

CYWA_Simple_UART Community Component Datasheet

Table of Contents

Overview	2
Features.....	2
General Description	2
When to Use a UART.....	2
Disclaimer.....	2
Input/Output Connections.....	3
Component Parameters.....	4
Basic Tab	4
Advanced Tab.....	5
Application Programming Interface	6
Component Debug Window	6
Resources.....	6
References	6
Component Changes.....	6
Module Index.....	7
Data Structure Index.....	7
Module Documentation	8
Functions	8
General APIs.....	8
Power Management APIs	17
Interrupt Service Routine.....	17
API Constants	17
Global Variables	17
Structures.....	18
Data Structure Documentation	19
UART_BACKUP_STRUCT	19
Index.....	19

Overview

The CYWA_Simple_UART is a trimmed down UART component for PSoC 4s that have a serial communication block (SCB).

Features

- User-friendly Instance Customizer with UART-only options
- API focused on essential UART capabilities

General Description

Allows transmission and reception of UART messages.

When to Use a CYWA_Simple_UART

Use this component when you want to set up UART communication.

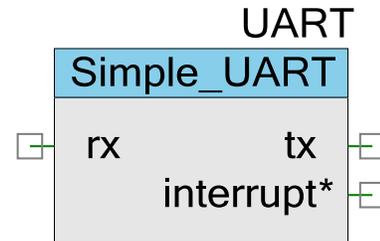


Figure 1. CYWA_Simple_UART_v1_0 instance.

Disclaimer

©Cypress Semiconductor Corporation, 2015. 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® is a registered trademark, and PSoC Creator™ and Programmable System-on-Chip™ are trademarks 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.

Input/Output Connections

This section describes the various input and output connections for the UART component. An asterisk (*) in the list of I/Os indicates that the I/O may be hidden on the symbol under the conditions listed in the description of that I/O.

Terminal Name (optional*)	I/O Type	Description
rx	Digital Input	This is the terminal that receives data. ▲ Note: Any pin connected to the rx terminal cannot be synced. It must be “Transparent”, as shown in Figure 3 below.
tx	Digital Output	This is the terminal that transmits data.
interrupt*	Optional Digital Output	Connect this terminal to an ISR to take advantage of the UART interrupts. This pin is optional. To enable it, open the Instance Customizer (double click the component), go to the 'Advanced' tab, within the 'Interrupt' panel, select the 'External' radio button.

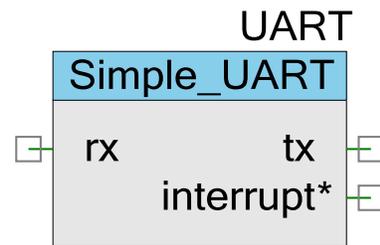


Figure 2. CYWA_Simple_UART_v1_0 instance with optional terminals shown.

