To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

RENESAS

H8/300L

Auto Baud Rate Detection (AutoBaud)

Introduction

This document describes the software implementation of baud rate detection of an incoming data. The SLP MCU is used to demonstrate the detection of 1200, 2400, 4800, 9600 and 19200bps.

This automatic detection is useful for establishing communication link between two devices. The slave device will be able to detect the baud rate of the master controller and adjust accordingly.

This protocol can be implemented on any MCU that carry an asynchronous serial port with a baud rate generator.

In this application note, the protocol is demonstrated using the ALE300L emulator (SLP MCU H8/38024) connected to the General Application Board. The SLP MCU is emulating as a slave device, whereas the PC (using the built-in HyperTerminal) is acting as a master controller.

Target Device

SLP – H8/38024

Contents

1. Theory

1.1 Detection algorithm

The detection algorithm is to read the sequence of bit received, based on a preset baud rate, and thus determining the incoming data rate.

In this example, the incoming data from the master is predefined as the “RETURN” character (0x0D), and the initial Baud rate is preset to 9600 bps.

The detection algorithm can be classified into three main methods

1. Baud < 1200
2. Baud > 1200 and Baud <= 9600
3. Baud > 9600

The following are the communication mode setting:

Start Bit:	1
Data Bit:	8
Stop Bit:	1
Parity Bit:	None
Flow Control:	None

For the <RETURN> Character (0x0D)

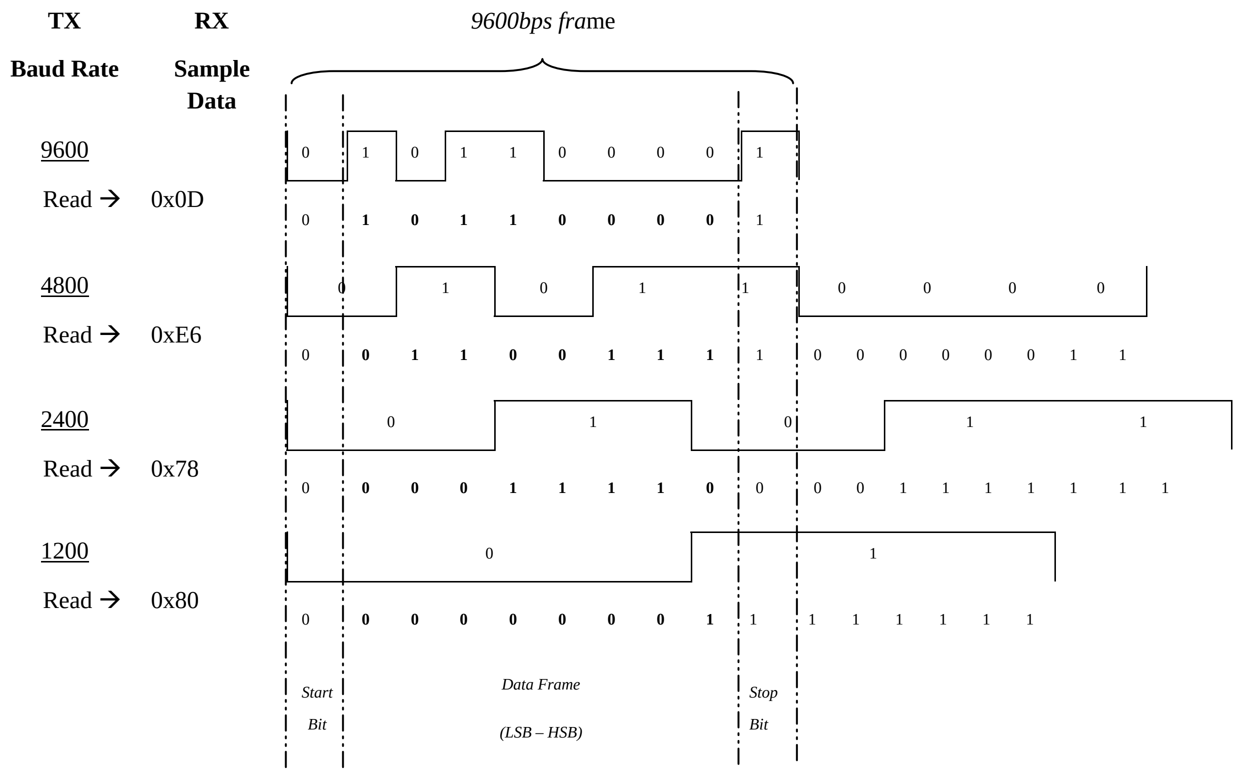
0	1	0	1	1	0	0	0	0	0
Start Bit	Data – 0x0D								Stop Bit

If a data is sent based on 9600 Baud, the receiver will receive the same data if it is set at the same Baud rate. This is because the incoming data is sampled correctly, upon the activation of the start bit. The Serial bus is normally in the mark state (high level). When a space (low level) is detected, identify as a Start bit, the incoming serial data will be sampled. In SCI3 of SLP MCU, the data is sampled on the 8^{th} pulse of a clock with a frequency, 16 times the bit rate (Data latched at the center of bit).

