Programming Serial Ports

for Basics of Microprocessor technology

Nikolay Arsenov, Alexey Tukalo,
EFA12SF,
Information Technology,
Savonia University of Applied Sciences

December 20, 2014

Contents

1 Utilizing the serial communication

Serial communication in such fields like telecommunication and computer science, is the process of sending data one unit per a time over a communication channel, bus or cable. In embedded systems, the AVR microcontrollers, serial communication also founded in UART to send digital signal from the microcontroller to a PC or otherwise, depends on what is our target. UART is a Universal Asynchronous Receiver/Transmitter. Transmitter (TX) is a device which sends data and Receiver (RX) gets the data from the transmitter. Transmitter and Receiver have flags to detect errors while transmission or receiving the frame, the same we can see in Network, for example in TCP/IP protocols, where each frame with our data transfers to the another device and in the frame there is also information about the state of connection, is there errors or loss and etc. If yes, send again, else, everything is good, wait a new packet with a frame.

2 The features of USART circuit

- Full Duplex Operation (Independent Serial Receive and Transmit Registers)
- Asynchronous or Synchronous Operation
- Master or Slave Clocked Synchronous Operation
- High Resolution 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 Supported by Hardware
- Data OverRun Detection
- Framing Error Detection
- Noise Filtering Includes False Start Bit Detection and Digital Low Pass Filter
- Three Separate Interrupts on TX Complete, TX Data Register Empty and RX Complete
- Multi-processor Communication Mode
- Double Speed Asynchronous Communication Mode

3 Programming the transmission speed

There are two different ways to set the necessary speed to the microcontroller, the first one is to calculate which value we should set to the UBRRnL and/or if we need UBRRnH. Or another way is to use the already calculated values from the Boud Rate settings Figure. We just need to know the frequency of our device, for example 16MHertz and what kind of speed (symbols per second or frames per second) we need.

For example, if we want to set our microcontroller speed equal to 9600 we can write this three lines:

```
case 9600:UBRR0L=103; //speed 9600BD
                      UBRR0H=0;
```

For starting communication between transmitter and receiver, we have to set necessary flags to zero, set the Stop bit, Parity bit and Data Bits Format. Then enable transmitter and receiver:

UCSR0B=(1<<TXEN0) ∨ (1<<RXEN0);//enable transmitter and receiver.

Baud Rate [bps]	f_{osc} = 16.0000MHz			
	U2Xn = 0		U2Xn = 1	
	UBRR	Error	UBRR	Error
2400	416	-0.1%	832	0.0%
4800	207	0.2%	416	-0.1%
9600	103	0.2%	207	0.2%
14.4K	68	0.6%	138	-0.1%
19.2K	51	0.2%	103	0.2%
28.8K	34	-0.8%	68	0.6%
38.4K	25	0.2%	51	0.2%
57.6K	16	2.1%	34	-0.8%
76.8K	12	0.2%	25	0.2%
115.2K	8	-3.5%	16	2.1%
230.4K	3	8.5%	8	-3.5%
250K	3	0.0%	7	0.0%
0.5M	1	0.0%	3	0.0%
1M	0	0.0%	1	0.0%

Figure 1: Boud Rate calculations

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRR Value
Asynchronous Normal mode (U2Xn = 0)	$BAUD = \frac{f_{OSC}}{16(UBRRn+1)}$	$UBRRn = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed mode (U2Xn = 1)	$BAUD = \frac{f_{OSC}}{8(UBRRn+1)}$	$UBRRn = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master mode	$BAUD = \frac{f_{OSC}}{2(UBRRn+1)}$	$UBRRn = \frac{f_{OSC}}{2BAUD} - 1$

Note: 1. The baud rate is defined to be the transfer rate in bit per second (bps).

BAUD Baud rate (in bits per second, bps).

f_{OSC} System Oscillator clock frequency.

UBRRn Contents of the UBRRHn and UBRRLn Registers, (0-4095).

Figure 2: Boud Rate settings

4 Programming the other serial parameters

There are three main linear parameters of the frame format:

- Stop bit
- Number of data bits
- Parity settings

4.1 Stop bit

Stop bits are used to separate different frames, it can contain one or two bits. The settings should be made for transmitter, the receiver ignores it. The stop bits are controlled by USBSn register in according with table 3.

USBSn	Stop Bit(s)
0	1-bit
1	2-bit

Figure 3: USBS Bit Settings

4.2 Number of data bits

The number of data bits on the frame is controlled by UCSZn registers bits from 0 to 2. The settings have to be performed by table 4.

