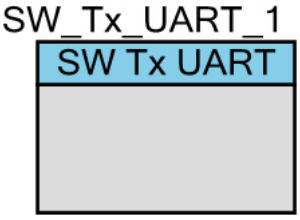
# Software Transmit UART

1.10




## Features

- Baud rates from 9,600 up to 115,200 bps
- High baud rate accuracy
- Low Flash/ROM resource usage

## General Description

The Software Transmit UART (SW_Tx_UART) component is an 8-bit RS-232 data-format compliant serial transmitter.

### When to Use a SW_Tx_UART

The SW_Tx_UART component is used to send out serial data in the RS-232 data-format. The component consists solely of firmware and a pin, so it is useful on devices without digital resources, or in projects where all digital resources are consumed.

The SW_Tx_UART supports PSoC 3, PSoC 4 and PSoC 5LP.

## Input/Output Connections

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

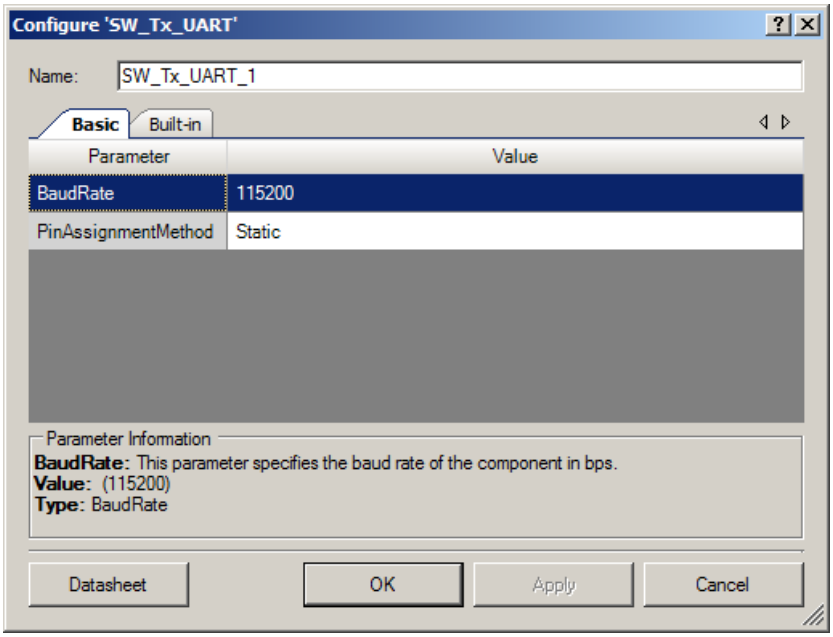
### Tx – Output*

The Tx output is the serial transmit output of the UART. When configured in static mode this pin is present within the component and assigned to a physical pin using the Pin editor in the Design

Wide Resources. When configured in dynamic mode this pin is not present in the component. Instead the pin used is chosen at run-time using the StartEx() function.

# Component Parameters

Drag a SW_Tx_UART onto your design and double click it to open the Configure dialog.




The SW_Tx_UART provides the following parameters.

## Basic Options

### BaudRate

This parameter specifies the baud rate of the component in bps.

Range: 115200, 57600, 38400, 19200, 9600. Default: 115200.

### PinAssignmentMethod

This parameter specifies the method by which the component's output pin is assigned. Static indicates that the component will contain a buried pin that will be assigned in the cydwr file. Dynamic requires that the pin be specified via the StartEx() API.

Range: Static, Dynamic. Default: Static.

# Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. The following table lists and describes the interface to each function. The subsequent sections cover each function in more detail.

By default, PSoC Creator assigns the instance name "SW_Tx_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 "SW_Tx_UART."

