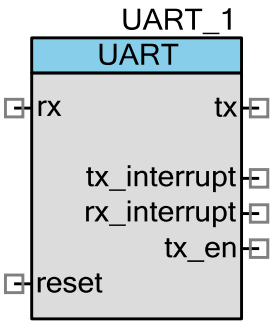
Universal Asynchronous Receiver Transmitter (UART)

1.20

Features



- 9-bit address mode with hardware address detection
- BAUD rates from 110 – 921600bps or arbitrary up to 4Mbps
- Rx and Tx buffers = 1-255 (8-bit), 1-65535 (32-bit)
- Detection of Framing, Parity and Overrun errors
- Full Duplex, Half Duplex, Tx only and Rx only optimized hardware
- 2 out of 3 voting per bit
- Break signal generation and detection

General Description

The UART provides asynchronous communications commonly referred to as RS-232 or RS-485. The UART component can be configured for Full Duplex, Half Duplex, RX only or TX only versions. All versions provide the same basic functionality differing only in the amount of resources utilized.

To assist with processing of the UART receive and transmit data, independent size configurable buffers are provided. The independent circular receive and transit buffers in SRAM as well as hardware FIFOs help to ensure that data will not be missed while allowing the CPU to spend more time on critical real time tasks rather than servicing the UART.

For most use cases the UART can be easily configured by choosing the BAUD rate, parity, number of data bits and number of start bits. The most common configuration for RS-232 is often listed as "8N1" which is shorthand for 8 data bits, No parity and 1 stop bit which is also the default for the UART component. Therefore in most applications only the BAUD rate must be set. A second common use for UARTs is in multi-drop RS-485 networks. The UART component supports 9-bit addressing mode with hardware address detect as well as a Tx output enable signal to enable the Tx transceiver during transmissions.

The long history of UARTs has resulted in many physical layer and protocol layer variations over time including but not limited to RS-423, DMX512, MIDI, LIN bus, legacy terminal protocols and IrDa. To support the UART variations commonly used, the component provides configuration support for the number of data bits, stop bits, parity, hardware flow control and parity generation and detection.

As a hardware compiled option the user can select to output a clock and serial data stream that outputs only the UART data bits on the clock's rising edge. An independent clock and data

output is provided for both the Tx and Rx. The purpose of these outputs is to allow automatic calculation of the data CRC by connecting a CRC component to the UART.

When to use a UART

The UART should be used any time that a compatible asynchronous communications interface is required especially RS-232 and RS-485 and other variations. The UART can also be used to create more advanced asynchronous based protocols such as DMX512, LIN and IrDa or customer/industry proprietary.

A UART should not be used in those cases where a specific component has already been created to address the protocol. For example if a DMX512, LIN or IrDa component were provided, it would have a specific implementation providing both hardware and protocol layer functionality and the UART should not be used in this case (subject to component availability).

Input/Output Connections

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

clock – Input *

The clock input defines the baud rate (bit-rate) of the serial communication. The baud-rate is 1/8th the input clock frequency. This input is visible if the "ClockInternal" parameter is set to false. If the internal clock is selected then you define the desired baud-rate during configuration and the necessary clock frequency is solved by PSoC Creator.

reset – Input

Resets the UART state machines (RX and TX) to the idle state. This will throw out any data that was currently being transmitted or received but will not clear data from the FIFO that has already been received or is ready to be transmitted.

rx – Input *

The rx input carries the input serial data from another device on the serial bus. This input is visible and must be connected if the RX portion of the UART is used. The RX portion of the UART is used when the RX Enable parameter is set by the "RX Only" or "Full UART (RX & TX)" implementations.

cts_n – Input *

The CTS input indicates that another device is ready to receive data. This input is an active-low input indicated by the _n, and indicates when the other device has room for more data to be transmitted to it. This input is visible if the FlowControl parameter is set to "Hardware".

tx – Output *

The tx output carries the output serial data to another device on the serial bus. This output is visible if the TX portion of the UART is used. The TX portion of the UART is used when the TX Enable parameter is set by the "TX Only" or "Full UART (RX & TX)" implementations.

rts_n – Output *

The RTS output indicates to another device that you are ready to receive data. This output is active-low indicated by the _n, and informs another device when you have room for more data to be received. This output is visible if the FlowControl parameter is set to "Hardware".

tx_en – Output *

The tx_en output is used primarily for RS-485 communication to indicate that you are transmitting on the bus. This output will go high before a transmit starts and low when transmit is complete indicating a busy bus to the rest of the devices on the bus. This output is visible when the Hardware TX Enable parameter is true.

tx_interrupt– Output *

The interrupt output is the logical OR of the group of possible interrupt sources. This signal will go high while any of the enabled interrupt sources are true.

rx_interrupt– Output *

The interrupt output is the logical OR of the group of possible interrupt sources. This signal will go high while any of the enabled interrupt sources are true.

tx_data – Output *

The tx_data output used to shift out the TX data to a CRC component or other Logic. This output is visible when the CRC outputs Enable parameter is true.

tx_clk – Output *

The tx_clk output provides clock edge used to shift out the TX data to a CRC component or other logic. This output is visible when the CRC outputs Enable parameter is true.

rx_data – Output *

The tx_data output used to shift out the RX data to a CRC component or other Logic. This output is visible when the CRC outputs Enable parameter is true.

rx_clk – Output *

The rx_clk output provides clock edge used to shift out the RX data to a CRC component or other logic. This output is visible when the CRC outputs Enable parameter is true.