UCSZn2	UCSZn1	UCSZn0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

Figure 4: UCSZn Bits Setting

4.3 Parity settings

Parity is some kind of error detection technique which is used in USART. These bits enable and set type of parity generation and check. If enabled, the Transmitter will automatically generate and send the parity of the transmitted data bits within each frame. The Receiver will generate a parity value for the incoming data and compare it to the UPMn setting. If a mismatch is detected, the UPEn Flag in UCSRnA will be set.

UPMn1	UPMn0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

Figure 5: UPMn Bits Settings

5 The inspection of the data transmission

The session between Transmitter and Receiver happens by sending frames one by one (frame is a little piece of data). When the first portion taken, then sent and gotten by Receiver, UART starts new session with the same algorithms.

The Transmitter has two very important flags that indicate its state: USART Data Register Empty (UDREn) and Transmit Complete (TXCn). The Data Register Empty checks if the transmit buffer is ready to receive new data (empty or cleared), this bit sets and UCSRnA Register bit to zero. Then we enable Data Register Empty Interrupt Enable (UDRIEn) bit in UCSRnB, after that which will be executed. The Transmit Complete (TXCn) Flag bit is one, when entire frame in the Transmit Shift Register has been shifted out and there are no data in the buffer. The TXCn Flag bit is automatically cleared when transmit complete interrupt executed.

The Receiver has only one flag that indicated the Receiver state the Receiver Complete (RXCn) Flag checks if there are unread data in the receiver buffer or not. If it exists flag is one, and zero if the receiver buffer is empty. When the RXCn is set, there will be the Receive Complete Interrupt Enable (RXCIEn) in UCSRnB is set, the USART Receive Complete interrupt will be executed.

Errors checks on the USART Receiver, there are three Error Flags: Frame Error (FEn), Data OverRun (DORn) and Parity Error (UPEn). All can be accessed by reading UCSRnA. All the flags located in the receive buffer together with the frame, which they indicate the error status. All flags must be set to zero when the UCSRnA is written for upward compatibility of future USART implementations. None of the Error Flags can generate interrupts.

The Frame Error (FEn) Flag indicates the state of the first stop bit of the next readable frame stored in the receive buffer. The FEn Flag is zero when the stop bit was correctly read (as one), and else if it was incorrect (zero). This flag can be used for detecting out-of-sync conditions, detecting break conditions and protocol handling. The FEn Flag is not affected by the setting of the USBSn bit in UCSRnC since the Receiver ignores all, except for the first, stop bits.

The Data OverRun (DORn) Flag indicates data loss due to a receiver buffer full condition. A Data OverRun occurs when the receive buffer is full (two characters), it is a new character waiting in the Receive Shift Register, and a new start bit is detected. If the DORn Flag is set there was one or more serial frame lost between the frame last read from UDRn, and the next frame read from UDRn. For compatibility with future devices, always write this bit to zero when writing to UCSRnA. The DORn Flag is cleared when the frame received was successfully moved from the Shift Register to the receive buffer.

The Parity Error (UPEn) Flag indicates that the next frame in the receive buffer had a Parity Error when received. If Parity Check is not enabled the UPEn bit will always be read zero. For compatibility with future devices, always set this bit to zero when writing to UCSRnA.

