1. **Summary**

Acting as a peripheral, the UART module provides serial communication capabilities to the Airi5c processor. After a complete redesign, this Module now supports the following features:

* AHB-Lite interface
* Separate registers for control, rx and tx status, all with set/clear access capability
* configurable and independent rx and tx FIFO stack size (1 – 256 frames)
* configurable number of data bits (5, 6, 7, 8, 9)
* configurable parity settings (none, even, odd)
* configurable number of stop bits (1, 1.5, 2)
* support for hardware flow control (rts/cts)
* support for default and none default baud rates
* accessible rx and tx FIFO stack size
* configurable and independent watermark settings for rx and tx stack size with interrupt generation
* error detection
* extensive interrupt capabilities

1. **Parameters**

These parameters have to be set at compile time, they cannot be changed at runtime.

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Default** | **Description** |
| BASE\_ADDR | 0xC0000200 | Base address of the UART module, the addresses of all registers are increments of 4 beginning at this address |
| TX\_ADDR\_WIDTH | 5 | Address width of the tx stack, defining the max size of the tx stack () |
| RX\_ADDR\_WIDTH | 5 | Address width of the rx stack, defining the max size of the rx stack () |
| TX\_MARK | 8 | Tx watermark, a status signal is generated, when the tx stack size falls below this value |
| RX\_MARK | 24 | Rx watermark, a status signal is generated, when the rx stack size exceeds this value |

1. **Registers**

The UART module includes the following 10 32-bit data, control and status registers, which can be accessed via AHB-Lite interface. In the old processor design, the address space of each peripheral was restricted to 4 32-bit words. With the introduction of the new UART module this number has been increased to 64. Remember that the base address of each peripheral has been changed accordingly and need to be changed in your programs too!

Reserved fields are hardwired to zero, writing to those fields has no effect. Errors are normally set at the end of the particular frame where the error occurred. The only exceptions are tx overflow error and rx underflow error, which are set immediately. Once set, all error stay set as long as they get reset manually.

|  |  |  |
| --- | --- | --- |
| **Address** | **Type** | **Description** |
| BASE\_ADDR + 0x00  0xC0000200 | FIFO stack | Write access writes to tx stack, read access reads from rx stack |
| 0xC0000204  BASE\_ADDR + 0x04 | Ctrl reg | This register contains all communication settings, such as data bits, parity, stop bits, flow control and baud rate |
| BASE\_ADDR + 0x08  0xC0000208 | Ctrl reg set | Writing to this register automatically sets the specified bits in ctrl reg |
| BASE\_ADDR + 0x0C  0xC000020C | Ctrl reg clr | Writing to this register automatically clears the specified bits in ctrl reg |
| BASE\_ADDR + 0x10  0xC0000210 | Tx stat reg | This register contains the tx status, such as tx stack size, errors and interrupt enables |
| BASE\_ADDR + 0x14  0xC0000214 | Tx stat reg set | Writing to this register automatically sets the specified bits in tx stat reg |
| BASE\_ADDR + 0x18  0xC0000218 | Tx stat reg clr | Writing to this register automatically clears the specified bits in tx stat reg |
| BASE\_ADDR + 0x1C  0xC000021C | Rx stat reg | This register contains the rx status, such as rx stack size, errors and interrupt enables |
| BASE\_ADDR + 0x20  0xC0000220 | Rx stat reg set | Writing to this register automatically sets the specified bits in rx stat reg |
| BASE\_ADDR + 0x24  0xC0000224 | Rx stat reg clr | Writing to this register automatically clears the specified bits in rx stat reg |

* 1. **Control Register**

|  |  |  |
| --- | --- | --- |
| **Bits** | **Access** | **Description** |
| 31:29 | rw | Number of data bits (0b000: 5, …, 0b011: 8, 0b100: 9) |
| 28:27 | rw | Parity setting (0b00: none, 0b01: even, 0b10: odd) |
| 26:25 | rw | Number of stop bits (0b00: 1, 0b01: 1.5, 0b10: 2) |
| 24 | rw | Flow control (0b0: none, 0b1: rts/cts) |
| 23:0 | rw | Baud register, containing the number of clock cycles per bit () |