Parameters and Setup

Drag an UART component onto your design and double-click it to open the Configure dialog.

Hardware vs. Software Options

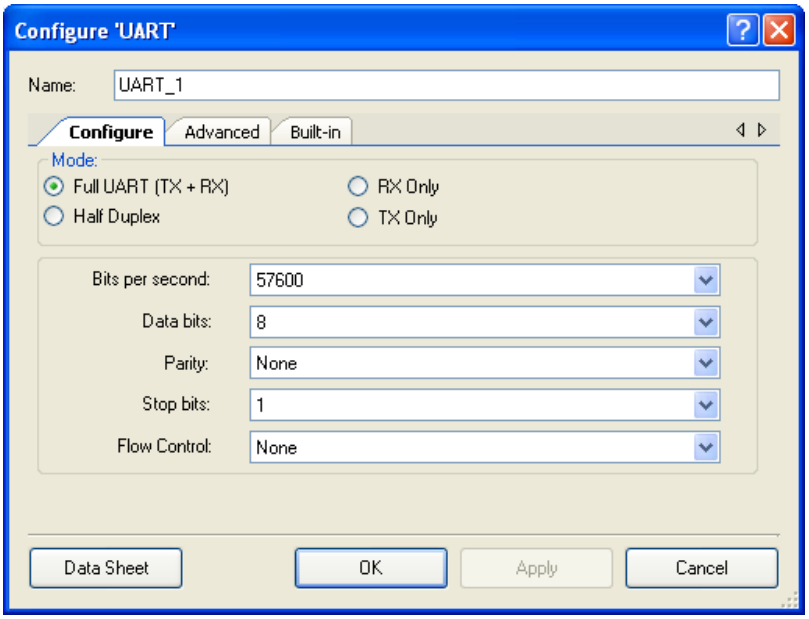
Hardware configuration options change the way the project is synthesized and placed in the hardware. You must rebuild the hardware if you make changes to any of these options. Software configuration options do not affect synthesis or placement. When setting these parameters before build time you are setting their initial value which may be modified at any time with the API provided.

The following sections describe the UART parameters, and how they are configured using the dialog. They also indicate whether the options are hardware or software.

Configure Tab

The dialog is set up to look like a hyper terminal configuration window to avoid incorrect configuration of two sides of the bus, where the PC using hyper-terminal is quite often the other side of the bus.

All of these options are hardware configuration options.



Mode

The Mode parameter defines the desired functional components to include in the UART. This can be setup to be an bi-directional UART (TX + RX) (default), Half Duplex UART(uses half the resources), RS-232 Receiver (RX Only) or Transmitter (TX Only).

Bits Per Second (enum)

The Bits per second parameter defines the baud-rate or bit width configuration of the hardware for clock generation. The default is 57600.

If the internal clock is used (set by the "InternalClock" parameter) the necessary clock to achieve this baud-rate will be generated.

Data Bits (5, 6, 7, 8, 9)

The Data Bits parameter defines the number of data bits transmitted between start and stop of a single UART transaction. 8 data bits is the default configuration sending a byte per transfer. 9-bit mode does not transmit 9 data bits, the 9th bit takes the place of the parity bit as an indicator or Address using Mark/Space parity.

Parity (enum)

The parity parameter defines the functionality of the parity bit location in the transfer. This can be set to None (default), Odd, Even or Mark/Space or Software.

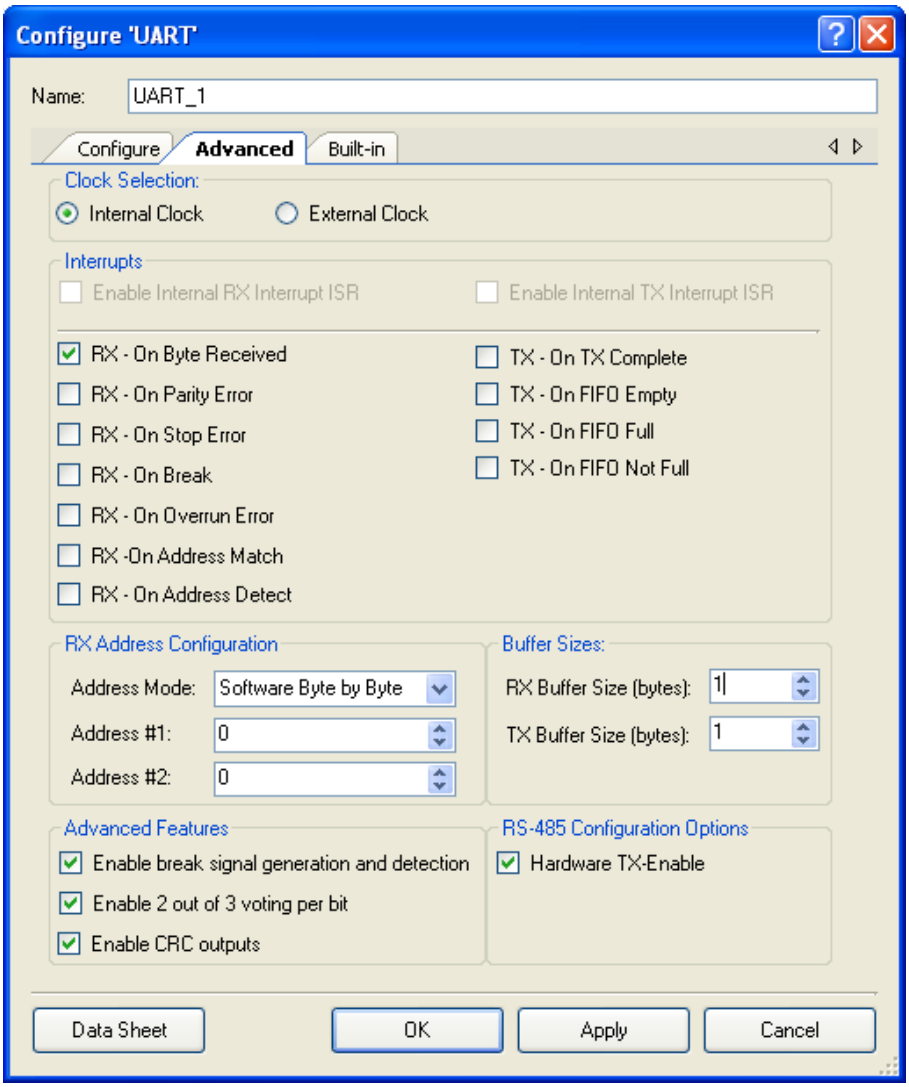
Stop Bits (1, 2)