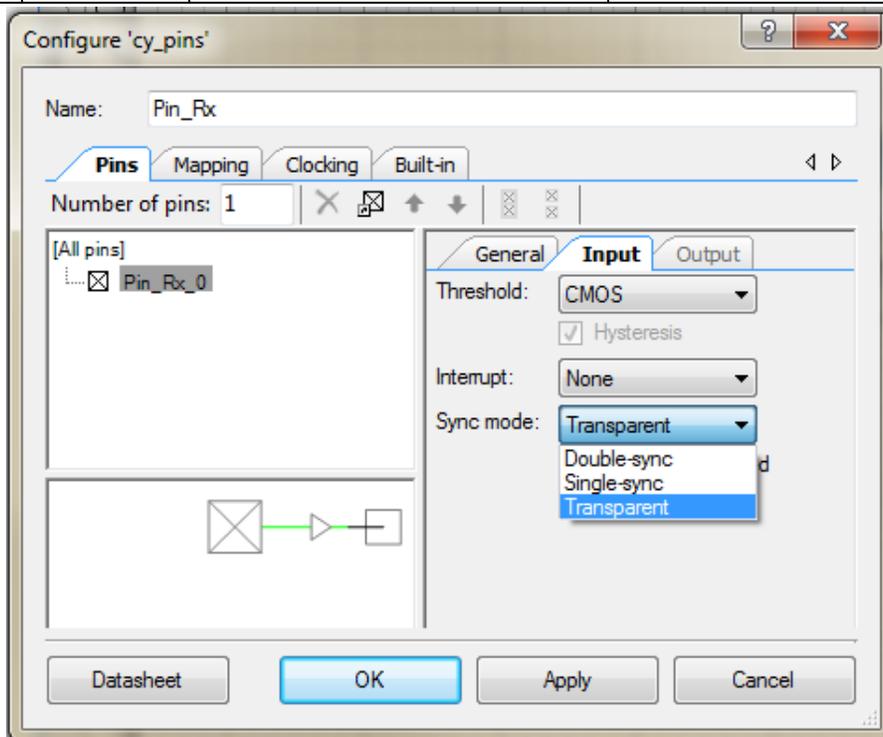


Figure 3. Pin configuration dialog for rx terminal.

Component Parameters

The Built-in tab is covered in the PSoC Creator Help and Component Author Guide, so it is not included here.

Drag a CYWA_Simple_UART component onto your design and double click it to open the Configure dialog. This dialog has the following tabs with different parameters. Default values are given in **bold**.

Basic Tab

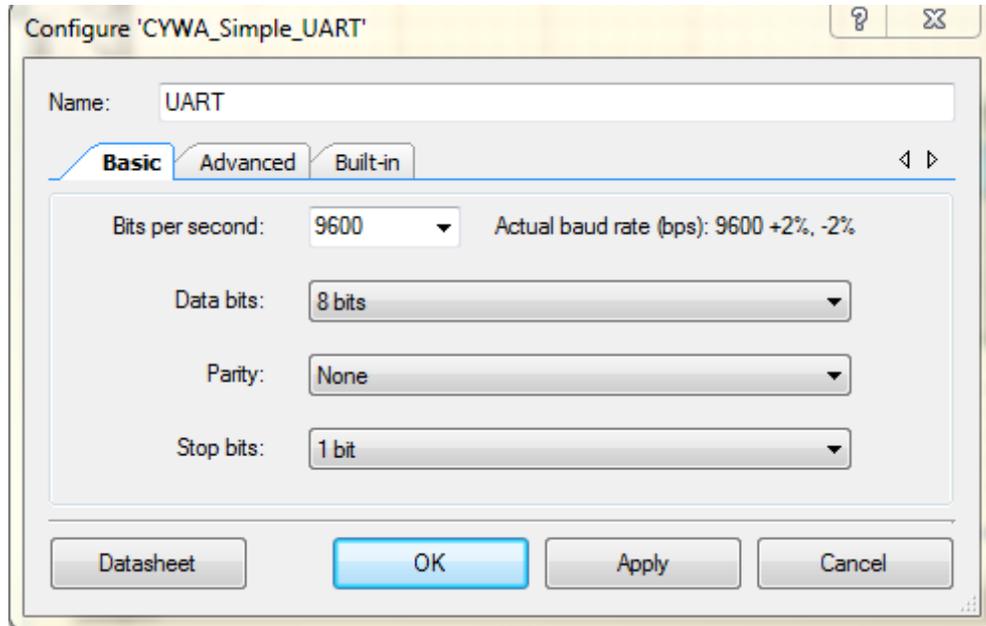


Figure 4. Basic tab configuration.

Control	Description
Bits per second	The desired bitrate. Default value is 9600 . The actual value is calculated based on desired bitrate and oversample factor. (See Advanced Tab for oversample factor)
Data bits	Length of frame payload, in bits. Options are 5, 6, 7, or 8 .
Parity	Used for error detection. Options are Even, Odd, or None .
Stop Bits	Width of terminating symbol, in bit-lengths. Options are 1 bit , 1.5 bits, or 2 bits.

