AVR 305: Half Duplex Compact Software UART

8-Bit Microcontroller

Application Note

Features

- **32 Words of Code, Only**
- **Handles Baud Rates of up to 38.4 kbps with a 1 MHz XTAL**
- **Runs on Any AVR Device, Only Two Port Pins Required**
- **Does Not Use Any Timer**

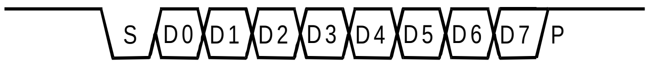
Introduction

In many applications utilizing serial communication, the software handling communication does not have to be performed as a background task. This application note describes how to implement a polled software UART capable of handling speeds up to 614 400 bps on an AT90S1200. This implementation is intended for applications where small code is preferred. All bit delays are software delays, so no timer is required. The controller can not perform any other tasks while receiving or transmitting, but when the operation is complete, the controller is free to continue program execution.

Theory of Operation

In asynchronous serial communication, no clock information is transferred. The data is transferred serially, one bit at a time. In idle state, the line is held at logical '1'. When data is to be transferred, the first bit is a so called start bit, often given the symbol S. The start bit is a '0', causing a '1' to '0' transition at the line. This can be detected by the receiver, signaling that data is coming. The following bits transferred are the data bits, with the LSB first. Then, one or more stop bits (P) are transferred. The stop bit is a logical '1', putting the line back to idle state before a new start bit followed by a data byte can be transferred. There must be at least one stop bit, so that a '1' to '0' transition can be detected by the receiver, indicating a new start bit. The *frame* shown in Figure 1 has got eight data bits and one stop bit. Sometimes, a parity bit is included between the last data bit and the stop bit, and there can be several stop bits.

Figure 1. Frame Format



The receiver must sample the data line in the middle of every bit in order to receive the data properly. The bit length has to be equal for all bits, so that the receiver knows when to sample the line. The receiver is synchronized with the transmitter by the falling edge of the start bit. The receiver must have an internal timer in order to sample the bits at the right time.

The bit length must be the same for both transmitter and receiver, and some standard speeds are defined in bits per second, *bps*.

0952A-A–9/97

Implementation

Bit Length Delays - *UART_delay*

The delay between the bits are generated by calling a delay subroutine twice (as the subroutine generates a half-bit delay, see receiving data). If very short delays are required (when transmitting and receiving at very high speed), the delay must be implemented inside the *putchar* and *getchar* routines. The required delay length in clock cycles can be calculated using the following formula:

$$c = \frac{f_{CLCL}}{\text{bit rate}}$$

where c is the bit length in clock cycles and f_{CLCL} is the crystal frequency.

Both *putchar* and *getchar* use 9 CPU cycles to send or receive a bit. Hence, a delay of c - 9 cycles must be generated between each bit. The *rcall* and *ret* instructions require a total of 7 cycles. If the subroutine is to be called twice to generate the required delay, the delay has to be d cycles:

$$d = \frac{c-9}{2} - 7$$

If the delay is generated as shown below, the total execution time is 3·*b* cycles plus 7 cycles for *rcall* and *ret*.

```
              rcall     UART_delay
UART_delay:   ldi       temp,b
UART_delay1:  dec       temp
              brne      UART_delay1
              ret
```

The value b is found by the equation

$$b = \frac{\frac{c-9}{2} - 7}{3} = \frac{c-23}{6}$$

The actual delay generated, calling *delay* twice is

$$d = (3 \times b + 7) \times 2 + 9 = 6 \times b + 23$$

From this, the minimum and maximum delays are d_{min} = 29 and d_{max} = 1 559 cycles. The c and b values for some bit rates and frequencies are shown in Table 8.

Table 1. "UART_delay" Subroutine Performance Figures

Parameter	Value	
Code Size	4 words	
Execution Cycles	Min: 7 cycles Max: 772 cycles (including *ret*)	
Register Usage	Low registers High registers Global	:None :None :1

Table 2. "UART_delay" Register Usage

Register	Input	Internal	Output
R17		"temp" - count variable	

Transmitting Data - *putchar*

The *putchar* subroutine transmits the byte stored in the register *Txbyte*. The data bits are shifted into the carry bit. The easiest way to generate stop bits is to let the zeros shifted into the transmitted byte be interpreted as ones. If the data byte is inverted before it is shifted, a '0' in carry must give a '1' on the line, and a '0' in carry gives a '1' on the line. When 0's are shifted into the data byte, they are handled as 1's. This way, any number of stop bits can generated by just repeating the transmittal loop. The start bit is generated by setting the carry bit before data are shifted out.

