register.

C Prototype:

```
void  RX8_EnableInt(void)
```

Assembly:

```
lcall  RX8_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 (CY8C29xxx and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

RX8_DisableInt

Description:

Disables the RX8 Interrupt on Receive Register Full by clearing the appropriate enable bit in the Digital PSoC Block Interrupt Mask register.

C Prototype:

```
void  RX8_DisableInt(void)
```

Assembly:

```
lcall  RX8_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 (CY8C29xxx and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

RX8_bReadRxData

Description:

Reads received data byte from the Buffer register.

C Prototype:

```
BYTE  RX8_bReadRxData(void)
```

Assembly:

```
lcall  RX8_bReadRxData
mov   [bRxData],A
```

Parameters:

None

Return Value:

Received data is returned in the Accumulator.

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 (CY8C29xxx and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

RX8_bReadRxStatus

Description:

Reads and returns the status bits of the Control register.

C Prototype:

```
BYTE  RX8_bReadRxStatus(void)
```

Assembly:

```
call  RX8_bReadRxStatus
mov   [bRxStatus],A
```

Parameters:

None

Return Value:

Returns status byte read. Use the defined masks below, to test for specific status conditions. Note that masks can be OR'ed together to check for combined conditions.

RX Status Masks	Value
RX8_RX_ACTIVE	0x10
RX8_RX_COMPLETE	0x08
RX8_RX_PARITY_ERROR	0x80
RX8_RX_OVERRUN_ERROR	0x40
RX8_RX_FRAMING_ERROR	0x20
RX8_RX_ERROR	0xE0

Side Effects:

A read of this register clears all status bits. Care should be taken to check all applicable status conditions before discarding the return value. 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 (CY8C29xxx and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

RX8_cGetChar

Description:

Waits for valid character in RX8 and return its value.

C Prototype:

```
CHAR RX8_cGetChar(void)
```

Assembler:

```
lcall RX8_cGetChar         ; lcall function to print single character to
                           ; serial port.
mov  [CharBuffer],A        ; Store retrieved character in buffer
```

Parameters:

None

Return Value:

Char bData: Character read from RX8 is returned in the accumulator.

Side Effects:

Program flow stays in this function until a character is received or a character was previously received but not read. The RX8 interrupt should be disabled when this function is used. 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 (CY8C29xxx and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

RX8_cReadChar

Description:

Reads RX8 port immediately, if data is not available or an error condition exists, or zero is returned; otherwise, character is read and returned.

C Prototype:

```
CHAR RX8_cReadChar(void)
```

Assembler:

```
lcall RX8_cReadChar          ; lcall function to read a character
cmp  A,0x00                  ; Check for error
jz   ProcessError            ; If error, Process the error condition
mov  [CharBuffer],A          ; Store retrieved character in buffer
```

Parameters:

None

Return Value:

CHAR bData: Character read from RX8 port. ASCII characters from 1 to 255 are valid. A returned zero signifies an error condition or no data available.

Side Effects:

Function only accepts characters from 1 to 255 as valid. A 0x00 (null) character is detected as an error condition. 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 (CY8C29xxx and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

RX8_iReadChar

Description:

Reads RX8 port immediately, returns received character and error condition.

C Prototype:

```
INT RX8_iReadChar(void)
```

Assembler:

```
lcall RX8_iReadChar         ; lcall function to read a character
swap A,X                   ; Swap the contain of A and X, to have error condition in A
cmp  A,0x00                ; Check for error
jnz  ProcessError          ; If error, Process the error condition
swap A,X                   ; Swap the contain of A and X, to have a character in A
mov  [CharBuffer],A        ; Store retrieved character in buffer
```

Parameters:

None

Return Value:

unsigned int iData: MSB contains status and LSB contains UART RX data. If the MSB is non-zero, an error has occurred. The table below shows possible returned error codes in the MSB.

Error Flags	Value	Description
RX8_RX_PARITY_ERROR	0x80	Parity Error
RX8_RX_OVERRUN_ERROR	0x40	Buffer Overrun Error
RX8_RX_FRAMING_ERROR	0x20	Character Framing Error
RX8_RX_NO_ERROR	0x0E	No error
RX8_RX_NO_DATA	0x01	No data available

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 (CY8C29xxx and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

RX8_CmdReset

Description:

Resets command buffer and flags. This allows characters for the next command to be accepted.