Advanced Tab

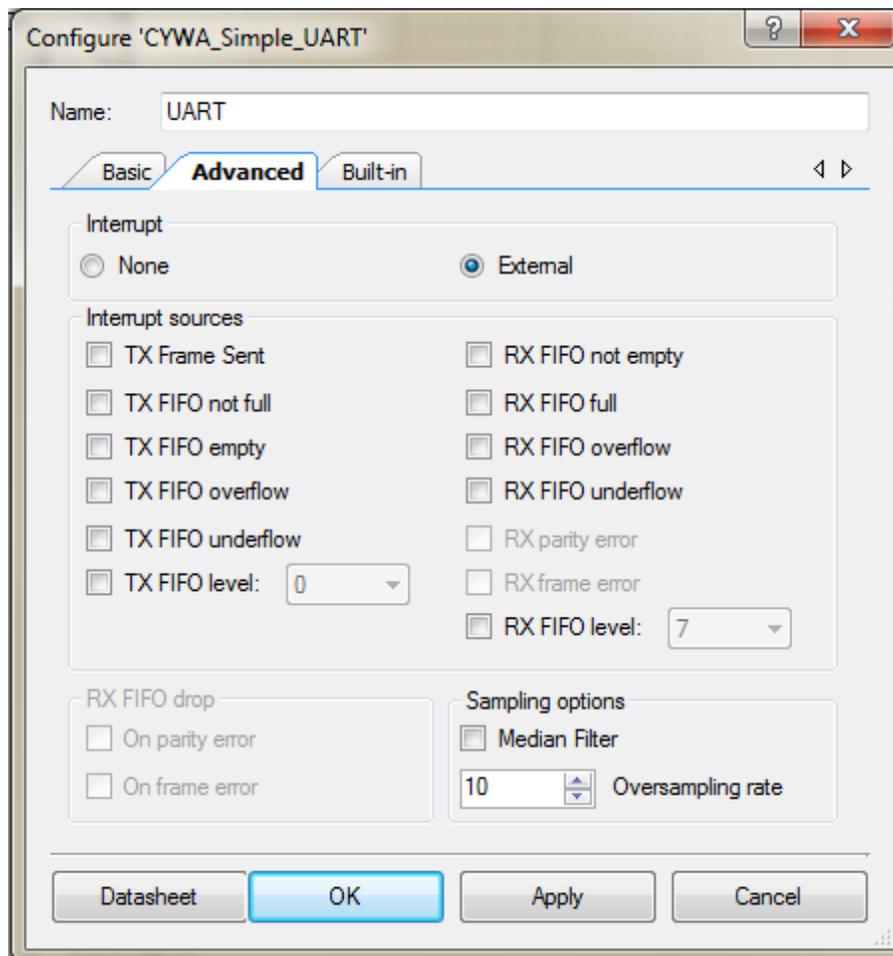


Figure 5. Advanced tab configuration.

Control	Description
Interrupt Panel	Choice of whether to expose interrupt pin and enable interrupts. Options are None and External .
Interrupt sources Panel	A collection of available interrupts. Mouse-over tooltips give more explanation for individual interrupts. RX parity error and RX frame error are allowed when Parity != 'None' and when Stop Bits != '1bits,' respectively.
RX FIFO drop Panel	Hardware will drop the frame if either of two errors are detected: On parity error, or On frame error. This panel and its checkboxes are allowed when Parity != 'None' and when Stop Bits != '1bits.'
Sampling options Panel	On this panel, you can enable the median filter or set the oversampling factor, which can be between 8-16 (10).

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. For the list of APIs used in this component, refer to the [API Reference](#).

Component Debug Window

PSoC Creator allows viewing debug information about the components in the design. Each component window lists the memory and registers for the instance. For detailed hardware registers descriptions, refer to the appropriate device technical reference manual.

To open the Component Debug window:

1. Make sure the debugger is running or in break mode.
2. Choose Windows > Components ... from the Debug menu.
3. In the Component Window Selector dialog, select the component instances to view and click OK.

The selected Component Debug window(s) will open within the debugger framework. Refer to the "Component Debug Window" topic in the PSoC Creator Help for more information.

Resources

The CYWA_Simple_UART component uses the following device resources:

- Serial Communication Block

References

See the PSoC4 UART (SCB Mode) Component for more complete access to the SCB.

Component Changes

This section lists the changes in the component from the previous versions.

Version	Description of Changes	Reason for Changes / Impact
v1_0	Initial version.	SCB component is giant and difficult to start with.

Module Index

Modules

Here is a list of all modules:

Functions	8
General APIs	8
Power Management APIs	17
Interrupt Service Routine	17
API Constants.....	17
Global Variables.....	17
Structures.....	18

Data Structure Index

Data Structures

Here are the data structures with brief descriptions:

<u>UART BACKUP STRUCT</u>	19
---	----

Module Documentation

Functions

API functions allow configuration of the component using the CPU.

Modules

- [General APIs](#)
- *General APIs are used for run-time configuration of the component during active power mode. [Power Management APIs](#)*

Power management APIs perform the necessary configurations to the components to prepare it for entering low power modes.