The Stop Bits parameter defines the number of stop bits implemented in the transmitter and expected on receiving data. This parameter can be set to 1 (default) or 2 data bits.

Flow Control (enum)

The Flow Control parameter allows you to choose between Hardware or None (default). When this parameter is set to Hardware, the CTS and RTS signals become available on the symbol.

Advanced Tab



Advanced Tab – Hardware Configuration Options

Clock Selection (Internal, External)

The Clock Selection parameter allows the user to choose between an internally configured clock or an externally configured clock or I/O for the baud-rate generation. When set to the "Internal" setting the required clock frequency is calculated and configured by PSoC Creator. In the "External" mode the component does not control the baud-rate but can calculate the expected baud-rate.

If this parameter is "Internal" then the clock input is not visible on the symbol.

Interrupts

- **Enable Internal TX Interrupt ISR (bool) –** Enables the ISR supplied by the component for the TX portion of the UART. This parameter is not available if only the RX section of the UART is chosen. If this parameter is not true, you will be responsible for handling the ISR development. The interrupt output is an OR'd grouping of the TX status register bits defined by the mask register. This parameter is true when the TX buffer size is greater than 4 and false when TX buffer is 4 or less.
- **Enable Internal RX Interrupt ISR (bool) –** Enables the ISR supplied by the component for the RX portion of the UART. This parameter is not available if only the TX section of the UART is chosen. If this parameter is not true, you will be responsible for handling the ISR development. The interrupt output is an OR'd grouping of the RX status register bits defined by the mask register. This parameter is true when the RX buffer size is greater than 4 and false when RX buffer size is 4 or less.

RX Address Configuration

- **Address Mode (enum) –** The Address mode parameter defines how hardware and software interact to handle device addresses and data bytes. This parameter can be set to the following types:
  - Software Byte-by-Byte: Hardware indicates the detection of an address byte for every byte received. Software must read the byte and determine if this address matches the device addresses defined as in the Address #1 or Address #2 parameters
  - Software Detect to Buffer: Hardware indicates the detection of an address byte and software will copy all data into the RX buffer defined by the RX Buffer Size parameter.
  - Hardware Byte-By-Byte: Hardware detects a byte and forces an interrupt to move all data from the hardware FIFO into the data buffer defined by RX Buffer Size.
  - Hardware Detect to buffer: Hardware detects a byte and forces an interrupt to move only the data (address byte is not included) from the hardware FIFO into the data buffer defined by RX Buffer Size.
  - None: No RX address detection is implemented.

Advanced Tab – Software Configuration Options

Interrupts

The "Interrupt On" parameters allow you configure the initial interrupt sources. These values are OR'd with any of the other "Interrupt On" parameter to give a final group of events that can trigger an interrupt. The software can re-configure these modes at any time; these parameters simply define an initial configuration.

- **RX – On Byte Received (bool)**
- **RX – On Parity Error (bool)**
- **RX – On Stop Error (bool)**
- **RX – On Break (bool)**
- **RX – On Overrun Error (bool)**
- **RX – On Address Match (bool)**
- **RX – On Address Detect (bool)**
- **TX – On TX Complete (bool)**
- **TX – On FIFO Empty (bool)**
- **TX – On FIFO Full (bool)**
- **TX – On FIFO Not Full (bool)**

RX Address Configuration

- **RX Address #1/#2 (uint8) –** The RX Address parameters indicate up to two device addresses that the UART may assume. These parameters are stored in hardware for hardware address detection modes described in the RX Address Mode parameter and are available to firmware the software address modes.

RS-485 Configuration Options

- **Hardware TX Enable (bool) –** The Hardware TX enable parameter enables or disables the use of the TX-Enable output of the TX UART. This signal is used in RS-485 communications. The hardware provides the functionality of this output automatically, based on buffer conditions.

