UART User Guide

Introduction

This user guide provides comprehensive information about the Agate Logic Universal Asynchronous Receiver/Transmitter (UART) Intellectual Property (IP).

UART Function

The Agate Logic UART implements asynchronous transmitter, receiver logic and programmable baud rate generator logic.

Transmitter and Receiver Features

- Support 5 to 8 bits per character
- Odd, Even or no parity detection and generation
- 1,1.5 or 2 stop bit detection and generation
- False start bit detection and recovery
- Independent transmit and receive FIFO buffer, up to 32-byte.
- Support transmitting done flag.
- Support frame error flag and optional parity error flags.
- Support Transmitter FIFO overflow and full flags.
- Support Receiver FIFO overflow, underflow, full and empty flags.

Baud rate generator Features

- Programmable baud rate.
- Divides any input clock by 2 to 65536 and generates the 16×baud clock.

Transmitter Logic

The Transmitter input data is 8-bit, and push into the internal FIFO, however, you can control the transmitting data width to 5-, 6-, 7-, 8-bit, and the Transmitter will start if the internal FIFO isn't empty. The Transmitter support odd or even parity bit, and you can choose no parity bit too. And we support 1, 1.5 and 2 bit stop bit. The Transmitter supply tf_full and tf_overflow flag for internal FIFO, and tx_done flag means the internal FIFO is empty and the final data has been transmitted.

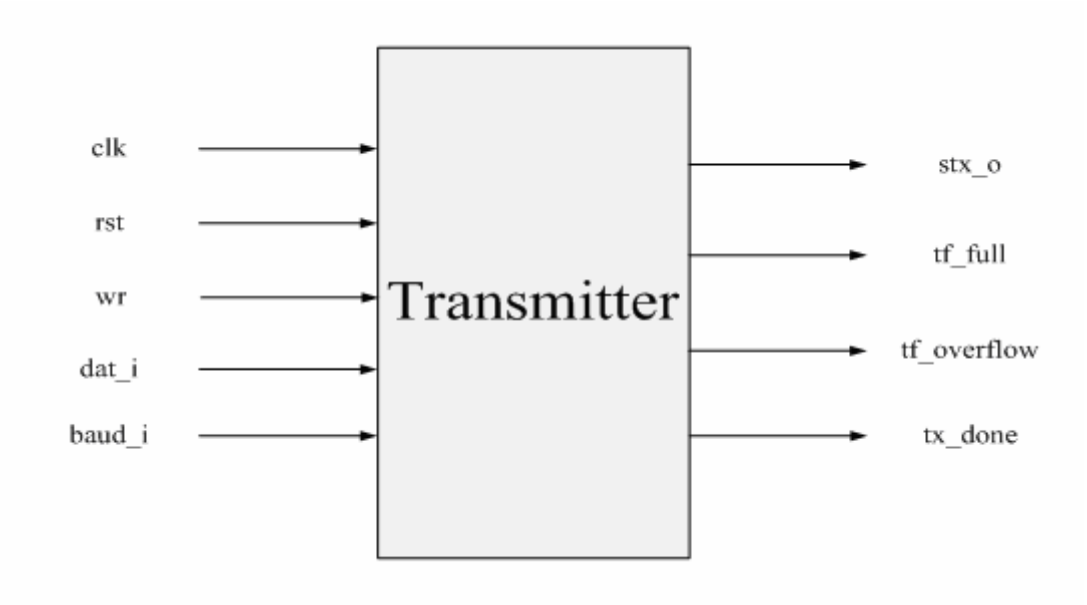

Figure1. Transmitter Interface

Table 1 describes the interface of the UART Transmitter.