Detailed Description

API functions allow configuration of the component using the CPU.

By default, PSoC Creator assigns the instance name **UART_1** to the first instance of a component in a given design. You can rename it to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is **UART**.

General APIs

General APIs are used for run-time configuration of the component during active power mode.

Functions

- void [UART_Init](#) (void)
Initializes UART according the customizer configure dialogue settings.
- void [UART_Enable](#) (void)
Enables the UART component.
- void [UART_Start](#) (void)
Initializes and enables the UART Component.
- void [UART_Stop](#) (void)
Stops UART functionality.
- void [UART_SetRxInterruptMask](#) (uint32 intScr)
Configures the RX Interrupt sources. This API reloads entire INTR_RX_MASK register values with the parameter values (configures all interrupts at one call).
- uint32 [UART_ReadRxData](#) (void)
Reads the next byte data from the FIFO.
- uint32 [UART_ReadRxIntStatus](#) (void)
Reads the status of the RX interrupt register.
- void [UART_ClearRxInterrupt](#) (uint32 intClr)
Clears the RX interrupt flag in the INTR_RX_REG.
- uint32 [UART_GetChar](#) (void)
Reads the last received data from RX buffer and interpret as an ASCII data.
- uint32 [UART_GetByte](#) (void)

Reads the last received data from RX FIFO.

- uint32 [UART_GetRxBufferSize](#) (void)
Checks the received byte available in the RX FIFO.
- void [UART_ClearRxBuffer](#) (void)
Clears the RX FIFO from all received data.
- void [UART_SetTxInterruptMask](#) (uint32 intScr)
Configures the TX Interrupt sources. This API reloads entire INTR_TX_MASK register values with the parameter values (configures all interrupts at one call).
- void [UART_WriteTxData](#) (uint32 txData)
Writes data immediately to the TX FIFO.
- uint32 [UART_ReadTxIntStatus](#) (void)
Reads the status register in the TX interrupt register.
- void [UART_ClearTxInterrupt](#) (uint32 intClr)
Clears the TX interrupt flag in the INTR_TX_REG.
- void [UART_PutChar](#) (uint8 txDataByte)
Writes a byte of data to RX buffer.
- void [UART_PutString](#) (const char8 string[])
Writes a string to TX FIFO.
- void [UART_PutArray](#) (const uint8 wrBuf[], uint32 count)
Places N bytes of data from a memory array into the TX FIFO for transmission.
- void [UART_PutCRLF](#) (uint8 txDataByte)
Writes a byte of data followed by a carriage return (0x0D) and line feed (0x0A) to the transmit buffer.
- uint32 [UART_GetTxBufferSize](#) (void)
Determines the number of bytes loaded into the TX FIFO. An empty buffer returns 0.
- void [UART_ClearTxBuffer](#) (void)
Clears all data from the TX FIFO.

Detailed Description

General APIs are used for run-time configuration of the component during active power mode. These include, initializing, starting, stopping, reading from registers and writing to registers.

Function Documentation

void UART_Init (void)

Initializes UART according the customizer configure dialogue settings.

Initialize or restore UART configurations, but does not turn on power to active components. It is not necessary to call [UART_Init\(\)](#) because the [UART_Start\(\)](#) API calls this function, which is the preferred method to begin UART component operation.

See also:

[UART_Start\(\)](#)
[UART_Enable\(\)](#)
[UART_Stop\(\)](#)

None

void UART_Enable (void)

Enables the UART component.

Activates the hardware and begins component operation. It is not necessary to call [UART_Enable\(\)](#) because the [UART_Start\(\)](#) API calls this function, which is the preferred method to begin UART component operation.

See also:

[UART_Start\(\)](#)

[UART_Init\(\)](#)

[UART_Stop\(\)](#)

None

void UART_Start (void)

Initializes and enables the UART Component.

This function is equivalent to calling [UART_Init\(\)](#) followed by [UART_Enable\(\)](#).

Parameters:

<i>weekOfMonth</i>	The week number of the target month. See the weekOfMonth constants for the valid values.
--------------------	--

See also:

[UART_Init\(\)](#)

[UART_Enable\(\)](#)

[UART_Stop\(\)](#)

None

void UART_Stop (void)

Stops UART functionality.