If the number of data bits is set to 9, the number of stop bits is automatically set to 1 and parity is set to none. When writing an invalid value (e.g. 0b101: 10 data bits), the particular field is set to the highest possible value instead. A set access resulting in an invalid value is ignored. Modifications of the bits in the control register come into effect immediately. Make sure that there is no active communication when modifying this register, otherwise data loss and communication errors can occur. The default communication settings are:

* Data bits: 8
* Parity: none
* Stop bits: 1
* Flow control: none
* Baud rate: 9600 (at 32 MHz)
  1. **Tx status register**

|  |  |  |
| --- | --- | --- |
| **Bits** | **Access** | **Description** |
| 31:20 | r | reserved |
| 19 | rw | Tx overflow error interrupt enable |
| 18 | rw | Tx watermark reached interrupt enable |
| 17 | rw | Tx stack empty interrupt enable |
| 16 | rw | Tx stack full interrupt enable |
| 15:12 | r | reserved |
| 11 | rw | Tx overflow error (write to full tx stack) |
| 10 | r | Tx stack size ≤ tx watermark |
| 9 | r | Tx stack empty |
| 8 | r | Tx stack full |
| 7:0 | r | Tx stack size |

* 1. **Rx status register**

|  |  |  |
| --- | --- | --- |
| **Bits** | **Access** | **Description** |
| 31:24 | rw | reserved |
| 23 | rw | Rx frame error interrupt enable |
| 22 | rw | Rx parity error interrupt enable |
| 21 | rw | Rx noise error interrupt enable |
| 20 | rw | Rx underflow error interrupt enable |
| 19 | rw | Rx overflow error interrupt enable |
| 18 | rw | Rx watermark reached interrupt enable |
| 17 | rw | Rx stack empty interrupt enable |
| 16 | rw | Rx stack full interrupt enable |
| 15 | rw | Rx frame error (no stop bit detected) |
| 14 | rw | Rx parity error (parity received ≠ calculated) |
| 13 | rw | Rx noise error (samples taken from one bit differ) |
| 12 | rw | Rx underflow error (read from empty rx stack) |
| 11 | rw | Rx overflow error (received data while rx stack was full) |
| 10 | r | rx stack size ≥ rx watermark |
| 9 | r | Rx stack empty |
| 8 | r | Rx stack full |
| 7:0 | r | Rx stack size |

1. **Interrupts**

The UART module supports several interrupts, which are stated in the tx and rx status register. All interrupts are disabled by default and have to be enabled manually if desired. Besides the individual interrupt signals, there is also a special signal “int\_any” available at the port of this module which is set whenever at least one interrupt has occurred. Some interrupt signals are connected to the specific error signals. In this case an interrupt service routine has to reset the specific error flag, otherwise the interrupt will fire again and again.

1. **Functionality**

Transmitting data can be achieved writing to the FIFO stack address, which effectively writes to the tx stack. As long as the tx stack is not full, new data can be written to it immediately in a row. The UART module automatically reads the data in the tx stack and transmits it via the tx pin. When writing to the tx stack while it is full, the data written to it is lost and the tx overflow error is set.

Incoming data via the rx pin is automatically written to the rx stack, which can be read from by reading from the FIFO stack address. As long as the rx stack is not full, data can be received. As soon as the rx stack is full, any incoming data is lost and the rx overflow error is set. The data in the rx stack (as well as the tx stack) never gets overwritten. In order to free stack memory, data has to be read.

Each bit of incoming data is sampled 3 times at and around its timed midpoint. If the samples differ, the noise error is set at the end of the specific frame.

1. **Flow Control**

The UART module supports rts/cts hardware flow control. Rts is an output of the receiver called *ready to send* which is connected to the cts input of the transmitter called *clear to send* (and vice versa). Set to high, rts signals the transmitter, that its rx stack is not full and new data can be received. As soon as the rx stack is full, rts is set to low, signaling the transmitter that it has to stop transmission. To prevent data loss, rts is already set to low, when there is only space for 4 more frames in the rx stack.

The rts and cts pins are currently not connected in our FPGA designs!