Documentation

SERCOM – USART/RS232

Samuel Kranz, Simon Fink

Table of contents

Table of figures

List of literature

[1] Atmel SAM D21 Datasheet: Atmel-42181-SAM-D21_Datasheet.pdf (moodle version)

Serial Communication Interface

The SAM D21 has four SERCOM units, which can be configured either as USART, I2C or SPI interface.

USART

The universal synchronous and asynchronous receiver and transmitter (USART) is one of the available modes in the Serial Communication Interface (SERCOM).

Features

- Full-duplex operation
- Asynchronous (with clock reconstruction) or synchronous operation
- Internal or external clock source for asynchronous and synchronous operation
- Baud-rate generator
- Supports serial frames with 5, 6, 7, 8 or 9 data bits and 1 or 2 stop bits
- Odd or even parity generation and parity check
- Selectable LSB- or MSB-first data transfer
- Buffer overflow and frame error detection
- Noise filtering, including false start-bit detection and digital low-pass filter
- Collision detection
- Can operate in all sleep modes
- Operation at speeds up to half the system clock for internally generated clocks
- Operation at speeds up to the system clock for externally generated clocks
- RTS and CTS flow control
- IrDA modulation and demodulation up to 115.2 kbps
- LIN slave support
- Auto-baud and break character detection
- Start-of-frame detection
- Can be used with DMA

Block Diagram

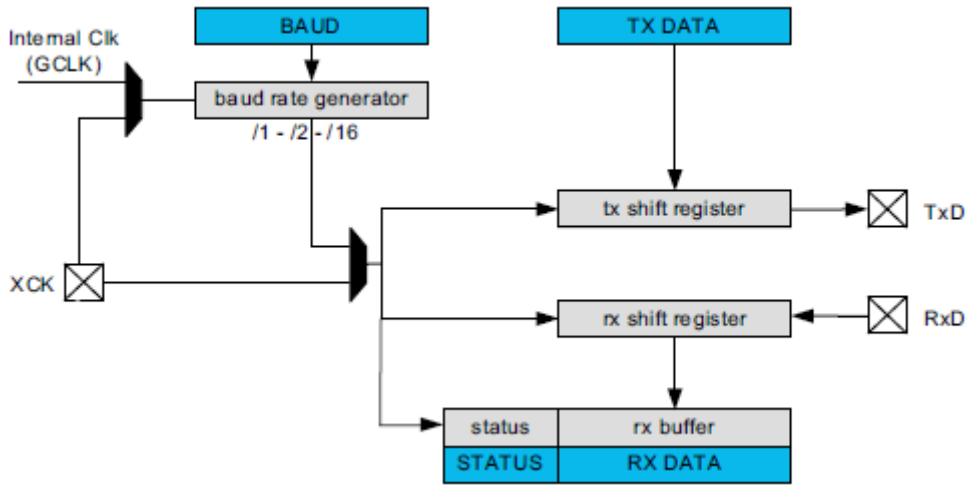

Figure 1: Block Diagram

Functional Description

Principle of Operation

The USART uses three communication lines for data transfer:

- RxD for receiving
- TxD for transmitting
- XCK for the transmission clock in synchronous operation

USART data transfer is frame based, where a serial frame consists of:

- 1 start bit
- 5, 6, 7, 8 or 9 data bits
- MSB or LSB first
- No, even or odd parity bit
- 1 or 2 stop bits

A frame starts with the start bit followed by one character of data bits. If enabled, the parity bit is inserted after the data bits and before the first stop bit. One frame can be directly followed by a new frame, or the communication line can return to the idle (high) state. Figure 2 illustrates the possible frame formats. Bits inside brackets are optional.