## Functions

| Function | Description |
|---|---|
| SW_Tx_UART_Start() | Empty function, included for consistency with other components. |
| SW_Tx_UART_StartEx() | Configures the SW_Tx_UART to use the pin specified by the parameters. |
| SW_Tx_UART_Stop() | Empty function, included for consistency with other components. |
| SW_Tx_UART_PutChar() | Sends one byte via the Tx pin. |
| SW_Tx_UART_PutString() | Sends a NULL terminated string via the Tx pin. |
| SW_Tx_UART_PutArray() | Sends byteCount bytes from a memory array via the Tx pin. |
| SW_Tx_UART_PutHexByte() | Sends a byte in Hex representation (two characters, uppercase for A-F) via the Tx pin. |
| SW_Tx_UART_PutHexInt() | Sends a 16-bit unsigned integer in Hex representation (four characters, uppercase for A-F) via the Tx pin. |
| SW_Tx_UART_PutCRLF() | Sends a carriage return (0x0D) and a line feed (0x0A) via the Tx pin. |

## void SW_Tx_UART_Start(void)

**Description:** Empty function. Included for consistency with other components. This API is not available when PinAssignmentMethod is set to Dynamic.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

## void SW_Tx_UART_StartEx(uint8 port, uint8 pin)

**Description:** Configures the SW_Tx_UART to use the pin specified by the parameters. This API is only available when PinAssignmentMethod is set to Dynamic.

**Parameters:** port: Port number for dynamic pin assignment

pin: Pin number for dynamic pin assignment

**Return Value:** None

**Side Effects:** None

## void SW_Tx_UART_Stop(void)

**Description:** Empty function. Included for consistency with other components.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

## void SW_Tx_UART_PutChar(uint8 txDataByte)

**Description:** Sends one byte via the Tx pin.

**Parameters:** txDataByte: Byte to send

**Return Value:** None

**Side Effects:** None

## void SW_Tx_UART_PutString(const char8 string[])

**Description:** Sends a NULL terminated string via the Tx pin.

**Parameters:** string: Pointer to the null terminated string to send

**Return Value:** None

**Side Effects:** None

## void SW_Tx_UART_PutArray(const uint8 data[], uint16/uint32 byteCount)

**Description:** Sends byteCount bytes from a memory array via the Tx pin.

**Parameters:** data: Pointer to the memory array

byteCount: Number of bytes to be transmitted (uint16 on PSoC 3, uint32 on PSoC 4 and PSoC 5LP)

**Return Value:** None

**Side Effects:** None

## void SW_Tx_UART_PutHexByte(uint8 txHexByte)

**Description:** Sends a byte in Hex representation (two characters, uppercase for A-F) via the Tx pin.

**Parameters:** txHexByte: The byte to be converted to ASCII characters and sent via the Tx pin.

**Return Value:** None

**Side Effects:** None

## void SW_Tx_UART_PutHexInt(uint16 txHexInt)

**Description:** Sends a 16-bit unsigned integer in Hex representation (four characters, uppercase for A-F) via the Tx pin.

**Parameters:** txHexInt: The uint16 to be converted to ASCII characters and sent via the Tx pin.

**Return Value:** None

**Side Effects:** None

## void SW_Tx_UART_PutCRLF()

**Description:** Sends a carriage return (0x0D) and a line feed (0x0A) via the Tx pin.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

# MISRA Compliance

This section describes the MISRA-C:2004 compliance and deviations for the component. There are two types of deviations defined: project deviations – deviations that are applicable for all PSoC Creator components and specific deviations – deviations that are applicable only for this component. This section provides information on component specific deviations. The project deviations are described in the MISRA Compliance section of the *System Reference Guide* along with information on the MISRA compliance verification environment.

The Software transmit UART component does not have any specific deviations.

# Sample Firmware Source Code

PSoC Creator provides numerous example projects that include schematics and example code in the Find Example Project dialog. For component-specific examples, open the dialog from the Component Catalog or an instance of the component in a schematic. For general examples, open the dialog from the Start Page or **File** menu. As needed, use the **Filter Options** in the dialog to narrow the list of projects available to select.

Refer to the "Find Example Project" topic in the PSoC Creator Help for more information.

# Functional Description

The SW_Tx_UART supports 9600, 19200, 38400, 57600, and 115200 bps transfer speeds with no parity bit, 8 data bits, and a single stop bit.

The component consists of a pin and firmware.

If the PinAssignmentMethod is “Static,” the pin is buried in the component implementation, and is assigned in the PSoC Creator Pin Editor. If the PinAssignmentMethod is “Dynamic,” the pin is assigned and configured using the SW_Tx_UART_StartEx() API. In this configuration, the SW_Tx_UART_StartEx() API must be called before calling any other APIs; otherwise, the component will not function as expected.