C Prototype:

```
void RX8_CmdReset(void)
```

Assembler:

```
lcall RX8_CmdReset          ; lcall function to reset command buffer.
```

Parameters:

None

Return Value:

None

Side Effects:

Resets RX8 character count and clears receive buffer. Any characters remaining in the buffer will be lost. 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 (CY8C29xxx and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

RX8_bCmdCheck

Description:

Checks if command terminator has been received.

C Prototype:

```
BYTE RX8_bCmdCheck(void)
```

Assembler:

```
lcall RX8_bCmdCheck      ; lcall function to get command complete status.
cmp  A,0x00              ; Check if command complete
jnz  ProcessCmd          ; Process command buffer
```

Parameters:

None

Return Value:

A non-zero value will be returned if a command terminator has been received.

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 (CY8C29xxx and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

RX8_bCmdLength

Description:

Returns length of characters in the command buffer. This command will return the current command length, whether a command terminator has been received or not.

C Prototype:

```
BYTE RX8_bCmdLength(void)
```

Assembler:

```
lcall RX8_bCmdLength      ; lcall function to get current command
                          ; Command length is returned in Accumulator
```

Parameters:

None

Return Value:

Length of current string in receive buffer.

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 (CY8C29xxx and CY8CLED16).

When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

RX8_szGetParam

Description:

Returns the next parameter from the receive buffer, which is delimited by the Param_Delimiter set in the Device Editor. After all parameters have been returned, any subsequent calls will return a null pointer (zero).

C Prototype:

```
char * RX8_szGetParam(void)
```

Assembler:

```
lcall RX8_szGetParam       ; lcall function to return pointer to the
                           ; next parameter.
                           ; Pointer is returned in A and X.
```

Parameters:

None

Return Value:

char * strPtr: Pointer to parameter string.

Side Effects:

The receive buffer is modified each time szGetParam is called. Nulls are placed after each parameter. 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 (CY8C29xxx and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, the CUR_PP and IDX_PP page pointer registers are modified.

RX8_szGetRestOfParams

Description:

Returns a pointer to the remainder of the receive buffer string that has not been returned with szGet-Param. If this function is called before szGetParam, a pointer to the entire receive buffer is returned.

C Prototype:

```
char * RX8_szGetRestOfParams(void)
```

Assembler:

```
lcall RX8_szGetRestOfParams    ; lcall function to return pointer to the
                               ; remainder of the receive buffer.
                               ; Pointer is returned in A and X.
```

Parameters:

None

Return Value:

char * strPtr: Pointer to remainder of receive string.

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 (CY8C29xxx and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

RX8_bErrCheck

Description:

Checks command error since the last time bErrCheck was called.

C Prototype:

```
BYTE RX8_bErrCheck(void)
```

Assembler:

```
lcall RX8_bErrCheck             ; lcall function to return error status
cmp  A,0x00                     ; Check for error
jnz  ProcessError               ; If error, Process the error condition
```

Parameters:

None

Return Value:

BYTE bErr: MSB contains status of any error condition that may have occurred since the last time this function was called.

Error Flags	Value	Description
RX8_RX_PARITY_ERROR	0x80	Parity Error
RX8_RX_OVERRUN_ERROR	0x40	Buffer overrun Error
RX8_RX_FRAMING_ERROR	0x20	Character framing Error
RX8_RX_BUF_OVERRUN	0x10	Software RX buffer overrun

Side Effects:

Error status is cleared. 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 (CY8C29xxx and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