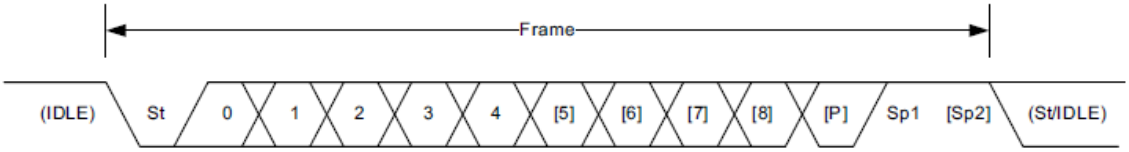

Figure 2: Frame Formats

St Start bit; always low
(n) Data bits; 0 to 8
P Parity bit; odd or even
Sp Stop bit; always high
IDLE No transfers on the communication line; always high in this state

Basic Operation

Initialization

Refer to the following code example.

Enabling, Disabling and Resetting

These states can be achieved by setting the appropriate setting in the CTRLA control register.

Clock Generation and Selection

For both synchronous and asynchronous modes, the clock used for shifting and sampling data can be generated internally by the SERCOM baud-rate generator or supplied externally through the XCK line. These configurations are set in CTRLA.

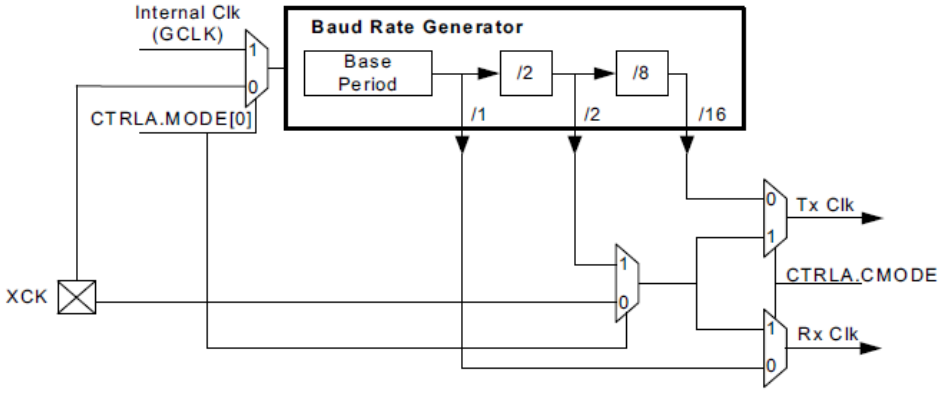

Figure 3: Clock Generation and Selection

Baudrate Calculation

The baud-rate generator, as shown in Figure 3, is used for internal clock generation for asynchronous and synchronous communication. The generated output frequency (f_{BAUD}) is determined by the Baud register (BAUD) setting and the baud reference frequency (f_{REF}). The baud reference clock is the serial engine clock, and it can be internal or external.
For asynchronous operation, the /16 (divide-by-16) output is used when transmitting and the /1 (divide-by-1) output is used when receiving. For synchronous operation the /2 (divide-by-2) output is used. This functionality is automatically configured, depending on the selected operating mode.

Operating Mode	Condition	Baud Rate (Bits Per Second)	BAUD Register Value Calculation
Asynchronous Arithmetic	$f_{BAUD} \le \frac{f_{REF}}{S}$	$f_{BAUD} = \frac{f_{REF}}{S}(1 - BAUD/65{,}536)$	$BAUD = 65{,}536\left(1 - S\frac{f_{BAUD}}{f_{REF}}\right)$
Asynchronous Fractional	$f_{BAUD} \le \frac{f_{REF}}{S}$	$f_{BAUD} = \frac{f_{REF}}{S(BAUD + (FP/8))}$	$BAUD = \frac{f_{REF}}{S \times f_{BAUD}} - \frac{FP}{8}$
Synchronous	$f_{BAUD} \le \frac{f_{REF}}{2}$	$f_{BAUD} = \frac{f_{REF}}{2(BAUD+1)}$	$BAUD = \frac{f_{REF}}{2 f_{BAUD}} - 1$

S – Number of samples per bit. Can be 16, 8, or 3.

The Asynchronous Fractional option is used for auto-baud detection.

Figure 4: Baudrate Calculation Table

Data Register

The USART Transmit Data register (TxDATA) and USART Receive Data register (RxDATA) share the same I/O address, referred to as the Data register (DATA). Writing the DATA register will update the Transmit Data register. Reading the DATA register will return the contents of the Receive Data register.