Port name	In/Out	Width	Description
dat_i	In	8	Transmitter FIFO data input
clk	In	1	Transmitter main clock.
baud_i	In	1	Band rate input.
wr	In	1	Transmitter FIFO write enable signal, high active.
rst	In	1	Asynchronous reset, high active
stx_o	Out	1	Serial data output
tf_full	Out	1	Transmitter FIFO full, high active, optional
tf_overflow	Out	1	Active when wr active but transmitter FIFO is full, high active, optional
tx_done	Out	1	Transmitting done, no data in transmitter FIFO and the final data has been sent out through stx_o, high active, optional

Table1. UART Transmitter Interface Description

Parameter	default	Description
tx_data_width	2'b11	Transmitting data width 00: 5bit 01: 6bit 10: 7bit 11:8bit
tx_nparity_bit	1'b1	1: no parity bit; 0: enable parity bit
tx_parity_bit	1'b1	0: odd parity; 1: even parity
tx_one_stop_bit	1'b1	1: only one bit stop; 0: stop bit decided by tx_more_stop_bit
tx_more_stop_bit	1'b1	0: 1.5 bit stop; 1: 2 bit stop
tf_depth	1'b1	Used to set Transmitter fifo depth(1-32), the FIFO data width is 8.
tf_pointer_w	1'b1	Decided by tf_depth, from 1-5 bit

Table2. UART Transmitter parameter table

Note:

1. If you set tx_nparity_bit = 1, then the parameter tx_parity_bit is invalid.
2. If you set tx_one_stop_bit = 1, then the parameter tx_more_stop_bit is invalid.

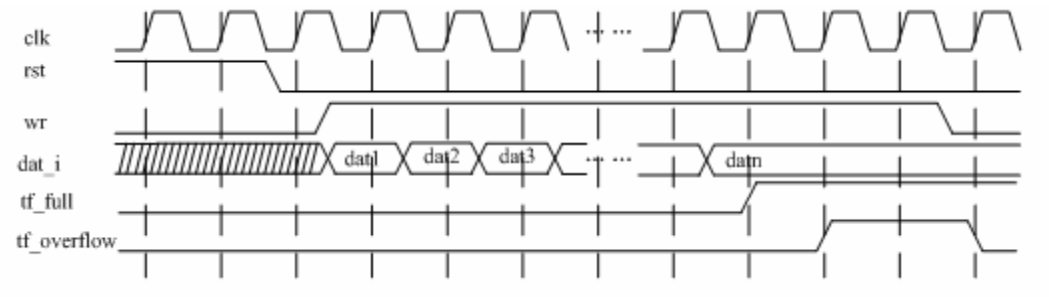

Figure2. Transmitter data write timing diagram

The next paragraph is the verilog instantiation model:

```
module uart_transmitter (
        clk,            //transmitter clock input
        rst,            //transmitter asynchronous reset input, high active
        wr,             //transmitter fifo write enable signal, high active
        dat_i,          //transmitter fifo data input
        baud_i,         //transmitter baud rate input
        stx_o,          //transmitter serial data out
        tf_overflow,    //tranmsitter fifo overflow flag output
        tf_full,        //transmitter fifo full flag output
        tx_done         //transmitter done flag output, all data in tx_fifo have been send out
        );
parameter tf_depth          = 1;        //fifo depth
parameter tf_pointer_w      = 1;        //fifo depth pointer
parameter tx_data_width     = 2'b11;    //data width 00: 5bit   01: 6bit   10: 7bit    11: 8bit
```

```
parameter tx_nparity_bit      = 1'b1;    //no parity bit      1: has parity bit
parameter tx_parity_bit       = 1'b1;    //0: odd parity check, 1: even parity check
parameter tx_one_stop_bit     = 1'b1;    //1bit stop
parameter tx_more_stop_bit    = 1'b1;    //0: 1.5 bit stop; 1: 2bit stop
input             clk;
input             rst;
input             wr;
input    [7:0]    dat_i;
input             baud_i;
output            stx_o;
output            tf_overflow;
output            tf_full;
output            tx_done;
uart_transmitter my_uart_transmitter(
                  .clk                (clk),
                  .rst                (rst),
                  .wr                 (wr),
                  .dat_i              (dat_i),
                  .baud_i             (baud_i),
                  .stx_o              (stx_o),
                  .tf_overflow        (tf_overflo),
                  .tf_full            (tf_full),
                  .tx_done            (tx_done)
                  );
defparam my_uart_transmitter.tf_depth = tf_depth ;
defparam my_uart_transmitter.tf_pointer_w = tf_pointer_w ;
defparam my_uart_transmitter.tx_data_width = tx_data_width ;
defparam my_uart_transmitter.tx_nparity_bit = tx_nparity_bit ;
defparam my_uart_transmitter.tx_parity_bit = tx_parity_bit ;
defparam my_uart_transmitter.tx_one_stop_bit = tx_one_stop_bit ;
defparam my_uart_transmitter.tx_more_stop_bit = tx_more_stop_bit ;
endmodule
```