Disable the UART operation. Does not store volatile settings.

See also:

[UART_Start\(\)](#)

[UART_Init\(\)](#)

[UART_Enable\(\)](#)

None

void UART_SetRxInterruptMask (uint32 intSrc)

Configures the RX Interrupt sources. This API reloads entire INTR_RX_MASK register values with the parameter values (configures all interrupts at one call).

Enables the interrupt sources in the UART_INTR_RX_MASK_REG.

- bit 0 : Trigger. Set if RX FIFO has more entries then specified be RX trigger level
- bit 2 : Not Empty. Set if RX FIFO is NOT EMPTY
- bit 3 : Full. Set if TX FIFO is FULL
- bit 5 : Overflow. Set if attempt to write to a full RX FIFO
- bit 6 : Underflow. Set if attempt to read from an empty RX FIFO
- bit 8 : Frame Error. Set if frame error in received data frame
- bit 9 : Parity Error. Set if parity error in received data frame

Bit fields definition for each interrupt in the UART_INTR_RX_MASK_REG.

- bit 0 : UART_INTR_RX_MASK_TRIGGER_MASK
- bit 2 : UART_INTR_RX_MASK_NOT_EMPTY_MASK
- bit 3 : UART_INTR_RX_MASK_FULL_MASK
- bit 5 : UART_INTR_RX_MASK_OVERFLOW_MASK
- bit 6 : UART_INTR_RX_MASK_UNDERFLOW_MASK
- bit 8 : UART_INTR_RX_MASK_FRAME_ERROR_MASK
- bit 9 : UART_INTR_RX_MASK_PARITY_ERROR_MASK

Parameters:

<i>intSrc</i>	Value for selecting RX HW interrupt.
---------------	--------------------------------------

```
1 UART_SetRxInterruptMask(UART_INTR_RX_MASK_TRIGGER_MASK |
2                          UART_INTR_RX_MASK_OVERFLOW_MASK);
```

See also:

- [UART_ClearRxInterrupt\(\)](#)
- [UART_ReadRxIntStatus\(\)](#)

None

uint32 UART_ReadRxData (void)

Reads the next byte data from the FIFO.

Returns the next byte of received data. This function returns data without checking the status.

Returns:

uint32: Received data from RX register.

See also:

- [UART_GetChar\(\)](#)
- [UART_GetByte\(\)](#)

None

uint32 UART_ReadRxIntStatus (void)

Reads the status of the RX interrupt register.

Returns contents of the RX interrupt register (1 = set/event has occurred).

- bit 0 : Trigger. Set when RX FIFO has more entries then specified be RX trigger level
- bit 2 : Not Empty. Set if RX FIFO is NOT EMPTY
- bit 3 : Full. Set if TX FIFO is FULL
- bit 5 : Overflow. Set if attempt to write to a full RX FIFO
- bit 6 : Underflow. Set if attempt to read from an empty RX FIFO
- bit 8 : Frame Error. Set if frame error in received data frame
- bit 9 : Parity Error. Set if parity error in received data frame

Returns:

uint32: Contents of the RX interrupt register.

See also:

- [UART_SetRxInterruptMask\(\)](#)
- [UART_ClearRxInterrupt\(\)](#)

None

void UART_ClearRxInterrupt (uint32 intClr)

Clears the RX interrupt flag in the INTR_RX_REG.

Clears RX interrupt flags by writing "1" to the specific bit field. Cleared interrupt flags will be updated (set "1" or not set "0") with the latest RX HW status.

- bit 0 : Trigger.
- bit 2 : Not Empty.
- bit 3 : Full.
- bit 5 : Overflow.
- bit 6 : Underflow.
- bit 8 : Frame Error.
- bit 9 : Parity Error.

Bit fields definition for each interrupt in the UART_INTR_RX_REG.