Table 3. "putchar" Subroutine Performance Figures

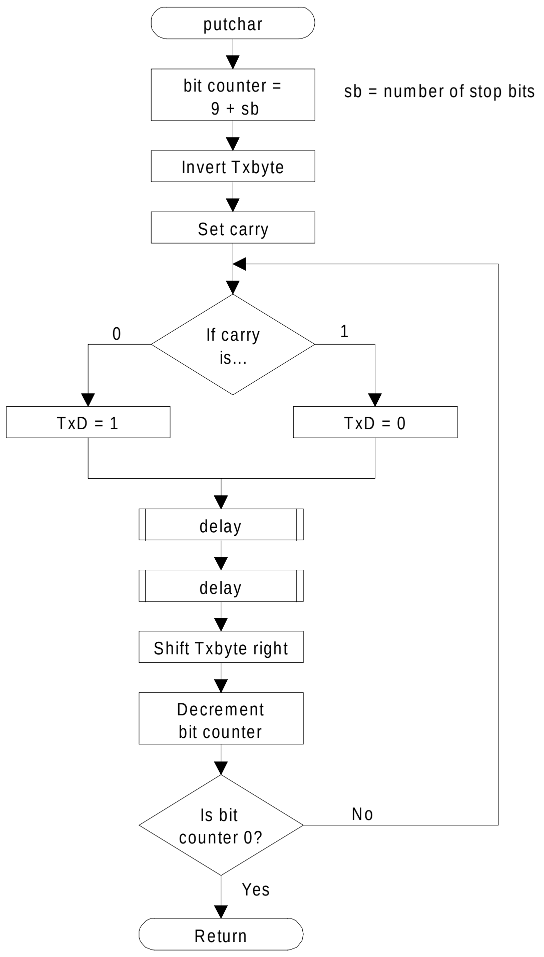
Parameter	Value
Code Size	14 words
Execution Cycles	Depends on bit rate
Register Usage	Low registers :None High registers :None Global :2

Table 4. "putchar" Register Usage

Register	Input	Internal	Output
R16		"bitcnt" - counts the number of bits transfered	
R18	"Txbyte" - the byte to send		

The algorithm for transmitting data is shown in the flow-chart:

Figure 1. *putchar* subroutine



Receiving Data - *getchar*

First, the routine waits for a logical '0' (not a transition). When the start bit is detected, a 1.5 bit delay is generated. This is performed by calling the delay subroutine three times. Sampling then starts at 1 bit intervals. The carry is set or cleared according to the logic value at the RxD-pin. If less than eight data bits are received, the carry is shifted into the receive byte. If not, the routine returns with the received byte in *Rxbyte*.

The routine waits for one bit length after the last data bit and terminates in the middle of the stop bit. This is done to prevent detection of a false startbit if the routine is called again immediately after a complete reception.

Table 5. "getchar" Subroutine Performance Figures

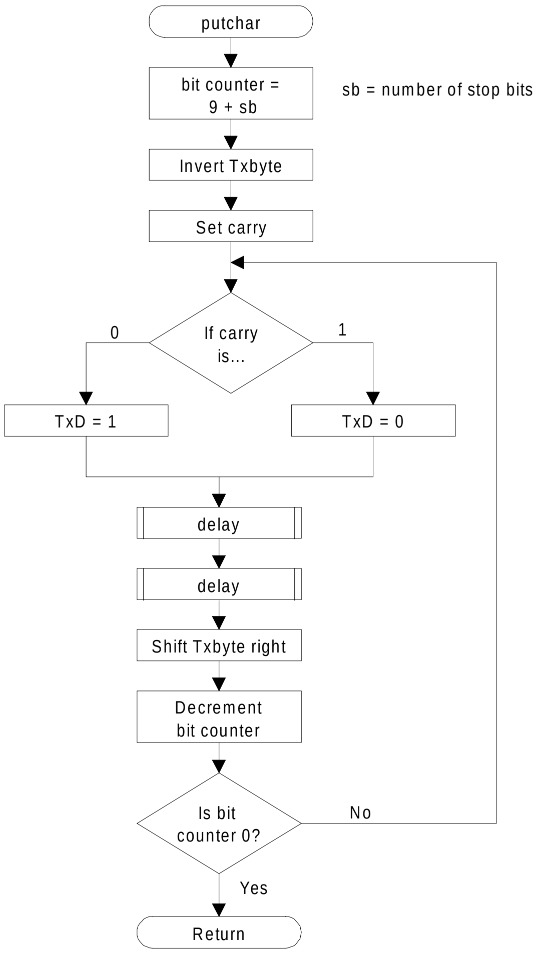
Parameter	Value
Code Size	14 words
Execution Cycles	Waits until byte received
Register Usage	Low registers :None High registers :None Global :2

Table 6. "getchar" Register Usage

Register	Input	Internal	Output
R16		"bitcnt" - counts the number of bits received	
R18			"Rxbyte" - the received byte

The algorithm for receiving data is:

Figure 2. *getchar* subroutine



Example Program

The example program receives a character with *getchar* and echoes it back with *putchar*.