Receiver Logic

The Receiver output data is 8-bit, and pop from the internal FIFO. The Receiver support 5-, 6-, 7-, 8-bit serial data format, and start receiving if the srx_i port has one bit start bit. The Receiver support odd or even parity checking, and you can choose no parity checking too. And we support 1, 1.5 and 2 bit stop bit checking. The Receiver supply rf_full, rf_empty, rf_underflow and rf_overflow flag for internal FIFO, and rparity_error flag for parity checking, rframing_error flag for stop bit checking.

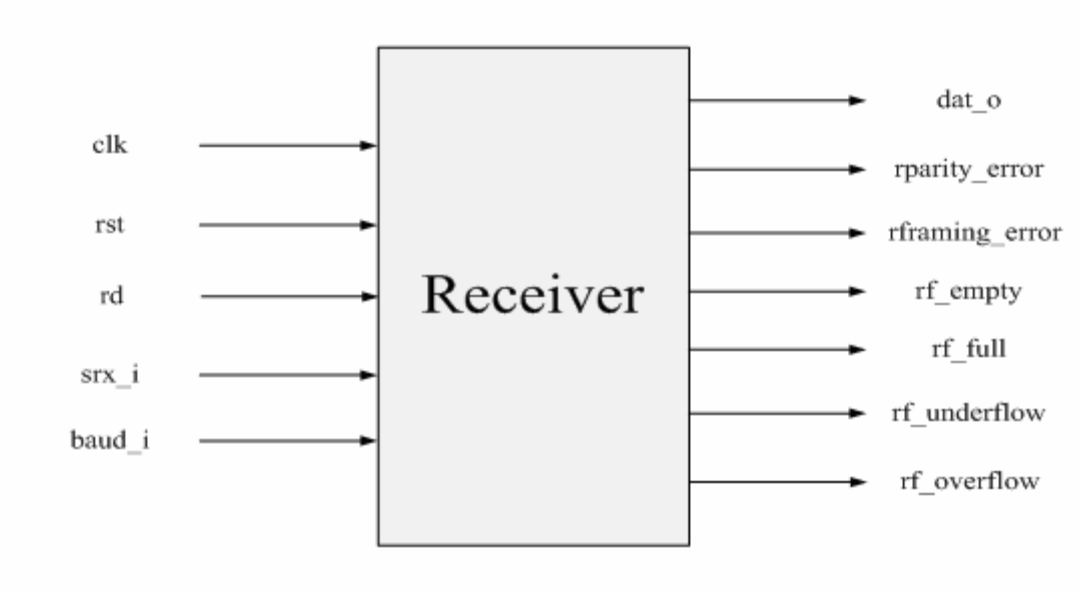

Figure3. Receiver Interface

Table 3 describes the interface of the UART Receiver.