Buffer Sizes

- **RX Buffer Size (uint8) –** The RX Buffer Size parameter defines how many bytes of RAM to allocate for an RX buffer. Data is moved from the receive registers into this buffer. Buffer sizes greater than 4 bytes require the use of interrupts to handle moving of the data from the receive FIFO into this buffer and GetChar() or ReadRXData() API get data from the correct source without any changes to your top level firmware.
- **TX Buffer Size (uint8) –** The TX Buffer Size parameter defines how many bytes of RAM to allocate for an TX buffer. Data is written into this buffer with the PutChar() and PutArray() API commands. Buffer sizes greater than 4 bytes require the use of interrupts to handle moving of the data from the transmit buffer into the hardware FIFO without any changes to your top level firmware.

Advanced Features

- **Enable break signal generation and detection –** The Enable break signal parameter enables or disables Break signal generation and detection. Send the break signal consisting of a start bit and 13 logic 0s (spec is the standard in use for LIN bus) followed by a stop bit. This option will save resources when disabled.
- **Enable 2 out of 3 voting per bit –** The Enable 2 out of 3 voting per bit enables or disables error compensation algorithm. This option will save resources when disabled.
- **Enable CRC outputs –** The Enable CRC outputs parameter enables or disables tx_data, tx_clk, rx_data, rx_clk outputs. They used to output a clock and serial data stream that outputs only the UART data bits on the clock's rising edge. The purpose of these outputs is to allow automatic calculation of the data CRC. This option will save resources when disabled.

Clock Selection

When the internal clock configuration is selected PSoC Creator will calculate the needed frequency and clock source and will generate the resource needed for implementation. Otherwise, you must supply the clock and calculate the baud-rate at 1/8th the input clock frequency.

Placement

The UART component is placed throughout the UDB array and all placement information is provided to the API through the *cyfitter.h* file.

Resources

Resolution	Digital Blocks					API Memory (Bytes)		Pins (per External I/O)
	Datapaths	Macro cells	Status Registers	Control Registers	Counter7	Flash	RAM	
Full UART	3	24	2	1	1			?
Full UART*	2	25	2	1	2			?
Full UART**	3	18	2	0	1			?
Half Duplex	1	26	1	1	1			?
RX Only	1	14	1	1	1			?
TX Only	2	11	1	1	0			?
TX Only*	1	12	1	1	1			?

* Parameter TxBitClkGenDP = false. (To switch go to Expression View of Configure Tab).

** Simple Full UART with settings:

Parity:	None
Flow Control:	None
Address Mode:	None
2 out of 3 voting:	Disable
Break signal:	Disable
CRC outputs:	Disable
Hardware TX:	Disable

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 "UART_1" to the first instance of a component in a given design. You can rename the instance 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".

void UART_Start(void)

Description:	Enable the UART operation. Sets the run bit of the control register for RX or TX or both as defined by the UART mode selection.
Parameters:	void
Return Value:	void
Side Effects:	Sets the run bit in the control registers of the UART.

void UART_Stop(void)

Description:	Disable the UART operation. Clears the run bit of the control register for RX or TX or both as defined by the UART mode selection.
Parameters:	void
Return Value:	void
Side Effects:	Clears the run bit in the control registers of the UART.

uint8 UART_ReadControlRegister(void)

Description:	Returns the current value of the control register
Parameters:	void
Return Value:	uint8 – Contents of the control register
Side Effects:	None

void UART_WriteControlRegister (uint8)

Description: Writes an 8-bit value into the control register

Parameters: uint8: Control Register Value

Return Value: void

Side Effects: None

void UART_EnableRxInt (void)

Description: Enables the internal interrupt irq

Parameters: void

Return Value: void

Side Effects: Only available if the RX internal interrupt implementation is selected in the UART

void UART_DisableRxInt (void)

Description: Disables the internal interrupt irq

Parameters: void

Return Value: void

Side Effects: Only available if the RX internal interrupt implementation is selected in the UART

void UART_SetRxInterruptMode (uint8)

Description: Configures the RX interrupt sources enabled

Parameters: uint8: Bit-Field containing the RX interrupts to enable. Based on the bit-field arrangement of the status register. This value must be a combination of status register bit-masks defined in the header file.

Return Value: void

Side Effects:

uint8 UART_ReadRxData (void)

Description: Returns the next byte of received data. ReadRXData returns data without checking status. That is, is there even new data and were there errors. It is up to the user to separately check status.

Parameters: void

Return Value: uint8: Received data from RX register

Side Effects:

uint8 UART_ReadRxStatus (void)

Description: Returns the current state of the status register

Parameters: void

Return Value: uint8: Current RX status register value

Side Effects: All status register bits are clear on read except UART_RX_STS_FIFO_NOTEMPTY. UART_RX_STS_FIFO_NOTEMPTY clears immediately after RX data register read. See the Registers section later in this data sheet.

uint8 UART_GetRxInterruptSource (void)

Description: Returns the status register containing the source of the interrupt

Parameters: void

Return Value: uint8: Current Status Register Value indicating which bit caused the interrupt

Side Effects: The Status register is clear on read and therefore any bits read by this function call must be handled accordingly even if they are not enabled as interrupt sources.

uint8 UART_GetChar (void)

Description: Returns the next byte of received data. GetChar is designed for ASCII characters only and returns an unit8 where 1-255 are valid characters and 0 is error -- or more typically no data present.

Parameters: void

Return Value: uint8: Character read from UART RX buffer. ASCII characters from 1 to 255 are valid. A returned zero signifies an error condition or no data available.