Data Transmission

A data transmission is initiated by loading the DATA register with the data to be sent. The data in TxDATA is moved to the shift register when the shift register is empty and ready to send a new frame. When the shift register is loaded with data, one complete frame will be transmitted.

Data Reception

The receiver starts data reception when a valid start bit is detected. Each bit that follows the start bit will be sampled at the baud rate or XCK clock, and shifted into the receive shift register until the first stop bit of a frame is received. When the first stop bit is received and a complete serial frame is present in the receive shift register, the contents of the shift register will be moved into the two-level receive buffer. The Receive Complete interrupt flag in the Interrupt Flag Status and Clear register (INTFLAG.RXC) is set, and the optional interrupt is generated. A second stop bit will be ignored by the receiver.

The received data can be read by reading the DATA register. DATA should not be read unless the Receive Complete interrupt flag is set.

Additional Features

Parity

Even or odd parity can be selected for error checking by writing 0x1 to the Frame Format bit group in the Control A register (CTRLA.FORM). If even parity is selected by writing a zero to the Parity Mode bit in the Control B register (CTRLB.PMODE), the parity bit of the outgoing frame is set to one if the number of data bits that are one is odd (making the total number of ones even).

Hardware Handshaking

The USART features an out-of-band hardware handshaking flow control mechanism, implemented by connecting the RTS and CTS pins with the remote device.

Hardware handshaking is only available with the following configuration:

- USART with internal clock (CTRLA.MODE = 1).
- Asynchronous mode (CTRLA.CMODE = 0).
- Flow control pinout (CTRLA.TXPO = 2).

The receiver drives its RTS pin high when disabled, or when the receive FIFO is full. This indicates to the remote device that it must stop transmitting after the ongoing transmission is complete. Enabling and disabling the receiver by writing RXEN will set/clear the RTS pin after a synchronization delay. When the receive FIFO goes full, RTS is immediately set and the frame that is currently being received will be stored in the shift register until the receive FIFO is no longer full.

The current CTS level is available in the STATUS register (STATUS.CTS). Character transmission will only start if CTS is low. When CTS goes high, the transmitter will stop transmitting after the ongoing transmission is complete.

Interrupts

The USART has the following interrupt sources:

- Error (ERROR): this is an asynchronous interrupt and can be used to wake-up the device from any sleep mode.
- Received Break (RXBRK): this is an asynchronous interrupt and can be used to wake-up the device from any sleep mode.
- Clear to Send Input Change (CTSIC): this is an asynchronous interrupt and can be used to wake-up the device from any sleep mode.
- Receive Start (RXS): this is an asynchronous interrupt and can be used to wake-up the device from any sleep mode.
- Receive Complete (RXC): this is an asynchronous interrupt and can be used to wake-up the device from any sleep mode.
- Transmit Complete (TXC): this is an asynchronous interrupt and can be used to wake-up the device from any sleep mode.
- Data Register Empty (DRE): this is an asynchronous interrupt and can be used to wake-up the device from any sleep mode.

Each interrupt source has an interrupt flag associated with it. The interrupt flag in the Interrupt Flag Status and Clear register (INTFLAG) is set when the interrupt condition occurs. Each interrupt can be individually enabled by writing a one to the corresponding bit in the Interrupt Enable Set register (INTENSET), and disabled by writing a one to the corresponding bit in the Interrupt Enable Clear register (INTENCLR). An interrupt request is generated when the interrupt flag is set and the corresponding interrupt is enabled. The interrupt request remains active until the interrupt flag is cleared, the interrupt is disabled or the USART is reset. See the register description for details on how to clear interrupt flags. The USART has one common interrupt request line for all the interrupt sources. The user must read INTFLAG to determine which interrupt condition is present.

Note that interrupts must be globally enabled for interrupt requests to be generated.

Register Description

In this paragraph are shown the register settings for a basic 8N1 (8 data bits, no parity, 1 stop bit, 9600 baud) operation.