Port name	In/Out	Width	Description
srx_i	In	1	Serial data input
clk	In	1	Receiver main clock.
baud_i	In	1	Band rate input.
rd	In	1	Receiver FIFO read enable signal, high active.
rst	In	1	Asynchronous reset, high active
dat_o	Out	8	Receiver FIFO output data
rframing_error	Out	1	Frame error, active when detected stop bit error
rparity_error	Out	1	Parity error, active when detected parity bit error, optional
rf_empty	Out	1	Receiver FIFO empty, high active, optional
rf_full	Out	1	Receiver FIFO full, high active, optional
rf_overflow	Out	1	Active when srx_i received valid data but receiver FIFO is full, high active, optional
rf_underflow	Out	1	Active when rd active but receiver FIFO is empty, high active, optional

Table3. UART Receiver Interface Description

Parameter	default	Description
rx_data_width	2'b11	Received data width 00: 5bit 01: 6bit 10: 7bit 11:8bit
rx_nparity_bit	1'b1	1: no parity bit; 0: enable parity bit
rx_parity_bit	1'b1	0: odd parity; 1: even parity
rx_one_stop_bit	1'b1	1: only one bit stop; 0: stop bit decided by rx_more_stop_bit
rx_more_stop_bit	1'b1	0: 1.5 bit stop; 1: 2 bit stop
rf_depth	1'b1	Use to set receiver fifo depth(1-32), the fifo data width is 8.
rf_pointer_w	1'b1	Decided by rf_depth, from 1-5 bit

Table4. UART Receiver parameter table

Note:

1. If you set rx_nparity_bit = 1, then the parameter rx_parity_bit is invalid.
2. If you set rx_one_stop_bit = 1, then the parameter rx_more_stop_bit is invalid.

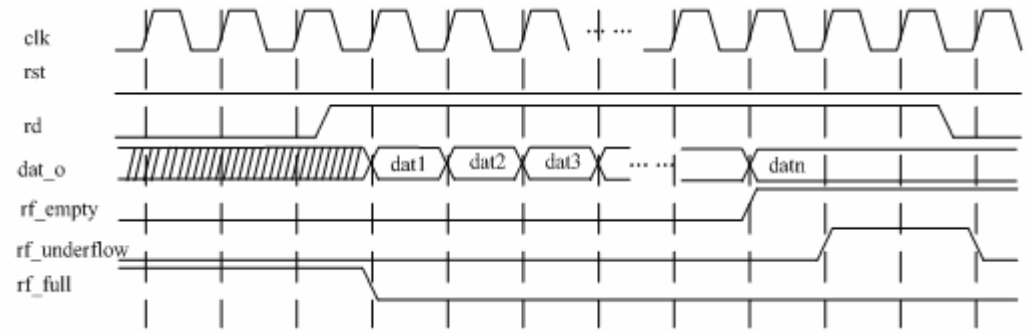

Figure4. Receiver read timing diagram

The next paragraph is the verilog instantiation model:

```
module uart_receiver (
        clk,            //receiver clock input
        rst,            //receiver asynchronous reset input, high active
        rd,             //receiver fifo read enable signal, hige active
        srx_i,          //receiver serial data in
        baud_i,         //receiver baud rate input
        dat_o,          //receiver fifo data output
        rf_overflow,    //receiver fifo overflow flag output
        rf_underflow,   //receiver fifo underflow flag output
        rf_full,        //receiver fifo full flag output
        rf_empty,       //receiver fifo empty flag output
        rparity_error,  //receiver parity error flag output
```

```
        rframing_error      //receiver framing error flag output
        );
parameter rf_depth          = 1'b1;         //fifo depth
parameter rf_pointer_w      = 1'b1;         //fifo depth pointer
parameter rx_data_width     = 2'b11;        //data width 00:5bit   01:6bit   10:7bit     11:8bit
parameter rx_nparity_bit    = 1'b1;         //no parity bit     1: has parity bit
parameter rx_parity_bit     = 1'b1;         //0: odd parity check, 1: even parity check
parameter rx_one_stop_bit   = 1'b1;         //1bit stop
parameter rx_more_stop_bit  = 1'b1;         //0: 1.5 bit stop; 1: 2bit stop
input           clk;
input           rst;
input           rd;
input           srx_i;
input           baud_i;
output  [7:0]   dat_o;
output          rf_underflow;
output          rf_overflow;
output          rf_full;
output          rf_empty;
output          rparity_error;
output          rframing_error;
uart_receiver my_uart_receiver(
                .clk            (clk),
                .rst            (rst),
                .rd             (rd),
                .srx_i          (srx_i),
                .baud_i         (baud_i),
                .dat_o          (dat_o),
                .rf_overflow    (rf_overflow),
                .rf_underflow   (rf_underflow),
                .rf_full        (rf_full),
                .rf_empty       (rf_empty),
                .rparity_error  (rparity_error),
                .rframing_error (rframing_erro)
```

```
                    );
defparam my_uart_receiver.rf_depth = rf_depth ;
defparam my_uart_receiver.rf_pointer_w = rf_pointer_w ;
defparam my_uart_receiver.rx_data_width = rx_data_width ;
defparam my_uart_receiver.rx_nparity_bit = rx_nparity_bit ;
defparam my_uart_receiver.rx_parity_bit = rx_parity_bit ;
defparam my_uart_receiver.rx_one_stop_bit = rx_one_stop_bit ;
defparam my_uart_receiver.rx_more_stop_bit = rx_more_stop_bit ;
endmodule
```