Side Effects:

Uint16 UART_GetByte (void)

Description: Reads UART RX buffer immediately, returns received character and error condition.

Parameters: void

Return Value: Uint16: MSB contains status and LSB contains UART RX data. If the MSB is non-zero, an error has occurred.

Side Effects:

uint8/uint16 UART_GetRxBufferSize (void)

Description: Returns the size (in bytes) of the RX memory buffer

Parameters: void

Return Value: uint8/uint16: Buffers size (in bytes)

Side Effects:

void UART_ClearRxbuffer (void)

Description: Clears the memory array of all received data

Parameters: void

Return Value: void

Side Effects:

void UART_SetRxAddressMode (uint8)

Description: Sets the software controlled Addressing mode used by the RX portion of the UART

Parameters: uint8: Enumerated value indicating the mode of RX addressing to implement

Return Value: void

Side Effects:

void UART_SetRxAddress1 (uint8)

Description: Sets the first of two hardware detectable Addresses

Parameters: uint8: Address #1 for hardware address detection

Return Value: void

Side Effects:

void UART_SetRxAddress2 (uint8)

Description: Sets the second of two hardware detectable Addresses

Parameters: uint8: Address #2 for hardware address detection

Return Value: void

Side Effects:

void UART_EnableTxInt (void)

Description: Enables the internal interrupt irq

Parameters: void

Return Value: void

Side Effects: Only available if the TX internal interrupt implementation is selected in the UART

void UART_DisableTxInt(void)

Description: Disables the internal interrupt irq

Parameters: void

Return Value: void

Side Effects: Only available if the TX internal interrupt implementation is selected in the UART

void UART_SetTxInterruptMode(uint8)

Description: Configures the TX interrupt sources enabled

Parameters: uint8: Bit-Field containing the TX interrupts to enable. Based on the bit-field arrangement of the status register. This value must be a combination of status register bit-masks defined in the header file.

Return Value: void

Side Effects:

void UART_WriteTxData(uint8)

Description: Puts a byte of data into the transmit buffer to be sent when the bus is available. WriteTxData sends a byte without checking for buffer room or status. It is up to the user to separately check status.

Parameters: uint8: data byte

Return Value: void

Side Effects:

uint8 UART_ReadTxStatus(void)

Description: Reads the status register for the TX portion of the UART

Parameters: void

Return Value: uint8: Contents of the TX Status register

Side Effects:

uint8 UART_GetTxInterruptSource(void)

Description: Returns the status register for the TX portion of the UART containing the source of an interrupt

Parameters: void

Return Value: uint8: Contents of the TX status register indicating the bit-wise source of the interrupt

Side Effects: This function reads the status register which is clear on read. It is up to the user to handle all bits in this return value accordingly, even if the bit was not enabled as an interrupt source the event happened and must be handled accordingly.

void UART_PutChar(uint8)

Description: Puts a byte of data into the transmit buffer to be sent when the bus is available. PutChar waits until the TX buffer has room and the sends the data. This is a blocking API.

Parameters: uint8: data byte

Return Value: void

Side Effects:

void UART_PutString(uint8*)

Description: Places data from a string into the memory buffer for transmitting

Parameters: uint8*: Address of null terminated string array residing in RAM or ROM

Return Value: void

Side Effects: This function will block if there is not enough memory to place the whole string, it will block until the entire string has been written to the transmit buffer.

void UART_PutArray(uint8*, uint8)

Description: Places data from a memory array into the memory buffer for transmitting

Parameters: uint8*: Address of the memory array residing in RAM or ROM, uint8: byte count

Return Value: void

Side Effects: This function will block if there is not enough memory to place the whole memory array, it will block until the entire array has been written to the transmit buffer.

void UART_PutCRLF(uint8)

Description: Writes a byte of data followed by a Carriage Return and Line Feed to the transmit buffer

Parameters: uint8: Data byte to transmit before the Carriage Return and Line Feed

Return Value: void

Side Effects: This function will block if there is not enough memory to place the three values in the transmit buffer.

uint8/16 UART_GetTxBufferSize(void)

Description: Returns the size (in bytes) of the Transmit buffer

Parameters: void

Return Value: uint8/16: Buffer size (in bytes)

Side Effects:

void UART_ClearTxBuffer(void)

Description: Clears all data from the TX buffer

Parameters: void

Return Value: void

Side Effects: Data waiting in the transmit buffer will not be sent, Data that is currently transmitting will finish transmitting (a single byte).

void UART_SendBreak(void)

Description: Transmit a break signal on the bus

Parameters: void

Return Value: void

Side Effects:

void UART_SetTxAddressMode(uint8)

Description: Sets the hardware Address mode configuration for the TX portion of the UART

Parameters: uint8

Return Value: void

Side Effects:

void UART_HardwareAddressEnable(void)

Description: Enables the Hardware Address Mode for the next transmit byte.

Parameters: void

Return Value: void

Side Effects:

void UART_LoadRxConfig(void)

Description: Loads the Rx configuration. Half Duplex UART is ready for receive byte.

Parameters: void

Return Value: void

Side Effects: Valid only for half duplex UART.It is the user's responsibility to ensure that any transmission is complete and it is safe to unload the Tx configuration.

void UART_LoadTxConfig(void)

Description: Loads the Tx configuration. Half Duplex UART is ready for transmit byte.

Parameters: void

Return Value: void

Side Effects: Valid only for half duplex UART.It is the user's responsibility to ensure that any transmission is complete and it is safe to unload the Rx configuration.