Sample Firmware Source Code

The following sample firmware illustrates how to use the API functions to create a routine that waits for the RX8 receiver to receive a byte of data.

```
;
;  This sample shows how to read the resived byte.
;
; OVERVIEW:
;
;  The RX8 input can be routed to any pin.
;  In this example the RX8 input is routed to P0[0].
;
;The following changes need to be made to the default settings in the Device Editor:
;
;   1. Select RX8 user module.
;   2. The User Module will occupy the space in dedicated system resources.
;   3. Rename User Module's instance name to RX8.
;   4. Set RX8's Clock Parameter to VC2.
;   5. Set RX8's Input Parameter to Row_0_Input_0.
;   8. Set RX8's ClockSync Parameter to SyncSysClk.
;   6. Set RX8's RxCmdBuffer Parameter to Disable.
;   7. Set RX8's RX Output Parameter to None.
;   8. Set RX8's Data Clock Out Parameter to None.
;   9. Other RX8's Parameters don't change.
;   10.Click on Row_0_Input_0 and connect Row_0_Input_0 to GlobalInEven_0.
;   11.Select GlobalInEven_0 for P0[0] in the Pinout.
;
; CONFIGURATION DETAILS:
;
;  1. The UM's instance name must be shortened to RX8.
;
; 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
;  ------------------------------------------------------------------------
;  RX8       Name               RX8                 UM's instance name
;            Clock              VC2
;            Input              Row_0_Input_0
;            ClockSync          SyncSysClk
```

```
;             RxCmdBuffer        Disable
;             RxBufferSize  16 Bytes
;             Command Terminator 13
;             Param_Delimiter32
;             IgnoreCharsBelow32
;             RX Output      None
;             Data Clock Out     None
;             InvertInput     Normal
;
;  ------------------------------------------------------------------------

; 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 fWaitToReceiveByte    ; assembly routine label
export _fWaitToReceiveByte   ; C code label

; Rx data storage
export  bRxData               ; assembly routine label
export _bRxData              ; C code label

;;;;;;;;;;;;;;;;;;;;;;;;;;
; data storage
;;;;;;;;;;;;;;;;;;;;;;;;;;
area bss(ram,con,rel)
 bRxData:                    ; Rx data storage area
_bRxData:
    blk   1

area text(rom,con,rel)

;;;;;;;;;;;;;;;;;;;;;;;;;;
; equates
;;;;;;;;;;;;;;;;;;;;;;;;;;
TRUE:   equ   1
FALSE:  equ   0

;;;;;;;;;;;;;;;;;;;;;;;;;;
; Routine Code
;;;;;;;;;;;;;;;;;;;;;;;;;;
 fWaitToReceiveByte::
_fWaitToReceiveByte::
; Wait for byte to be received
.WAIT_FOR_RX_COMPLETE:
    call  RX8_bReadRxStatus
    push  A
    and   A, RX8_RX_COMPLETE
    jnz   .CHECK_RX_ERRORS
    pop   A
    jmp   .WAIT_FOR_RX_COMPLETE

; Data completely received now check for errors
```

```
.CHECK_RX_ERRORS:
    pop   A                       ; Restore status register state
    and   A, RX8_RX_NO_ERROR      ; mask off non-status bits
    jz    .DATA_RX_WITH_NO_ERRORS ; data is valid - no error detected

; Errors detected in received data - return with error condition
;   1) A is set to FALSE indicating error condition
;   2) bRxData contains the RX status flags for further processing
.RX_ERRORS_FOUND:
    mov   [bRxData], A            ; bRxData contains the status flags
    call  RX8_bReadRxData         ; Read RxData reg to prevent future
                                  ; overrun error
    mov   A, FALSE                ; Set A to FALSE condition
    ret

; No error detected in received data - return with data
;   1) A is set to TRUE indicating NO error condition
;   2) bRxData contains the received data byte
.DATA_RX_WITH_NO_ERRORS:
    call  RX8_bReadRxData         ; get the received data in A
    mov   [bRxData], A            ; bRxData contains received
                                  ; data byte
    mov   A, TRUE                 ; set a to NO error condition
    ret
END_fWaitToReceiveByte:

export _main
_main:
    ; M8C_EnableGInt ; Uncomment this line to enable Global Interrupts
    mov    A, RX8_PARITY_NONE
    lcall  RX8_Start ; Enable user module and set parity.
    ; Insert your main assembly code here.

.terminate:
.wait:
    lcall fWaitToReceiveByte ; Wait for byte to be received
    jz .wait
    jmp .terminate
```