The supported baud rates are achieved using a combination of pin writes and software delays, implemented with the CyDelay function. If the CPU clock is changed from the configured frequency, then the CyDelayFreq API must be called before any data transmission.

During transmission, the component masks interrupts to ensure proper timing of signals. Interrupts are restored after a byte is sent; functions that send multiple bytes restore interrupts between each byte. All data transmission functions are blocking.

The component relies on the CPU clock frequency accuracy. For the best baud rate accuracy, the clock source for the CPU clock should be configured to use as accurate a clock source as available.

Software Transmit UART component requires to have Instruction Cache to be enabled for PSoC 3 and PSoC 5LP. This could be done in PSoC Creator project Design-Wide Resources editor on the **System** tab. By default, this option is enabled.

## SW_Tx_UART AC Specifications

| Parameter | Description | Baud Rate | Min | Typ | Max | Units |
|---|---|---|---|---|---|---|
| $f_{CLOCK}$ | Operating frequency | 9600 | 3 | - | 67 | MHz |
| | | 19200 | 3 | - | 67 | MHz |
| | | 38400 | 6 | - | 67 | MHz |
| | | 57600 | 12 | - | 67 | MHz |
| | | 115200 | 24 | - | 67 | MHz |

## How to Calculate the Baud Rate Error Value

In the PSoC Creator Clock Editor, the accuracy of each clock source is shown. For example:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| System | XTAL | DIGITAL | 25.000 MHz | ? MHz | ±0 | - | 0 | ☐ | |
| System | XTAL 32kHz | DIGITAL | 32.768 kHz | ? MHz | ±0 | - | 0 | ☐ | |
| System | Digital Signal | DIGITAL | ? MHz | ? MHz | ±0 | - | 0 | ☐ | |
| System | USB_CLK | DIGITAL | 48.000 MHz | ? MHz | ±0 | - | 1 | ☐ | IMOx2 |
| System | ILO | DIGITAL | ? MHz | 1.000 kHz | -50, +100 | - | 0 | ☑ | |
| System | IMO | DIGITAL | 3.000 MHz | 3.000 MHz | ±1 | - | 0 | ☑ | |
| System | PLL_OUT | DIGITAL | 24.000 MHz | 24.000 MHz | ±1 | - | 0 | ☑ | IMO |
| System | MASTER_CLK | DIGITAL | ? MHz | 24.000 MHz | ±1 | - | 1 | ☑ | PLL_OUT |
| System | BUS_CLK (CPU) | DIGITAL | ? MHz | 24.000 MHz | ±1 | - | 1 | ☑ | MASTER_CLK |

In this example, the BUS_CLK (CPU) is derived from the IMO, which has an accuracy of +/-1%.

The error in the baud error rate can be calculated as following:

*Divider = (int) (CPU_CLK+(BaudRate/2))/BaudRate*

$\%_{err}$ *= (BaudRate - CPU_CLK/ Divider)*100% + CPU_CLK_Accuracy*

The first part of the error is with dividing the CPU_CLK down to the baud rate. The second part of the error is the inaccuracy of the CPU_CLK.

# API Memory Usage

The component memory usage varies significantly, depending on the compiler, device, number of APIs used and component configuration. The following table provides the memory usage for all APIs available in the given component configuration.

The measurements have been done with associated compiler configured in Release mode with optimization set for Size. For a specific design the map file generated by the compiler can be analyzed to determine the memory usage.

| Configuration | PSoC 3 (Keil_PK51) | | PSoC 4 (GCC) | | PSoC 5LP (GCC) | |
|---|---|---|---|---|---|---|
| | Flash Bytes | SRAM Bytes | Flash Bytes | SRAM Bytes | Flash Bytes | SRAM Bytes |
| Software Transmit UART | 371 | 8 | 406 | 8 | 378 | 8 |

# Component Changes

This section lists the major changes in the component from the previous version.

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| 1.10.a | Minor datasheet edits. | |
| 1.10 | Added assembly file for IAR compiler. | To support project export to IAR IDE. |
| 1.0 | First component version. | First component version. |

© Cypress Semiconductor Corporation, 2013-2016. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC (“Cypress”). This document, including any software or firmware included or referenced in this document (“Software”), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress’s patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage (“Unintended Uses”). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.