Defines

UART_INIT_RX_INTERRUPTS_MASK

Defines the initial configuration of the interrupt sources chosen by the user in the configuration GUI. This is a mask of the bits in the status register that have been enabled at configuration as sources for the RX interrupt.

UART_INIT_TX_INTERRUPTS_MASK

Defines the initial configuration of the interrupt sources chosen by the user in the configuration GUI. This is a mask of the bits in the status register that have been enabled at configuration as sources for the TX interrupt.

UART_TXBUFFERSIZE

Defines the amount of memory to allocate for the TX memory array buffer. This does not include the 4 bytes included in the FIFO.

UART_RXBUFFERSIZE

Defines the amount of memory to allocate for the RX memory array buffer. This does not include the 4 bytes included in the FIFO.

UART_NUMBER_OF_DATA_BITS

Defines the number of bits per data transfer which is used to calculate the Bit Clock Generator and Bit-Counter configuration registers

UART_BIT_CENTER

Based on the number of data bits this value is used to calculate the center point for the RX Bit-Clock Generator which is loaded into the configuration register at startup of the UART.

UART_RXHWADDRESS1

Defines the initial address selected in the configuration GUI. This address is loaded into the corresponding hardware register at startup of the UART.

UART_RXHWADDRESS2

Defines the initial address selected in the configuration GUI. This address is loaded into the corresponding hardware register at startup of the UART.

Sample Firmware Source Code

The following is a C language example demonstrating the basic functionality of the UART component. This example assumes the component has been placed in a design with the default name "UART_1."

Note If you rename your component you must also edit the example code as appropriate to match the component name you specify.

```
#include <device.h>

void main()
```

```
{
    uint8 i = 0;
    UART_1_Start();
    while(1)
    {
        UART_1_PutChar(i++);
    }
}
```

Functional Description

Default Configuration

The default configuration for the UART is as an 8-bit UART with no Flow control and Odd Parity, running at a baud-rate of 57.6Kbps

UART Parity: None

In this mode, there is no parity bit. The data flow is "Start, Data, Stop."

UART Parity: Odd

Odd parity begins with a parity bit of 1 and examines the data stream and toggles the parity on any 1 data value. At the end of the data transmission the state of the parity bit is transmitted. Odd parity makes sure that there is always a transition on the UART bus. In this mode if all data is zero then 1 parity will be sent. The data flow is "Start, Data, Parity, Stop." Odd parity is the most common of the parity types used.

UART Parity: Even

Even parity begins with a parity bit of 0 and examines the data stream and toggles the parity on any 1 data value. At the end of the data transmission the state of the parity bit is transmitted. The data flow is "Start, Data, Parity, Stop."

UART Parity: Mark/Space

Mark/Space parity is used to define either address or data bytes. When sending an address byte the parity bit is set to 1 and when transmitting data the parity bit is set to 0. At the end of the data transmission the state of the parity bit is transmitted. The data flow is "Start, Data, Parity, Stop" as the other parity modes but this bit is set by software before the transfer rather than being based on the data bit values. This parity is available for RS-485 like requirements.

UART Parity: Software

Software option is used to change parity by using control register and API function UART_WriteControlRegister(). The parity type can be dynamically changed between bytes without disrupting UART operation if this option selected but component will use more resources.

UART Stop Bits: One, Two

The number of stop bits is available as a synchronization mechanism. In slower systems it is sometimes necessary for two bit times to be available on the receiving side to be able to process the data before more data is sent. By sending two bit-widths of the stop signal the transmitter is allowing the receiver the time to interpret the data byte and parity. The data flow is the same "Start, Data,[Parity],Stop" except that the stop bit time may be one or two bit-widths.

UART Mode: Full UART (RX+TX)

This mode implements a full UART consisting of asynchronous Receiver and Transmitter. There is a single clock needed in this mode which defines the baud-rate for both the receiver and transmitter. The clock frequency input must be 8x the desired baud-rate.

UART Mode: RX Only

This mode implements only the Receiver portion of the UART. There is a single clock needed in this mode which defines the baud-rate for the receiver. The clock frequency input must be 8x the desired baud-rate for the Receiver to implement 2 out of 3 polling per bit.

UART Mode: TX Only

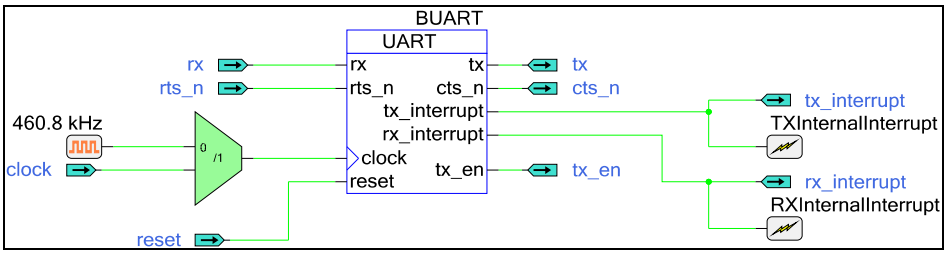
This mode implements only the transmitter portion of the UART. There is a single clock needed in this mode which defines the baud-rate for the transmitter. The clock frequency is 8x the baud-rate of the transmitter.