Based on the above reasoning, when the receiver is set to 9600 Baud rate. It will receive (sample) different patterns of data from the transmitter when the transmitter is set to a different baud rate.

1.1.1 Baud >= 1200 and Baud <= 9600

The diagram below will illustrate the read data when 0x0D (RETURN) is send at various baud rate.



The transmitter's serial data stream, which based on different baud rate will be fed into the 9600bps receiver. Data will be sampled at this rate. With the first data bit being the least significant bit.

Based on the above illustration, when data is transmitted at 4800 bps (slower by a factor of two as compared to the receiver 9600 bps), the transmitter initial start bit will be lengthened. This will be treated as the start bit and first data bit in the receiver 9600 bps frame.

Thus the following data is received at 9600 bps:

- At 9600bps, received data is 0x0D.
- At 4800bps, received data is 0xE6.
- At 2400bps, received data is 0x78.
- At 1200bps, received data is 0x80.

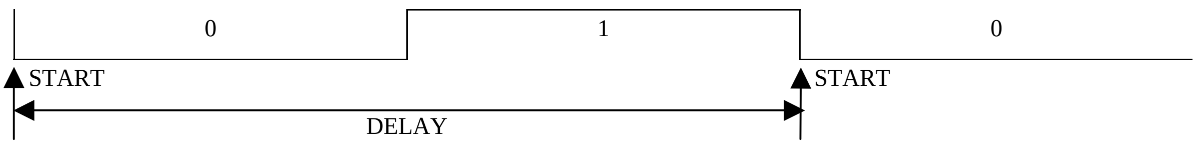
1.1.2 Baud < 1200

For Baud Rate below 1200 (600 & 300 Baud), the receiver will read a same data pattern of 0x00. This will not prevent the detection procedure. The <RETURN> (0xD0) has several transition of High to Low, which can signify a mark (Start) of a newly received byte at a higher baud rate receiver (In this case 9600 bps).

In another words, a high baud rate receiver will be able to receive two more bytes of data from a low baud rate transmitter.

If the timing delay of the "new" byte is measured, the transmitter baud rate can be predicted.

The following illustrate the initial transmitter bit stream and the calculation of the delay:



600 bps	16 '0'	16 '1'	16 '0'
300 bps	32 '0'	32 '1'	32 '0'

600 bps: 32 bit delay at 9600 Baud = 32* 1/9600 = 3.33 ms

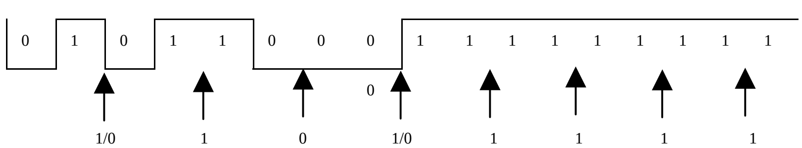
300 bps: 64 bit delay at 9600 Baud = 64* 1/9600 = 6.66ms

1.1.3 Baud > 9600

For Baud Rate above 9600 (19200 Baud), the receiver will read different patterns, as the sampling window will capture the data transition. The transition (0 to 1) and (1 to 0) may be interpret as '0' or '1'.

Thus in this case,

TX Baudrate	RX Sample Data
19200	F2, F3, FA or FB



The sampled data can fall into few possibilities:

- 0xF2
- 0xF3
- 0xFA
- 0xFB

1.2 MCU SCI Setting

The Serial communication Interface (SCI) is set to

- 9600 Baud
- 1 start bit
- 8 data bit
- 1 stop bit
- no parity bit

When the baud rate is determined, the SCI will have to set the following

- CKS1 (bit1) & CKS0 (bit 0) of SMR register (**n**) [determine the input clock to SCI]
- BRR register (**N**) [Baud rate generator] (0<N<255)

		SMR setting	
n	Clock	CKS1 (bit 1)	CKS0 (bit 0)
0	ϕ	0	0
0	$\phi w/2$ / ϕw	0	1
2	$\phi/16$	1	0
3	$\phi/64$	1	1

$$N = \frac{\phi OSC}{(64 \times 2^{2n} \times B)} - 1$$

whereby,

B Bit Rate (bit/sec)

N Baud Rate Generator BRR setting (0<N<255)

OSC Value of OSC (Hz)

n Baud Rate generator input clock number (n=0,2,3)

The above theory proves to able to detect the different baud rate. However due to the input clock selection, certain baud rates may not be feasible to be generated. The error rate may be too high (recommend < 1%).

The error can be calculated as:

$$\text{Error (\%)} = \frac{\text{B (rate obtained from n,N, OSC) –R(desired Bit rate)}}{\text{R (desired Bit rate)}} \times 100$$

The following is a generated recommended setting for n & N, based on main input clock.