Here is a sample project written in C:

```
//
// This sample shows how to read the resived byte.
//  In this example the RX8 input is routed to P0[0].
//
//The following changes need to be made to the default settings in the Device Editor:
//
//   1. Select RX8 user module.
//   2. The User Module will occupy the space in dedicated system resources.
//   3. Rename User Module's instance name to RX8.
//   4. Set RX8's Clock Parameter to VC2.
//   5. Set RX8's Input Parameter to Row_0_Input_0.
//   8. Set RX8's ClockSync Parameter to SyncSysClk.
//   6. Set RX8's RxCmdBuffer Parameter to Disable.
//   7. Set RX8's RX Output Parameter to None.
//   8. Set RX8's Data Clock Out Parameter to None.
```

```
//   9. Other RX8's Parameters don't change.
//   10.Click on Row_0_Input_0 and connect Row_0_Input_0 to GlobalInEven_0
//   11.Select GlobalInEven_0 for P0[0] in the Pinout.
//
// CONFIGURATION DETAILS:
//
//  1. The UM's instance name must be shortened to RX8.
//
// 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
//  ------------------------------------------------------------------------
//  RX8       Name               RX8                  UM's instance name
//            Clock              VC2
//            Input              Row_0_Input_0
//            ClockSync          SyncSysClk
//            RxCmdBuffer        Disable
//            RxBufferSize   16 Bytes
//            Command Terminator 13
//            Param_Delimiter 32
//            IgnoreCharsBelow 32
//            RX Output      None
//            Data Clock Out     None
//            InvertInput      Normal
//
//  ------------------------------------------------------------------------

/* Code begins here */

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

/* Global bRxData - saves code - ptrs are expensive */
BYTE bRxData;

BOOL fWaitToReceiveByte(void)
{
    BYTE bRxStatus;

    /* Wait to receive full byte*/
    while ( !( bRxStatus=RX8_bReadRxStatus() & RX8_RX_COMPLETE ) )
    {
        /* might want to sleep or keep track of time */
    }

    /* data received, now check for errors */
    if (( bRxStatus & RX8_RX_NO_ERROR ) == 0 )
    {
```

```
        /* no error detected */
        bRxData = RX8_bReadRxData();
        return( TRUE );
    }
    else
    {
        /* error detected */
        bRxData = bRxStatus;
        return( FALSE );
    }
}

void main(void)
{
    M8C_EnableGInt; // Enable Global Interrupts
    RX8_EnableInt();// Enable the RX8 interrupt
    RX8_Start(RX8_PARITY_NONE); // Set the parity of the RX8 receiver and enable the
                                //RX8 module

    while(1)
    {
        while(!fWaitToReceiveByte()); // wait to resive byte
        // Insert your main routine code here.
    }
}
```

Configuration Registers

The Digital Communication Type A PSoC block registers used to configure the RX8 User Module are described below. Only the parameterized symbols are explained.

Table 4. Block RX, Register: Function

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

This register defines the personality of this Digital Communications block to be an RX8 User Module.

Table 5. Block RX, Register: Input

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

Input Source selects the RX8 input source. Clock Source selects the clock to drive the receiver timing.

Table 6. Block RX, Register: Output

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

This register is not used.

Table 7. Block RX, Data Shift Register: DR0

Bit	7	6	5	4	3	2	1	0
Value	RX8 Shift Register							

RX8 Shift Register: When a start bit is detected on the input, the RX8 state machine hardware generates a divide-by-8 bit clock that shifts data into this register.

Table 8. Block RX, Data Register: DR1

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

This register is not used.

Table 9. Block RX, Data Buffer Register: DR2

Bit	7	6	5	4	3	2	1	0
Value	RX8 Buffer Register							

RX8 Buffer Register: Data is transferred from the RX8 Shift register after the stop bit has been sampled.

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

Bit	7	6	5	4	3	2	1	0
Value	Parity Error	Overrun Error	Framing Error	Rx Active	Rx Reg Full	Parity Type	Parity Enable	Rx Enable

Parity Error is a flag that indicates parity computation result of received data byte.

Overrun Error is a flag that indicates that the RX Buffer register data is overwritten.

Framing Error is a flag that indicates the stop bit was properly received.

Rx Active is a flag that indicates whether or not a data byte is actively being received.

Rx Reg Full is a flag that indicates a data byte has been completely received, the data byte has been transferred to the Rx Buffer Register, and the error conditions are valid.

Parity Type is a type of parity to compute. This bit is a “don’t care if Parity Enable bit is not set.”

Parity Enable enables or disables the computation of the received parity bit. Parity is selected by setting the Parity Type bit.

Rx Enable enables or disables the RX8 receiver.

Version History

Version	Originator	Description
3.4	DHA	Added compatibility for Large Memory Model chips. 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.
3.50.b	DHA	Updated the datasheet example.

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 © 2002-2015 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.