Register - Control A (CTRLA)

Bit	Name	Value	Remark
31	res.	-	-
30	data order	0x1	LSB first
29	clock polarity	0x0	TxD Change - Rising XCK edge, RxD Sample - Falling XCK edge
28	communication mode	0x0	asynchronous communication
27:24	frame format	0x00	USART frame
23:22	sample adjustment	0x0	16x Over-sampling: 7-8-9, 8x Over-sampling: 3-4-5
21:20	receive data pinout	0x3	RxD Pad[3]
19:18	res.	-	-
17:16	transmit data pinout	0x1	TxD pin: Pad[2], XCK Pin: Pad[3]
15:13	sample rate	0x0	16x over-sampling using arithmetic baud rate generation
12:9	res.	-	-
8	immediate buffer overflow notification	0x0	STATUS.BUFOVF is asserted when it occurs in the data stream
7	run in standby	0x0	XCK disconnected when transfer is finished, ICK disabled when ongoing transfer is finished, wake up on receive start or transfer complete interrupt.
6:5	res.	-	-
4:2	operating mode	0x1	USART with internal clock
1	enable	0x1	the peripheral is enabled
0	software reset	0x0	no reset operation ongoing

CTRLA set code: 0b0100 0000 0011 0001 0000 0000 0000 0110 = 0x40310006

Register - Control B (CTRLB)

Bit	Name	Value	Remark
31:18	res.	-	-
17	receiver enable	0x1	receiver is enabled when the usart is enabled
16	transmitter enable	0x1	transmitter is enabled when the usart is enabled
15:14	res.	-	-
13	parity mode	0x0	even parity
12:11	res.	-	-
10	encoding format	0x0	data is not encoded
9	SOF detection enable	0x0	disabled
8	collision detection enable	0x0	disabled
7	res.	-	-
6	stop bit mode	0x0	one stop bit
5:3	res.	-	-
2:0	character size	0x0	8 bits

CTRLB set code: 0b0000 0000 0000 0011 0000 0000 0000 0000 = 0x00030000

Register – Baudrate (BAUD)

Operating Mode	Condition	Baud Rate (Bits Per Second)	BAUD Register Value Calculation
Asynchronous Arithmetic	$f_{BAUD} \leq \frac{f_{REF}}{S}$	$f_{BAUD} = \frac{f_{REF}}{S}(1 - BAUD / 65{,}536)$	$BAUD = 65{,}536\left(1 - S\frac{f_{BAUD}}{f_{REF}}\right)$

S	=	number of samples per bit:	16x Oversampling	CTRLA [15:13]
f_{REF}	=	ref. clock:	internal 48 MHz clock	CTRLA [4:2]
f_{BAUD}	=	baudrate:	9600Hz	

$$BAUD = \left\lfloor 65'536 * \left(1 - 16 * \frac{9600\ Hz}{48\ MHz}\right)\right\rfloor = \lfloor 65326.285 \rfloor = 65326 = 0\text{xFF2E}$$