- bit 0 : UART_INTR_RX_TRIGGER_MASK
- bit 2 : UART_INTR_RX_NOT_EMPTY_MASK
- bit 3 : UART_INTR_RX_FULL_MASK
- bit 5 : UART_INTR_RX_OVERFLOW_MASK
- bit 6 : UART_INTR_RX_UNDERFLOW_MASK
- bit 8 : UART_INTR_RX_FRAME_ERROR_MASK
- bit 9 : UART_INTR_RX_PARITY_ERROR_MASK

Parameters:

<i>intClr</i>	Value for clearing (and then updating) RX HW interrupt.
---------------	---

Code example for clearing RX TRIGGER and OVERFLOW interrupt flags.

```
1 UART_ClearRxInterrupt (UART_INTR_RX_TRIGGER_MASK |  
2                       UART_INTR_RX_OVERFLOW_MASK) ;
```

See also:

[UART_ReadRxIntStatus\(\)](#)
[UART_SetRxInterruptMask\(\)](#)

None

uint32 UART_GetChar (void)

Reads the last received data from RX buffer and interpret as an ASCII data.

Returns the last received byte of data.

Returns:

uint32: A byte of data, where 1 to 255 are values for valid characters and 0 indicates an error occurred or no data is present.

See also:

[UART_ReadRxData\(\)](#)
[UART_GetByte\(\)](#)

None

uint32 UART_GetByte (void)

Reads the last received data from RX FIFO.

Retrieves the next data element from the RX FIFO, returns the received byte and error condition.

Returns:

uint32: rxData[7-0] contain the next data element from the receive buffer and rxData[8] contains the error condition (0 = New valid data, 1 = Underflow occurs).

See also:

[UART_ReadRxData\(\)](#)

[UART_GetChar\(\)](#)

[UART_GetRxBufferSize\(\)](#)

None

uint32 UART_GetRxBufferSize (void)

Checks the received byte available in the RX FIFO.

Returns the number of received bytes available in the RX buffer.

- RX software buffer disabled: returns the number of used entries in RX FIFO.

Returns:

uint32 Number of received data elements in the RX FIFO.

See also:

[UART_ClearRxBuffer\(\)](#)

None

void UART_ClearRxBuffer (void)

Clears the RX FIFO from all received data.

For clearing the RX FIFO, the CLEAR bit field in the RX_FIFO_CTRL_REG must be driven by logic "1" followed by logic "0" (A short pulse).

See also:

[UART_ReadRxData\(\)](#)

None

void UART_SetTxInterruptMask (uint32 intScr)

Configures the TX Interrupt sources. This API reloads entire INTR_TX_MASK register values with the parameter values (configures all interrupts at one call).

Enables the interrupt sources in the UART_INTR_TX_MASK_REG.

- bit 0 : Trigger. Set when TX FIFO has less entries then specified by TX trigger level
- bit 1 : Not Full. Set if TX FIFO is NOT full
- bit 4 : Empty. Set if TX FIFO is EMPTY
- bit 5 : Overflow. Set if attempt to write to a full TX FIFO
- bit 6 : Underflow. Set if attempt to read from an empty TX FIFO
- bit 9 : UART done. Set if UART Transmitter is done transmitting a data

Bit fields definition for each interrupt in the UART_INTR_TX_MASK_REG.

- bit 0 : UART_INTR_TX_MASK_TRIGGER_MASK
- bit 1 : UART_INTR_TX_MASK_NOT_FULL_MASK
- bit 4 : UART_INTR_TX_MASK_EMPTY_MASK
- bit 5 : UART_INTR_TX_MASK_OVERFLOW_MASK
- bit 6 : UART_INTR_TX_MASK_UNDERFLOW_MASK
- bit 9 : UART_INTR_TX_MASK_UART_DONE_MASK

Parameters:

<i>intScr</i>	Value for selecting TX HW interrupt.
---------------	--------------------------------------

Code example for selecting NOT_FULL, EMPTY and UART_DONE flags as TX interrupt sources.

```

1 UART_SetTxInterruptMask(UART_INTR_TX_MASK_NOT_FULL_MASK |
2                       UART_INTR_TX_MASK_EMPTY_MASK |
3                       UART_INTR_TX_MASK_UART_DONE_MASK);

```

See also:

- [UART_ClearTxInterrupt\(\)](#)
- [UART_ReadTxIntStatus\(\)](#)

None

void UART_WriteTxData (uint32 txData)

Writes data immediately to the TX FIFO.

Places a data entry into the TX FIFO to be sent at the next available bus time. This function is blocking and waits until there is space available to put the requested data in the TX FIFO.

Parameters:

<i>txData</i>	The data to be transmitted.
---------------	-----------------------------

See also:

- [UART_PutChar\(\)](#)
- [UART_PutString\(\)](#)
- [UART_ClearTxBuffer\(\)](#)

None

uint32 UART_ReadTxIntStatus (void)

Reads the status register in the TX interrupt register.