Baudgen Logic

Baudgen is used to generate the baud rate for transmitter and receiver.

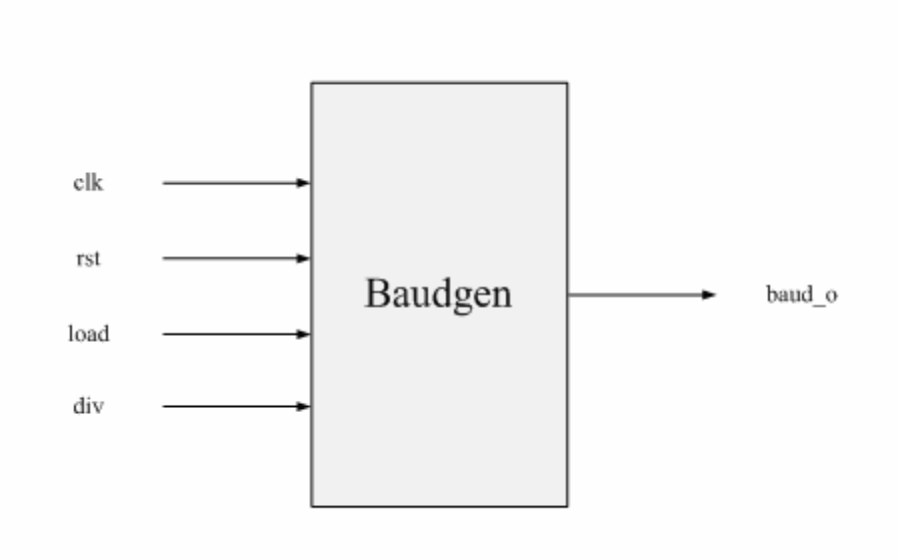

Figure5. Baudgen Interface

The input clk is main clock.

$$Fre_{baud\_o} = Fre_{clk} \quad / \quad (div+1).$$

The baud rate clock is generated by dividing the input reference clock by 16 and the div value.

$$Baud\_rate = Fre_{baud\_o} \ / \ 16$$

When load active, Baudgen can load div to generate baud_o, otherwise Baudgen can not load div value to generate baud_o.
The port div and load are optional, if you need not this programmable port to control baud_o, you can use parameter div_param to substitute it.

Table 5 describes the interface of the UART Baudgen.

Port name	In/Out	Width	Description
clk	In	1	Baudgen main clock.
div	In	div_width	Clock division, width is up to 16 bits. It has to larger than 0.
load	In	1	When load is high, Baudgen can load div to generate baud_o, otherwise, div changing can not affect baud_o. baud_o keep low until load isn't high. If you needn't this port, you should connect it with GND.
rst	In	1	Asynchronous reset, high active
baud_o	Out	1	Baud rate output, optional

Table5. UART Baudrategen Interface Description

Parameter	default	Description
div_port	0	1: use div port to control baud_o. 0: use parameter div_param to generate baud_o. Port load should connect with GND.
div_param	1'b1	If you set div_port to 1, this parameter is unuseful, otherwise it's used to generate baud_o.
div_width	1	Set div or div_param data width. Up to 16 bit.