Code Example

```
//------------------------
//uart-demo
//(c) H.Buchmann FHWN 2016
//------------------------
IMPLEMENTATION(app,$Id$)
#include "sys/msg.h"
#include "uart-poll.h"

class App
{
  static App app;
  static const UART::Config Config;
 UART uart;
  App();
};

App App::app;
const UART::Config App::Config={
   sys::SOC::SERCOM::COM0,
   //PAD                    PIN              FUNC
   {sys::reg::SER::PAD::P3,{sys::reg::PORT::PA11,sys::reg::PORT::F_C}},//RX
   {sys::reg::SER::PAD::P2,{sys::reg::PORT::PA10,sys::reg::PORT::F_C}} //TX
                                    };
App::App()
:uart(Config)
{
  sys::msg<<"UART Demo\n";
  while(true)
   {
     //simple echo
     uart.out(uart.in());
   }
}
```

```
//---------------------
//uart-poll
//(c) H.Buchmann FHNW 2016
//---------------------
#include "sys/reg/sercom.h"
IMPLEMENTATION(sys_uart_poll,$Id$)
#include "uart-poll.h"
#include "sys/reg/port.h"
#include "sys/reg/gclk.h"
#include "sys/reg/pm.h"
#include "sys/msg.h"

UART::UART(const Config& cfg)
:usart(sys::SOC::SERCOM[cfg.id].com.USART)
{
  sys::SOC::clkOn(sys::SOC::SERCOM[cfg.id].pid);
  sys::reg::GCLK.gen2dev(sys::reg::GCLK::G0,sys::SOC::SERCOM[cfg.id].clk);
  sys::reg::SER::mux(cfg.rx);
  sys::reg::SER::mux(cfg.tx);

  usart.CTRLA|=1;
  while( (usart.CTRLA&1) || (usart.SYNCBUSY&1))
   { }
  usart.BAUD =0x0000ff2e;
  usart.CTRLA=0x40310006;
  usart.CTRLB=0x00030000;
  while(usart.SYNCBUSY&(1<<1)){}
}

void UART::out(unsigned char ch)
{
 while((usart.INTFLAG & (1<<0))==0)
 { }
 usart.DATA=ch;
}

unsigned char UART::in()
{
 while((usart.INTFLAG & (1<<2))==0)
 { }
  return usart.DATA;
}

bool UART::avail()
{  return (usart.INTFLAG & (1<<2))!=0; }
```

Measurements

Example 1: 0XAA (8N1)

