APPLICATION NOTE

AT07896: Universal Asynchronous Receiver Transceiver (UART)

ASF PROGRAMMERS MANUAL

Universal Asynchronous Receiver Transceiver (UART)

Note This driver applies to SAM3, SAM4S, SAM4E, SAM4N, SAM4C, SAM4CM, SAM4CP, and SAMG devices.

The Universal Asynchronous Receiver Transmitter features a two-pin UART that can be used for communication and debug/trace purposes and offers an ideal medium for in-situ programming solutions. Moreover, the association with two peripheral DMA controller (PDC) channels permits packet handling for these tasks with processor time reduced to a minimum.

The outline of this documentation is as follows:

- Prerequisites
- Module Overview
- Special Considerations
- Extra Information
- Examples
- API Overview

Table of Contents

Software License

1. Prerequisites

There are no prerequisites for this module.

2. Module Overview

The Universal Asynchronous Receiver Transmitter (UART) features a two-pin UART that can be used for communication and trace purposes and offers an ideal medium for in-situ programming solutions. Moreover, the association with peripheral DMA controller (PDC) permits packet handling for these tasks with processor time reduced to a minimum.

The API provides the following features:

1. Enable the UART peripheral clock in the PMC.
2. Enable the required UART PIOs (see pio.h).
3. Configure the UART by calling uart_init.
4. Send data through the UART using the uart_write.
5. Receive data from the UART using the uart_read; the availability of data. can be polled with uart_is_rx_ready.
6. Disable the transmitter and/or the receiver of the UART with. uart_disable_tx and uart_disable_rx.

3. Special Considerations

- This device provides a simple two pin (Recieve and Transmit) serial connection.
- For more sophisticated usage requiring hardware handshaking etc. consider using a USART or making use of GPIO pins and additional software. In this case consideration of power management regimes should also be considered.

4. Extra Information

For extra information, see Extra Information. This includes:

- Acronyms
- Dependencies
- Errata
- Document Revision History

5. API Overview

5.1 Variable and Type Definitions

5.1.1 Type sam_uart_opt_t

```
typedef struct sam_uart_opt sam_uart_opt_t
```

5.2 Structure Definitions

5.2.1 Struct sam_uart_opt

Table 5-1. Members

Type	Name	Description
uint32_t	ul_baudrate	Expected baud rate
uint32_t	ul_chmode	Configure channel mode (Normal, Automatic, Local_loopback or Remote_loopback)
uint32_t	ul_mck	MCK for UART
uint32_t	ul_mode	Initialize value for UART mode register

5.3 Function Definitions

5.3.1 Function uart_disable()

Disable the specified UART receiver and transmitter.

```
void uart_disable(
  Uart * p_uart)
```

Table 5-2. Parameters

Data direction	Parameter name	Description
[in]	p_uart	Pointer to a UART instance

5.3.2 Function uart_disable_interrupt()

Disable UART interrupts.

```
void uart_disable_interrupt(
  Uart * p_uart,
  uint32_t ul_sources)
```

Table 5-3. Parameters

Data direction	Parameter name	Description
[in]	p_uart	Pointer to a UART instance
[in]	ul_sources	Interrupts to be disabled

Note

For the meaning of ul_sources, refer to the description of the UART Interrupt Disable Register (IDR) in the appropriate data sheet.

5.3.3 Function uart_disable_rx()

Disable UART receiver.

```
void uart_disable_rx(
  Uart * p_uart)
```

Table 5-4. Parameters

Data direction	Parameter name	Description
[in]	p_uart	Pointer to a UART instance

5.3.4 Function uart_disable_tx()

Disable UART transmitter.

```
void uart_disable_tx(
  Uart * p_uart)
```

Table 5-5. Parameters

Data direction	Parameter name	Description
[in]	p_uart	Pointer to a UART instance

5.3.5 Function uart_enable()

Enable the specified UART receiver and transmitter.

```
void uart_enable(
  Uart * p_uart)
```

Table 5-6. Parameters

Data direction	Parameter name	Description
[in]	p_uart	Pointer to a UART instance

5.3.6 Function uart_enable_interrupt()

Enable UART interrupts.

```
void uart_enable_interrupt(
  Uart * p_uart,
  uint32_t ul_sources)
```

Table 5-7. Parameters

Data direction	Parameter name	Description
[in]	p_uart	Pointer to a UART instance

Data direction	Parameter name	Description
[in]	ul_sources	Interrupts to be enabled

Note For the meaning of ul_sources, refer to the description of the UART Interrupt Enable Register (IER) in the appropriate data sheet.