Table6. UART Baudgen parameter table

Note:

If you set div_port = 1, then the parameter div_param is invalid.

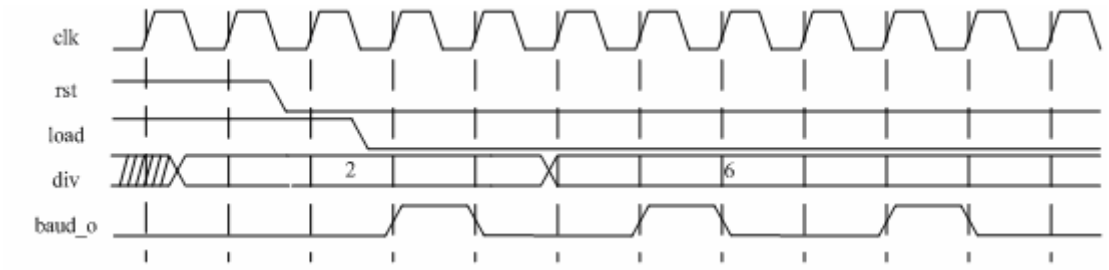

Figure6. Baudgen timing diagram

Table 7 gives an example of divisor values for desired baud rates when using an 18.432 MHz reference clock.

Baud Rate	Decimal (div+1) value	Percent Error(%)
50	23040	0
75	15360	0
110	10473	-0.2865
134.5	8565	0.0876
150	7680	0
300	3840	0
600	1920	0
1200	920	4.3478
1800	640	0
2000	576	0
2400	480	0
3600	320	0
4800	240	0
7200	160	0
9600	120	0
19200	60	0
38400	30	0

56000	21	-2.0408
115200	10	0

Table7 · Baud Rates and Divisor Values for 18.432 MHz Reference Clock

The next paragraph is the verilog instantiation model:

```
module my_uart_baudgen(
        clk,            //baudgen main clock input
        rst,            //baudgen asynchronous reset input, high active
        load,           //baudgen division load signal, high active
        div,            //baudgen division input
        baud_o          //baudgen baud rate output
        );
parameter div_width     = 1;        //division port or parameter width
parameter div_port      = 0;        //0: port div control baud_o; 1: div_param control baud_o
parameter div_param     = 1'b1;     //division parameter, active if div_port=0
input                   clk;
input                   rst;
input                   load;
input   [div_width-1:0] div;
output                  baud_o;
uart_baudgen my_uart_baudgen(
                .clk(clk),
                .rst(rst),
                .load(load),
                .div(div),
                .baud_o(baud_o)
                );
defparam my_uart_baudgen.div_width = div_width;
defparam my_uart_baudgen.div_port = div_port;
defparam my_uart_baudgen.div_param = div_param;
endmodule
```

UART Simulation Model

The Agate Logic provide the behavioral model to customers for simulation. The behavioral models are considered to be zero-delay models. The behavioral models are functionally correct, and will represent the behavioral of the configured UART.

UART Resource Utilization and Performance

Performance and resource utilization for a UART varies depending on the configuration and features selected when customizing the core. The tables below provide UART default configurations and the maximum performance and resources required.

Functions	Family	Performance (MHz)	Resources (LUTs)
Transmitter	Angelo	31.742	273
Receiver		19.758	744
Baudgen		378.358	4

Table8 · UART Performance and Resources

About Agate Logic

Agate Logic is the global pioneer and leader of the innovative Adaptable Programmable Gate Array (APGA) technologies. The company offers a full spectrum of programmable logic devices, software design tools, intellectual property (IP) and design services. Focusing on multiple applications such as telecommunication equipments, industrial control systems and consumer products, we use the Chinese leading foundry partner, SMIC, to manufacture our chips to offer solutions tailored for the market in China.

Technical Support Assistance

Tel: +86 10 82150100

E-mail: support@agatelogic.com

Website: www.agatelogic.com.cn