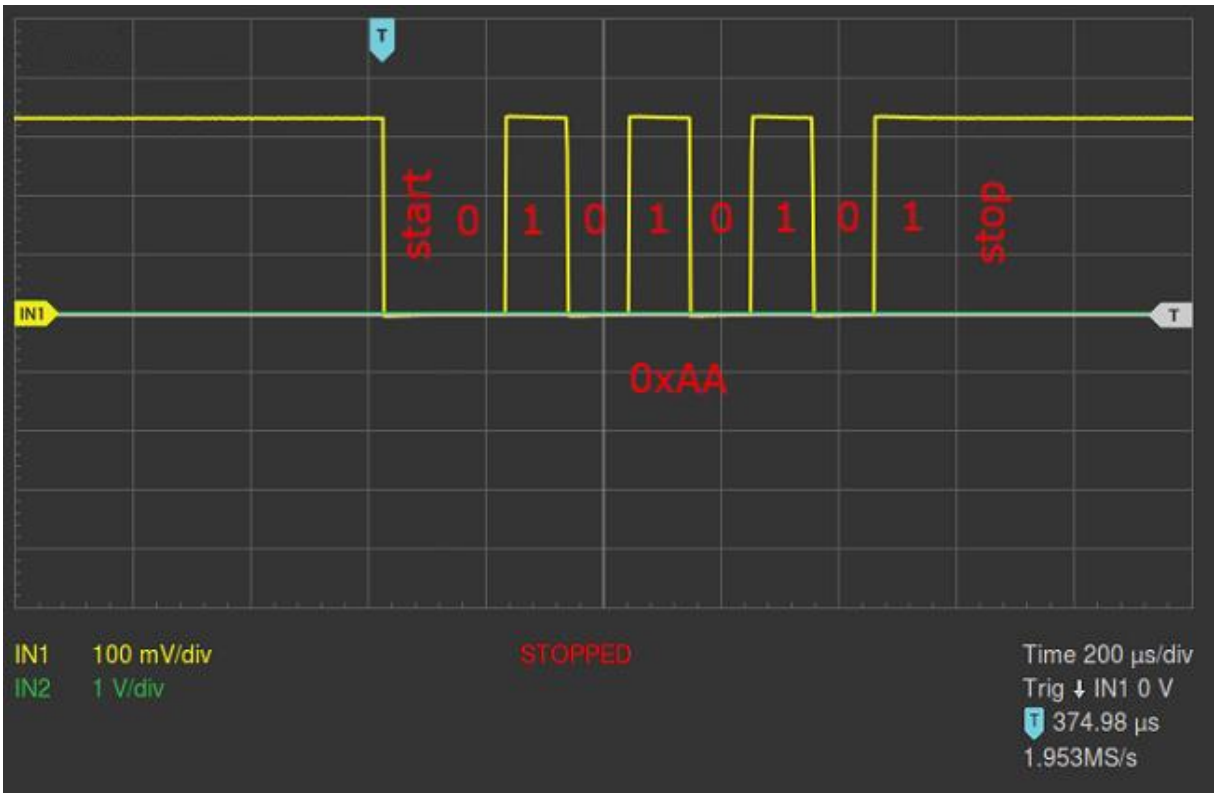

Figure 5: Measurement 8N1 - 0xAA

Example 2: 0xA5 (8N1)

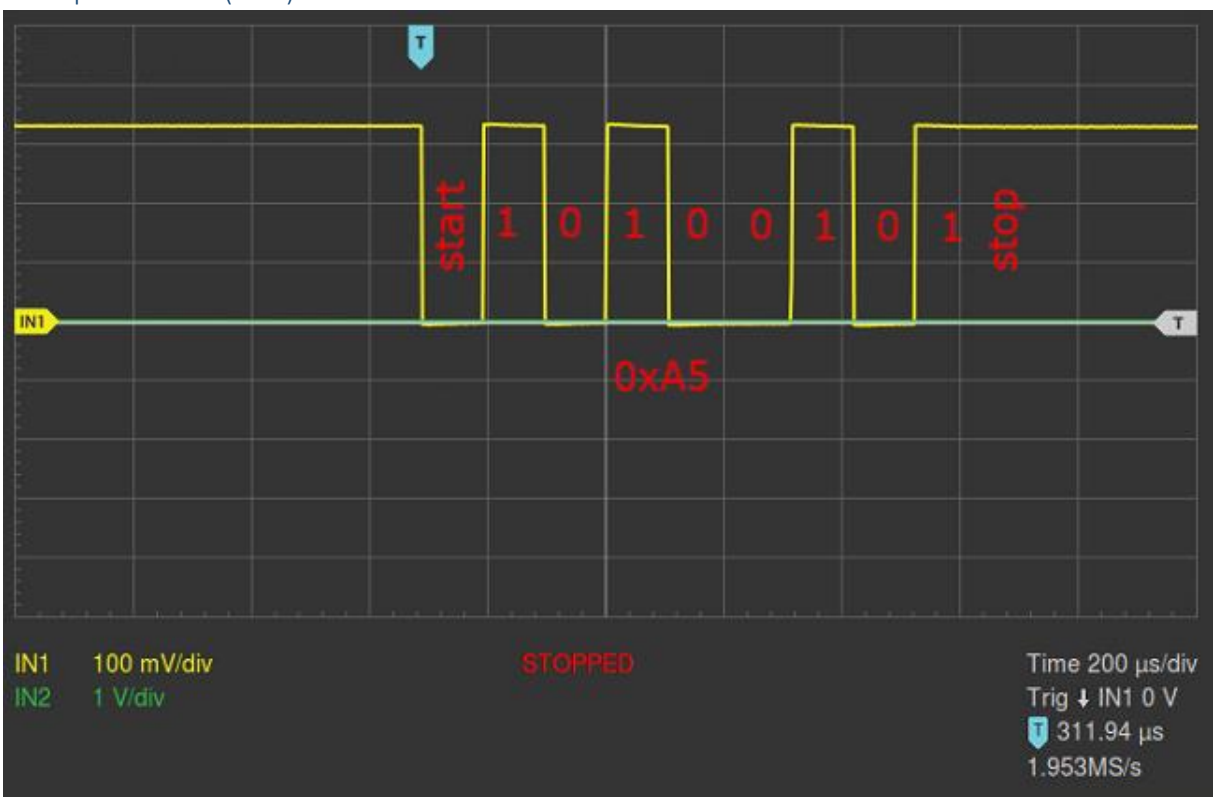

Figure 6: Measurement 8N1 - 0xA5