5.3.7 Function uart_enable_rx()

Enable UART receiver.

```
void uart_enable_rx(
  Uart * p_uart)
```

Table 5-8. Parameters

Data direction	Parameter name	Description
[in]	p_uart	Pointer to a UART instance

5.3.8 Function uart_enable_tx()

Enable the UART transmitter.

```
void uart_enable_tx(
  Uart * p_uart)
```

Table 5-9. Parameters

Data direction	Parameter name	Description
[in]	p_uart	Pointer to a UART instance

5.3.9 Function uart_get_interrupt_mask()

Read the UART interrupt mask.

```
uint32_t uart_get_interrupt_mask(
  Uart * p_uart)
```

Table 5-10. Parameters

Data direction	Parameter name	Description
[in]	p_uart	Pointer to a UART instance

Returns The interrupt mask value.

Note For the meaning of ul_sources, refer to the description of the UART Interrupt Enable Register (IER) in the appropriate data sheet.

5.3.10 Function uart_get_pdc_base()

Get the UART PDC base address.

```
Pdc * uart_get_pdc_base(
  Uart * p_uart)
```

Table 5-11. Parameters

Data direction	Parameter name	Description
[in]	p_uart	Pointer to a UART instance

Returns

UART PDC registers base for PDC driver to access.

5.3.11 Function uart_get_status()

Get current UART status.

```
uint32_t uart_get_status(
  Uart * p_uart)
```

Table 5-12. Parameters

Data direction	Parameter name	Description
[in]	p_uart	Pointer to a UART instance

Returns

The current UART status.

Note

For the meaning of the value returned, refer to the description of the UART Status Register(SR) in the appropriate data sheet.

5.3.12 Function uart_init()

Configure the UART with the specified parameters.

```
uint32_t uart_init(
  Uart * p_uart,
  const sam_uart_opt_t * p_uart_opt)
```

Note

The PMC and PIOs must be configured first.

For more detail see sam_uart_opt_t

Table 5-13. Parameters

Data direction	Parameter name	Description
[in]	p_uart	Pointer to a UART instance

Data direction	Parameter name	Description
[in]	p_uart_opt	Pointer to sam_uart_opt_t instance

Table 5-14. Return Values

Return value	Description
0	Success
1	Bad baud rate generator value

5.3.13 Function uart_is_rx_buf_end()

Check if a receive buffer is filled.

```
uint32_t uart_is_rx_buf_end(
  Uart * p_uart)
```

Table 5-15. Parameters

Data direction	Parameter name	Description
[in]	p_uart	Pointer to a UART instance

Table 5-16. Return Values

Return value	Description
1	Receive is completed
0	Receive is still pending

5.3.14 Function uart_is_rx_buf_full()

Check if both receive buffers are full.

```
uint32_t uart_is_rx_buf_full(
  Uart * p_uart)
```

Table 5-17. Parameters

Data direction	Parameter name	Description
[in]	p_uart	Pointer to a UART instance

Table 5-18. Return Values

Return value	Description
1	Receive buffers are full
0	Receive buffers are not full

5.3.15 Function uart_is_rx_ready()

Check if Received data is ready. Check if data has been received and loaded into the Recieve Holding register(RHR).

```
uint32_t uart_is_rx_ready(
  Uart * p_uart)
```

Table 5-19. Parameters

Data direction	Parameter name	Description
[in]	p_uart	Pointer to a UART instance

Table 5-20. Return Values

Return value	Description
1	One data has been received
0	No data has been received

5.3.16 Function uart_is_tx_buf_empty()

Check if both transmit buffers are empty.

```
uint32_t uart_is_tx_buf_empty(
  Uart * p_uart)
```

Table 5-21. Parameters

Data direction	Parameter name	Description
[in]	p_uart	Pointer to a UART instance

Table 5-22. Return Values

Return value	Description
1	Transmit buffer is empty
0	Transmit buffer is not empty

5.3.17 Function uart_is_tx_buf_end()

Check if a transmit buffer is empty.

```
uint32_t uart_is_tx_buf_end(
  Uart * p_uart)
```

Table 5-23. Parameters

Data direction	Parameter name	Description
[in]	p_uart	Pointer to a UART instance

Table 5-24. Return Values

Return value	Description
1	Transmit is completed
0	Transmit is still pending

5.3.18 Function uart_is_tx_empty()

Check if Transmit Hold Register is empty. Check if the last data written in Transmit Holding Register (THR) has been loaded into the Transmit Shift Register(TSR) and the last data loaded in TSR has been transmitted.

```
uint32_t uart_is_tx_empty(
  Uart * p_uart)
```