UART Flow Control: None, Hardware

Flow control on the UART provides separate rx and tx status indication lines CTS and RTS to the existing bus. When hardware flow control is enabled the RTS and CTS lines are available between this UART and another UART. RTS means Request to Send and is set by the receiver on the other UART in the system saying that it is OK to send data on the bus. CTS which means Clear To Send is an output of this UART telling the other device on the UART that I am able to receive data. Either of these bits is valid only before a transmission is started. If the signal is set or cleared after a transfer is started the transfer will complete as expected and the affect will only be on the next transfer.

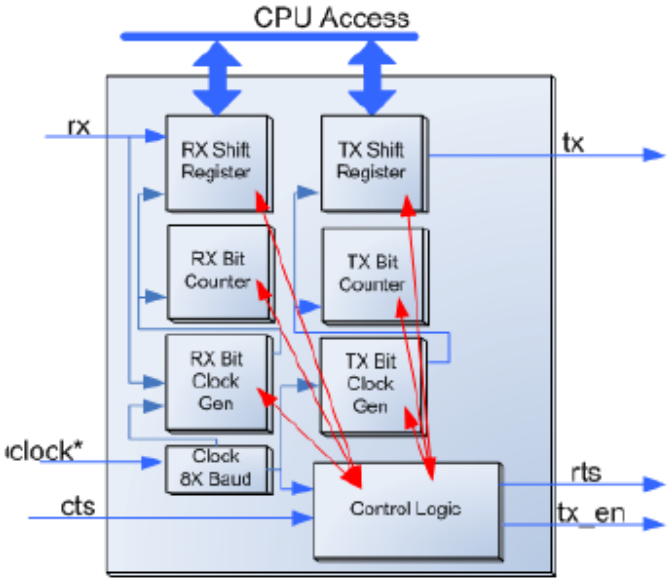
Block Diagram and Configuration

The UART is only available as a UDB configuration of blocks. The API is described above and the registers are described here to define the overall implementation of the UART.



The implementation is described in the following block diagram.

Figure 1 UDB Implementation



Registers

RX and TX Status

The status registers (RX and TX have independent status registers) are read-only registers that contain the various status bits defined for the UART. The value of these registers is available with the UART_ReadRxStatus() and UART_ReadTxStatus() function calls.

The interrupt output signals (tx_interrupt and rx_interrupt) are generated from an ORing of the masked bit-fields within each register. You can set the masks using the UART_SetRxInterruptMode() and UART_SetTxInterruptMode() function calls. Upon receiving an interrupt you can retrieve the interrupt source by reading the respective Status register with the

UART_GetRxInterruptSource() and UART_GetTxInterruptSource() function calls. The Status registers are clear on read so the interrupt source is held until one of the ReadRxStatus(), ReadTxStatus() or GetRxInterruptSource(), GetTxInterruptSource() functions is called. All operations on the status register must use the following defines for the bit-fields as these bit-fields may be moved around within the status register at build time.

There are several bit-fields masks defined for the status registers. Any of these bit-fields may be included as an interrupt source. The #defines are available in the generated header file (.h).

The status data is registered at the input clock edge of the UART giving all bits configured as Mode=1 the timing resolution of 8x the baud rate. Several of these bits are sticky (Mode = 1) and are cleared on a read of the status register, they are assigned as clear on read for use as an interrupt output of the UART. All other bits configured as transparent (Mode = 0) and represent the data directly from the inputs of the status register, they are not sticky and therefore not clear on read. All bits configured as Mode=1 are indicated with an asterisk (*) in the defines listed below.

RX Status Register

- UART_RX_STS_MRKSPC *– Status of the mark/space parity bit. This bit indicates whether a mark or space was seen in the parity bit location of the transfer. It is only implemented if the address mode is set to use hardware addressing.
- UART_RX_STS_BREAK *– Indicates that a break signal was detected in the transfer.
- UART_RX_STS_PAR_ERROR *– Indicates that a parity error was detected in the transfer.
- UART_RX_STS_STOP_ERROR *– Indicates that there was an error detected in the stop bit of the transfer.
- UART_RX_STS_OVERRUN *– Indicates that the receive buffer has been overrun.
- UART_RX_STS_FIFO_NOTEMPTY – Indicates whether or not the Receive FIFO is Not empty.
- UART_RX_STS_ADDR_MATCH *– Indicates that the address byte received matches one of the two addresses available for hardware address detection.

TX Status Register

- UART_TX_STS_FIFO_FULL – Indicates that the transmit FIFO is full. This should not be confused with the transmit buffer implemented in memory as the status of that buffer is not indicated in hardware, it must be checked in firmware.
- UART_TX_STS_FIFO_NOT_FULL– Indicates that the transmit FIFO is not full.
- UART_TX_STS_FIFO_EMPTY – Indicates that the transmit FIFO is empty.
- UART_TX_STS_COMPLETE * – Indicates that the last byte has been transmitted from FIFO.

Control

The Control register allows you to control the general operation of the UART. This register is written with the UART_WriteControlRegister() function call and read with the UART_ReadControlRegister(). Control register is not used if simple UART options are selected in customizer, for more details see Recourses paragraph. When reading or writing the control register you must use the bit-field definitions as defined in the header (.h) file. The #defines for the control register are as follows:

UART_HD_SEND

Used for dynamically reconfiguration RX or TX operation in Half Duplex mode. This bit is enabled by the UART_LoadTxConfig() function and cleared by the UART_LoadRxConfig() function.