Returns contents of the TX interrupt register (1 = set/event has occurred).

- bit 0 : Trigger. Set when TX FIFO has less entries then specified by TX trigger level
- bit 1 : Not Full. Set if TX FIFO is NOT full
- bit 4 : Empty. Set if TX FIFO is EMPTY
- bit 5 : Overflow. Set if attempt to write to a full TX FIFO
- bit 6 : Underflow. Set if attempt to read from an empty TX FIFO
- bit 9 : UART done. Set if UART Transmitter is done transmitting a data

Returns:

uint32: Content of TX interrupt register.

See also:

- [UART_ClearTxInterrupt\(\)](#)
- [UART_SetTxInterruptMask\(\)](#)

None

void UART_ClearTxInterrupt (uint32 intClr)

Clears the TX interrupt flag in the INTR_TX_REG.

Clears TX interrupt flags by writing "1" to the specific bit field. Cleared interrupt flags will be updated (set "1" or not set "0") with the latest TX HW status.

- bit 0 : Trigger.

- bit 1 : Not Full.
- bit 4 : Empty.
- bit 5 : Overflow.
- bit 6 : Underflow.
- bit 9 : UART done.

Bit fields definition for each interrupt in the UART_INTR_TX_REG.

- bit 0 : UART_INTR_TX_TRIGGER_MASK
- bit 1 : UART_INTR_TX_NOT_FULL_MASK
- bit 4 : UART_INTR_TX_EMPTY_MASK
- bit 5 : UART_INTR_TX_OVERFLOW_MASK
- bit 6 : UART_INTR_TX_UNDERFLOW_MASK
- bit 9 : UART_INTR_TX_UART_DONE_MASK

Parameters:

<i>intClr</i>	Value for clearing (and then updating) TX HW interrupt.
---------------	---

Code example for clearing the TX NOT_FULL and UART_DONE interrupt flags.

```
1 UART_ClearTxInterruptMask(UART_INTR_TX_MASK_NOT_FULL_MASK |
2                           UART_INTR_TX_MASK_UART_DONE_MASK);
```

See also:

- [UART_SetTxInterruptMask\(\)](#)
- [UART_ReadTxIntStatus\(\)](#)

None

void UART_PutChar (uint8 txDataByte)

Writes a byte of data to RX buffer.

Puts a byte of data into the transmit buffer to be sent when the bus is available. This is a blocking API that waits until the TX buffer has room to hold the data.

Parameters:

<i>txDataByte</i>	The data to be transmitted.
-------------------	-----------------------------

See also:

- [UART_PutString\(\)](#)
- [UART_PutArray\(\)](#)
- [UART_PutCRLF\(\)](#)
- [UART_WriteTxData\(\)](#)

None

void UART_PutString (const char8 string[])

Writes a string to TX FIFO.

Places a NULL terminated string in the TX FIFO to be sent at the next available bus time. This function is blocking and waits until there is space available to put all the requested data into the TX FIFO.

Parameters:

<i>string[]</i>	: char8 Pointer to the null terminated string array to be placed in the TX FIFO.
-----------------	--

See also:

- [UART_PutChar\(\)](#)
- [UART_PutArray\(\)](#)

[UART_PutCRLF\(\)](#)
[UART_WriteTxData\(\)](#)

None

void UART_PutArray (const uint8 *wrBuf*[], uint32 *count*)

Places N bytes of data from a memory array into the TX FIFO for transmission.

This function is blocking and waits until there is a space available to put all the requested data in the TX FIFO. The array size can be greater than TX FIFO.

Parameters:

<i>wrBuf</i>	Pointer to an array with data to be placed in TX FIFO.
<i>count</i>	Number of data elements to be placed in the TX FIFO.

See also:

[UART_PutChar\(\)](#)
[UART_PutString\(\)](#)
[UART_PutCRLF\(\)](#)
[UART_WriteTxData\(\)](#)

None

void UART_PutCRLF (uint8 *txDataByte*)

Writes a byte of data followed by a carriage return (0x0D) and line feed (0x0A) to the transmit buffer.