Table 7. Overall Performance Figures

Parameter	Value
Code Size	32 words - UART routines only 40 words - Complete application note
Register Usage	Low registers :None High registers :4 Global :None
Interrupt Usage	None
Peripheral Usage	Port D pin 0 and 1 (any two pins can be used)

Table 8. Baud Rate Table

1 MHz

BaudRate	Cycles required	b-value	Error %
2400	417	66	0.6
4800	208	31	0.3
9600	104	14	2.7
14400	69	8	2.2
19200	52	5	1.8
28800	35	2	0.8
57600	17	1	67.0
115200	9	1	234.1

1.8432 MHz

BaudRate	Cycles required	b-value	Error %
2400	768	124	0.1
4800	384	60	0.3
9600	192	28	0.5
14400	128	18	2.3
19200	96	12	1.0
28800	64	7	1.6
57600	32	2	9.4
115200	16	1	81.3

2 MHz

BaudRate	Cycles required	b-value	Error %
2400	833	135	0.0
4800	417	66	0.6
9600	208	31	0.3
14400	139	19	1.4
19200	104	14	2.7
28800	69	8	2.2
57600	35	2	0.8
115200	17	1	67.0

2.4576 MHz

BaudRate	Cycles required	b-value	Error %
2400	1024	167	0.1
4800	512	82	0.6
9600	256	39	0.4
14400	171	25	1.4
19200	128	18	2.3
28800	85	10	2.7
57600	43	3	3.9
115200	21	1	35.9

3.276 MHz

BaudRate	Cycles required	b-value	Error %
2400	1365	224	0.1
4800	683	110	0.0
9600	341	53	0.1
14400	228	34	0.2
19200	171	25	1.4
28800	114	15	0.7
57600	57	6	3.7
115200	28	1	2.0

3.6864 Mhz

BaudRate	Cycles required	b-value	Error %
2400	1536	252	0.1
4800	768	124	0.1
9600	384	60	0.3
14400	256	39	0.4
19200	192	28	0.5
28800	128	18	2.3
57600	64	7	1.6
115200	32	2	9.4

4 MHz

BaudRate	Cycles required	b-value	Error %
2400	1667	274	0.0
4800	833	135	0.0
9600	417	66	0.6
14400	278	42	1.0
19200	208	31	0.3
28800	139	19	1.4
57600	69	8	2.2
115200	35	2	0.8

4.608 MHz

BaudRate	Cycles required	b-value	Error %
2400	1920	316	0.1
4800	960	156	0.1
9600	480	76	0.2
14400	320	50	0.9
19200	240	36	0.4
28800	160	23	0.6
57600	80	10	3.8
115200	40	3	2.5

7.3728 MHz

BaudRate	Cycles required	b-value	Error %
2400	3072	508	0.0
4800	1536	252	0.1
9600	768	124	0.1
14400	512	82	0.6
19200	384	60	0.3
28800	256	39	0.4
57600	128	18	2.3
115200	64	7	1.6

8 MHz

BaudRate	Cycles required	b-value	Error %
2400	3333	552	0.0
4800	1667	274	0.0
9600	833	135	0.0
14400	556	89	0.3
19200	417	66	0.6
28800	278	42	1.0
57600	139	19	1.4
115200	69	8	2.2

9.216 MHz

BaudRate	Cycles required	b-value	Error %
2400	3840	636	0.0
4800	1920	316	0.1
9600	960	156	0.1
14400	640	103	0.2
19200	480	76	0.2
28800	320	50	0.9
57600	160	23	0.6
115200	80	10	3.8

11.059 MHz

BaudRate	Cycles required	b-value	Error %
2400	4608	764	0.0
4800	2304	380	0.0
9600	1152	188	0.1
14400	768	124	0.1
19200	576	92	0.2
28800	384	60	0.3
57600	192	28	0.5
115200	96	12	1.0

14.746 MHz

BaudRate	Cycles required	b-value	Error %
2400	6144	1020	0.0
4800	3072	508	0.0
9600	1536	252	0.1
14400	1024	167	0.1
19200	768	124	0.1
28800	512	82	0.6
57600	256	39	0.4
115200	128	18	2.3

16 MHz

BaudRate	Cycles required	b-value	Error %
2400	6667	1107	0.0
4800	3333	552	0.0
9600	1667	274	0.0
14400	1111	181	0.2
19200	833	135	0.0
28800	556	89	0.3
57600	278	42	1.0
115200	139	19	1.4