6 Programming Tasks

6.1 The tasks from first to third

Listing 1: L4_123.c

```
#include "L4_123.h"
#include <asf.h>
#include <stdlib.h>

void SetLineParameters( unsigned port, unsigned speed, unsigned parity, unsigned cDataBits
{

        if ( port == 0)
        {
                        /* Set baud rate for 16MHz */
                switch (speed)
                {
                        case 2400:
                        UBRR0L = 160; // speed 2400 BD
                        UBRR0H = 1; // = 256 +
                                UCSR0A = (1<<U2X0);
                        break;

                        case 4800:
                        UBRR0L = 207; // speed 4800 BD
                        UBRR0H = 0; //
                        break;

                        case 9600:
                        UBRR0L = 103; // speed 9600 BD
                        UBRR0H = 0; //
                        UCSR0A = (0<<U2X0);
                        break;

                        case 19200:
                        UBRR0L = 51; // speed 19200 BD
                        UBRR0H = 0; //
                }
                        //-----------------------------------


                        /* Set Parity */
                        switch(parity){
                        case OFF_PARITY : UCSR0C |=( 0 << UPM01) | ( 0 << UPM00);
                                break;
                        case ODD_PARITY : UCSR0C |=( 1 << UPM01) | ( 1 << UPM00);
                                break;
                        case EVEN_PARITY : UCSR0C |=( 1 << UPM01) | ( 0 << UPM00);
                        }
                        //-----------------------------------


                                /* Data Bits Format */
                                if(cDataBits==BIT_FORMAT_8bit)
```

```
                        //--------------------------------------

                        /* Stop Bit */
                        if(cStopBits==STOP_BIT_2bit)
                         UCSR0C = (1<<USBS0)|(3<<UCSZ00);

                        //--------------------------------------


                        /*Set transmitter and receiver in UCSR0B port*/
                        UCSR0B = ( 1 << TXEN0) | ( 1 << RXEN0);
// enable transmitter and receiver
        } else
        if ( port == 1)
        {
                        /* Set baud rate for 16MHz */
                switch (speed)
                {
                        case 2400:
                        UBRR1L = 160; // speed 2400 BD
                        UBRR1H = 1; // = 256 +
                                UCSR1A = (1<<U2X1);
                        break;

                        case 4800:
                        UBRR1L = 207; // speed 4800 BD
                        UBRR1H = 0; //
                        break;

                        case 9600:
                        UBRR1L = 103; // speed 9600 BD
                        UBRR1H = 0; //
                        UCSR1A = (0<<U2X1);
                        break;

                        case 19200:
                        UBRR1L = 51; // speed 19200 BD
                        UBRR1H = 0; //
                }
                        //--------------------------------------


                        /* Set Parity */
                        switch(parity){
                        case OFF_PARITY : UCSR1C |=( 0 << UPM11) | ( 0 << UPM10);
                                break;
                        case ODD_PARITY : UCSR1C |=( 1 << UPM11) | ( 1 << UPM10);
                                break;
                        case EVEN_PARITY : UCSR1C |=( 1 << UPM11) | ( 0 << UPM10);
                        }
                        //--------------------------------------
```

```
                                /* Data Bits Format */
                                if(cDataBits==BIT_FORMAT_8bit)

                        //---------------------------------

                        /* Stop Bit */
                        if(cStopBits==STOP_BIT_2bit)
                         UCSR1C = (1<<USBS1)|(3<<UCSZ10);

                        //---------------------------------


                        /*Set transmitter and receiver in UCSR0B port*/
                        UCSR1B = ( 1 << TXEN1) | ( 1 << RXEN1);
// enable transmitter and receiver
        }
}
void WriteCharacter(unsigned port,unsigned char character)
{
        if(port==0)
        {
                while ( !( UCSR0A & (1<<UDRE0)) );
                UDR0=character;
        } else
                if(port==1)
        {
                while ( !( UCSR1A & (1<<UDRE1)) );
                UDR1=character;
        } else
                if(port==2)
        {
                while ( !( UCSR2A & (1<<UDRE2)) );
                UDR2=character;
        }
}
int CharacterReady(unsigned port)
{
        if(potr==0)
                if ((UCSR0A & (1 << RXC0)))
                return 1;
                        else
                return 0;
        else if(port==1)
                if ((UCSR1A & (1 << RXC1)))
                return 1;
                        else
                return 0;
        else if(port==2)
                if ((UCSR2A & (1 << RXC2)))
                return 1;
                        else
                return 0;
}
unsigned char ReadCharacter(unsigned port)
```

```
{
        int a;
        if(port==0){
                while ( !(UCSR0A & (1<<RXC0)) );
                a=(unsigned char) UDR0;
        } else
        if(port==1){
                while ( !(UCSR1A & (1<<RXC1)) );
                a=(unsigned char) UDR1;
        } else
        if(port==2){
                while ( !(UCSR2A & (1<<RXC2)) );
                a=(unsigned char) UDR2;
        }
        return a; // return newly come character
}
```

Listing 2: L4_123.h

```
#ifndef L4_123_H_
#define L4_123_H_

#define F_CPU 16000000

//Parity 0
#define OFF_PARITY 0
#define ODD_PARITY 1
#define EVEN_PARITY 2
#define BIT_FORMAT_8bit 1
//StopBit
#define STOP_BIT_1bit 1
#define STOP_BIT_2bit 2

void SetLineParameters( unsigned port, unsigned speed, unsigned parity,
unsigned cDataBits, unsigned cStopBits );
void WriteCharacter( unsigned port, unsigned char character);
int CharacterReady( unsigned port);
unsigned char ReadCharacter( unsigned port);
int WriteString(unsigned port, char *szOutput);
char* ReadString( unsigned port);

#endif
```

6.2 The fourth task

Listing 3: L4_4.c

```
#include "L4_123.h"
#include "L4_4.h"
#include <asf.h>
#include <stdlib.h>

int WriteString(unsigned port, char *szOutput)
{
        do{
                WriteCharacter(port,*(szOutput++));
```

```
        }while(*szOutput!='\0');
        return 0;
}
char* ReadString( unsigned port)
{
        static char buffer[80] = {0};
        int i=0;
        do{
                buffer[i]=ReadCharacter(port);
                if(buffer[i] < 15)
                return buffer;
                else
                i++;

        }while(1);

}
```