This function is blocking and waits until there is space available to put all the requested data into the transmit buffer

Parameters:

<i>txDataByte</i>	The data to be transmitted.
-------------------	-----------------------------

See also:

[UART_PutChar\(\)](#)
[UART_PutString\(\)](#)
[UART_PutArray\(\)](#)
[UART_WriteTxData\(\)](#)

None

uint32 UART_GetTxBufferSize (void)

Determines the number of bytes loaded into the TX FIFO. An empty buffer returns 0.

Returns the number of elements currently in the TX buffer.

- TX software buffer is disabled: returns the number of used entries in TX FIFO.

Returns:

uint32: Number of bytes used in the TX FIFO

See also:

[UART_ClearTxBuffer\(\)](#)

None

void UART_ClearTxBuffer (void)

Clears all data from the TX FIFO.

For clearing the TX FIFO, the CLEAR bit field in the TX_FIFO_CTRL_REG must be driven by logic "1" followed by logic "0" (A short pulse).

See also:

[UART_WriteTxData\(\)](#)

None

Power Management APIs

Power management APIs perform the necessary configurations to the components to prepare it for entering low power modes.

Functions

- void **UART_Sleep** (void)
- void **UART_Wakeup** (void)

Detailed Description

Power management APIs perform the necessary configurations to the components to prepare it for entering low power modes.

These APIs must be used if the intent is to put the chip to sleep, then to continue the component operation when it comes back to active power mode.

Interrupt Service Routine

Include this section if relevant; otherwise, delete.

Include this section if relevant; otherwise, delete.

Provide any information related to the ISR(s) used in this component. Describe available sources of interrupts and ISR(s) which handle these sources. Specify requirements for the interrupt request priority if any exist.

API Constants

Component APIs are designed to work with pre-defined enumeration values.

Component APIs are designed to work with pre-defined enumeration values.

These values should be used with the APIs that reference them.

Global Variables

Global variables used in the component.

Variables

- uint32 `UART_InitVar`
-

Detailed Description

Global variables used in the component.

The following global variables are used in the component.

Structures

Data Structures are used to group related elements.

Data Structures

- struct [UART_BACKUP_STRUCT](#)
-

Detailed Description

Data Structures are used to group related elements.

The following data structures are used by the component APIs.

Data Structure Documentation

UART_BACKUP_STRUCT Struct Reference

Data Fields

- uint8 enableState

Detailed Description

Structure to save state before go to sleep

Index

API Constants, 17

Functions, 8

General APIs, 8

- UART_ClearRxBuffer, 13
- UART_ClearRxInterrupt, 12
- UART_ClearTxBuffer, 16
- UART_ClearTxInterrupt, 14
- UART_Enable, 10
- UART_GetByte, 12
- UART_GetChar, 12
- UART_GetRxBufferSize, 13
- UART_GetTxBufferSize, 16
- UART_Init, 9
- UART_PutArray, 16
- UART_PutChar, 15
- UART_PutCRLF, 16
- UART_PutString, 15
- UART_ReadRxData, 11
- UART_ReadRxIntStatus, 11
- UART_ReadTxIntStatus, 14
- UART_SetRxInterruptMask, 10
- UART_SetTxInterruptMask, 13
- UART_Start, 10
- UART_Stop, 10
- UART_WriteTxData, 14

Global Variables, 17

Interrupt Service Routine, 17

Power Management APIs, 17

UART_BACKUP_STRUCT, 19

UART_ClearRxBuffer

General APIs, 13

UART_ClearRxInterrupt

General APIs, 12

UART_ClearTxBuffer

General APIs, 16

UART_ClearTxInterrupt

General APIs, 14

UART_Enable

General APIs, 10

UART_GetByte

General APIs, 12

UART_GetChar

General APIs, 12

UART_GetRxBufferSize

General APIs, 13

UART_GetTxBufferSize

General APIs, 16

UART_Init

General APIs, 9

UART_PutArray

General APIs, 16

UART_PutChar

General APIs, 15

UART_PutCRLF

General APIs, 16

UART_PutString

General APIs, 15

UART_ReadRxData

General APIs, 11

UART_ReadRxIntStatus

General APIs, 11

UART_ReadTxIntStatus

General APIs, 14

UART_SetRxInterruptMask

General APIs, 10

UART_SetTxInterruptMask

General APIs, 13

UART_Start

General APIs, 10

UART_Stop

General APIs, 10

UART_WriteTxData

General APIs, 14

Structures, 18