UART_HD_SEND_BREAK

This bit is written by the UART_SendBreak() function in Half Duplex mode only. UART_HD_SEND bit should be enabled to send Break signal in Half Duplex mode.

UART_CTRL_MARK

Used to control the Mark/Space parity operation of the transmit byte. When set this bit indicates that the next byte transmitted will include a 1 (Mark) in the parity bit location and that all subsequent bytes will not until this bit is cleared and re-set by firmware.

UART_CTRL_PARITYTYPE_MASK

The parity type control is a 2-bit field used to define the parity operation for the next transfer. This bit-field will be 2 consecutive bits in the control register and all operations on this bit-field must use the #defines associated with the Parity types available. These are:

- UART__B_UART__NONE_REVA
- UART__B_UART__EVEN_REVA
- UART__B_UART__ODD_REVA
- UART__B_UART__MARK_SPACE_REVA

This bit-field is configured at initialization with the parity type defined in the "Parity Type" parameter and may be modified with the WriteControlRegister() API call.

UART_CTRL_RXADDR_MODE_MASK

The RX address mode control is a 3-bit field used to define the expected hardware addressing operation for the UART receive and transmit. This bit-field will be 3 consecutive bits in the control register and all operations on this bit-field must use the #defines associated with the compare modes available. These are:

- UART__B_UART__AM_SW_BYTE_BYTE
- UART__B_UART__AM_SW_DETECT_TO_BUFFER
- UART__B_UART__AM_HW_BYTE_BY_BYTE
- UART__B_UART__AM_HW_DETECT_TO_BUFFER
- UART__B_UART__AM_NONE

This bit-field is configured at initialization with the "AddressMode" parameter and may be modified with the WriteControlRegister() API call.

TX Data (8-bits)

The TX data register contains the transmit data value to send. This is implemented as a FIFO in the UART. There is a software state machine to control data from the transmit memory buffer to handle much larger portions of data to be sent. All API dealing with the transmitting of data must go through this register to place the data onto the bus. If there is data in this register and flow control indicates that data can be sent, then the data will be transmitted on the bus. As soon as this register (FIFO) is empty no more data will be transmitted on the bus until it is added to the FIFO. DMA may be setup to fill this FIFO when empty using the TX data register address defined in the header file.

RX Data

The RX data register contains the received data. This is implemented as a FIFO in the UART. There is a software state machine to control data movement from this receive FIFO into the memory buffer. Typically the RX interrupt will indicate that data has been received at which time that data has several routes to the firmware. DMA may be setup from this register to the memory array when not empty using the RX data register address defined in the header file.

Conditional Compilation Information

The UART API requires several conditional compile definitions to handle the multiple configurations it must support. It is required that the API conditionally compile on the RX and TX implementation chosen, the interrupts enabled and the Hardware addressing functionality chosen. The conditions defined are based on the parameters RXEnable, TXEnable, RXIntInterruptEnabled, TXIntInterruptEnabled, InternalClock and RXAddressMode. The API should never use these parameters directly but should use the defines listed below.

UART_RX_Enabled

This defines whether the RX portion of the UART has been chosen in the implementation.

UART_TX_Enabled

This defines whether the TX portion of the UART has been chosen in the implementation.

UART_HD_Enabled

This defines whether the Half Duplex portion of the UART has been chosen in the implementation.

UART_RX_IntInterruptEnabled

This defines whether the RX internal interrupt and ISR has been chosen in the implementation. If not chosen then the user must implement an external interrupt and the corresponding ISR to use interrupt driven transfers.

UART_TX_IntInterruptEnabled

This defines whether the TX internal interrupt and ISR has been chosen in the implementation. If not chosen then the user must implement an external interrupt and the corresponding ISR to use interrupt driven transfers.

UART_InternalClockUsed

This defines whether the user has chosen to use the internal clock and define the baud-rate or has chosen to provide and external clock and calculate the baud-rate at 1/8th of that input clock.

UART_RXHW_Address_Enabled

This defines whether the RX hardware addressing has been chosen and will be implemented in the hardware. This is available to limit resource usage if no hardware address detection is, or will be required in your design.

Constants

There are several constants defined for the status and control registers as well as some of the enumerated types. Most of these are described above for the Control and Status Register. However there are more constants needed in the header file to make all of this happen. Each of the register definitions requires either a pointer into the register data or a register address. Because of multiple Endianness` of the compilers it is required that the CY_GET_REGX and CY_SET_REGX macros are used for register accesses greater than 8-bits. These macros require the use of the _PTR definition for each of the registers.

It is also required that the control and status register bits be allowed to be placed and routed by the fitter engine in that we must have constants that define the placement of the bits. For each of

the status and control register bits there is an associated _SHIFT value which defines the bit's offset within the register. These are used in the header file to define the final bit mask as an _MASK definition (The _MASK extension is only added to bit-fields greater than a single bit, all single bit values drop the _MASK extension).

References

Not applicable

DC and AC Electrical Characteristics

The following values are indicative of expected performance and based on initial characterization data.

5.0V/3.3V DC and AC Electrical Characteristics

Parameter	Typical	Min	Max	Units	Conditions and Notes
Input					
Input Voltage Range	---		Vss to Vdd	V	
Input Capacitance	---		---	pF	
Input Impedance	---		---	Ω	
Maximum Clock Rate	---		67	MHz	