Table 5-25. Parameters

Data direction	Parameter name	Description
[in]	p_uart	Pointer to a UART instance

Table 5-26. Return Values

Return value	Description
1	Transmitter is empty
0	Transmitter is not empty

5.3.19 Function uart_is_tx_ready()

Check if Transmit is Ready. Check if data has been loaded into the Transmit Holding Register(THR) and is waiting to be loaded in the Transmit Shift Register (TSR).

```
uint32_t uart_is_tx_ready(
  Uart * p_uart)
```

Table 5-27. Parameters

Data direction	Parameter name	Description
[in]	p_uart	Pointer to a UART instance

Table 5-28. Return Values

Return value	Description
1	Data has been transmitted
0	Transmit is not ready, data pending

5.3.20 Function uart_read()

Read from UART Receive Holding Register(RHR). Before reading user should check if rx is ready.

```
uint32_t uart_read(
  Uart * p_uart,
  uint8_t * puc_data)
```

Table 5-29. Parameters

Data direction	Parameter name	Description
[in]	p_uart	Pointer to a UART instance
[out]	puc_data	Recieved data

Table 5-30. Return Values

Return value	Description
0	Success
1	I/O Failure, UART is not ready

5.3.21 Function uart_reset()

Reset UART receiver and transmitter.

```
void uart_reset(
  Uart * p_uart)
```

Table 5-31. Parameters

Data direction	Parameter name	Description
[in]	p_uart	Pointer to a UART instance

5.3.22 Function uart_reset_rx()

Reset UART receiver.

```
void uart_reset_rx(
  Uart * p_uart)
```

Table 5-32. Parameters

Data direction	Parameter name	Description
[in]	p_uart	Pointer to a UART instance

5.3.23 Function uart_reset_status()

Reset status bits.

```
void uart_reset_status(
  Uart * p_uart)
```

Table 5-33. Parameters

Data direction	Parameter name	Description
[in]	p_uart	Pointer to a UART instance

5.3.24 Function uart_reset_tx()

Reset UART transmitter.

```
void uart_reset_tx(
  Uart * p_uart)
```

Table 5-34. Parameters

Data direction	Parameter name	Description
[in]	p_uart	Pointer to a UART instance

5.3.25 Function uart_set_clock_divisor()

Set UART clock divisor value.

```
void uart_set_clock_divisor(
  Uart * p_uart,
  uint16_t us_divisor)
```

Table 5-35. Parameters

Data direction	Parameter name	Description
[in]	p_uart	Pointer to a UART instance
[in]	us_divisor	Value to be set

5.3.26 Function uart_write()

Write to UART Transmit Holding Register (THR). Before writing the user should check if tx is ready (or empty).

```
uint32_t uart_write(
  Uart * p_uart,
  const uint8_t uc_data)
```

Table 5-36. Parameters

Data direction	Parameter name	Description
[in]	p_uart	Pointer to a UART instance
[in]	uc_data	Data to be sent

Table 5-37. Return Values

Return value	Description
0	Success
1	I/O Failure, UART is not ready

6. UART SleepWalking Example

6.1 Purpose

The example demonstrates how to use SleepWalking function of the UART.

6.2 Requirements

This example can be used on SAMG53 device.

The code can be found in the uart_sleepwalking_example folder.

Note The example use a loose match condition to wake-up the system from wait mode.

6.3 Description

The example first tests the sleepwalking function in active mode. If the 's' character is received, the match interrupt is triggered. Then it tests the SleepWalking function in wait mode. As soon as a data is received, the system wake-up from wait mode.

6.4 Usage

1. Build the program and download it into the evaluation board.
2. On the computer, open, and configure a terminal application (e.g., HyperTerminal on Microsoft® Windows®) with these settings:
   - 115200 bauds
   - 8 bits of data
   - No parity
   - 1 stop bit
   - No flow control
3. In the terminal window, the following text should appear (values depend on the board and chip used):

```
    -- Uart Sleepwalking Example xxx --
    -- xxxxxx-xx
    -- Compiled: xxx xx xxxx xx:xx:xx --
```

4. the sent text should appear.

7. Extra Information

7.1 Acronyms

Below is a table listing the acronyms used in this module, along with their intended meanings.

Acronym	Definition
IDR	Interrupt Disable Register
IER	Interrupt Enable Register
PDC	Peripheral DMA Channel
SR	Status Register

7.2 Dependencies

This driver has the following dependencies:

- None

7.3 Errata

There are no errata related to this driver.

8. Examples

For an example application for the UART, see:

- UART SleepWalking Example

Index

Document Revision History

Doc. Rev.	Date	Comments
42299A	05/2014	Initial document release

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | www.atmel.com

Disclaimer: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. EXCEPT AS SET FORTH IN THE ATMEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATMEL WEBSITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS AND PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and products descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.