Frequency OSC (MHz)	n	Desired bps B	Calculated N	Selected N	New bps B	Error %
9.8304	0	1200	127.00	127	1200.00	0.00%
	0	2400	63.00	63	2400.00	0.00%
	0	4800	31.00	31	4800.00	0.00%
	0	9600	15.00	15	9600.00	0.00%
	0	19200	7.00	7	19200.00	0.00%
	0	31250	3.92	4	30720.00	-1.70%
	2	1200	7.00	7	1200.00	0.00%
10	0	1200	129.21	129	1201.92	0.16%
	0	2400	64.10	64	2403.85	0.16%
	0	4800	31.55	31	4882.81	1.73%
	0	9600	15.28	15	9765.63	1.73%
	0	19200	7.14	7	19531.25	1.73%
	0	31250	4.00	4	31250.00	0.00%
	2	1200	7.14	7	1220.70	1.73%
16	0	1200	207.33	207	1201.92	0.16%
	0	2400	103.17	103	2403.85	0.16%
	0	4800	51.08	51	4807.69	0.16%
	0	9600	25.04	25	9615.38	0.16%
	0	19200	12.02	12	19230.77	0.16%
	0	31250	7.00	7	31250.00	0.00%
	2	1200	12.02	12	1201.92	0.16%

For other frequency setting, please refer to the MCU hardware manual

1.3 Alternative solution

The simple alternative is to create a protocol such that the master controller will continue to send a series of characters, such as 0xA, 0xB, 0xC at the desired baud rate and wait for response. On the slave end, the MCU will switch itself to different baud rate, in order to capture the correct data stream, and send an acknowledgment when the pattern of data is recognized.

2. Operation

Objective: The SLP will be able to switch to the respective baud rates (1200, 4800, 9600 & 19200 bps), based on a single character (Return- 0x0D) send from the PC HyperTerminal.

2.1 Environment Setup

The setup is illustrated as follow:

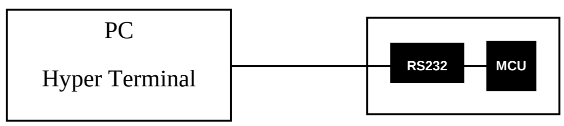
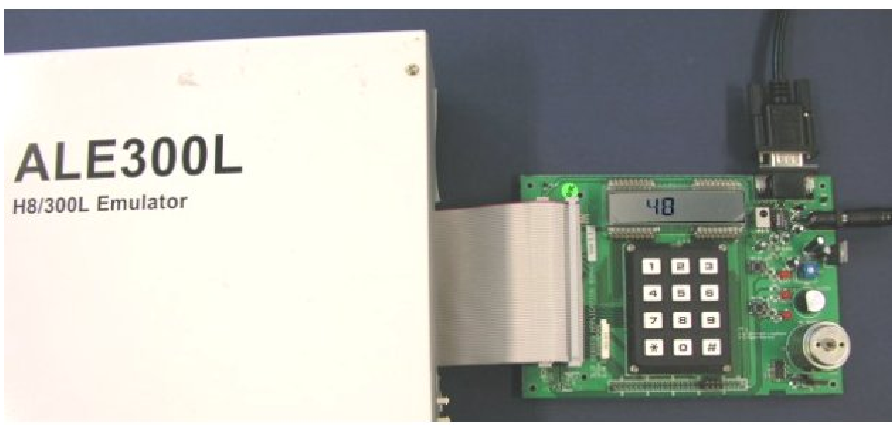


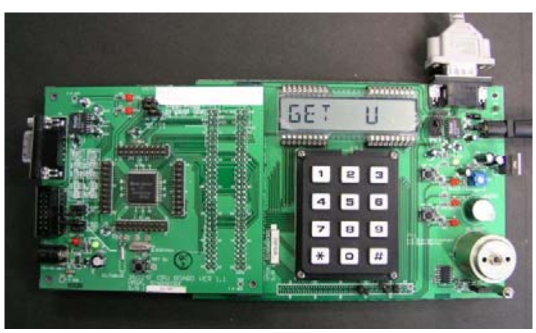
Figure 1 Basic Block Diagram

The setup of tools:



Detection of

Baud Rate - 4800 bps



Detection of

Character – ‘U’ from PC

Figure 2 Use of ALE300L Emulator or SLP CPU Board with the General Application Board

If the general application board is not available, a simple serial driver will have to be built to condition the signal level between the SLP MCU & the PC serial port. The LCD panel in the application board acted as a confirmation of data.

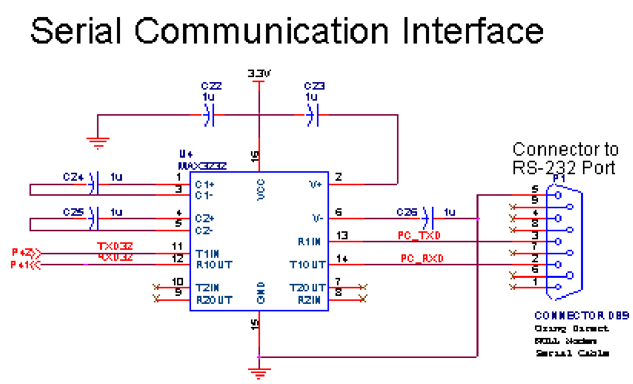


Figure 3 RS232 serial connection

The PC HyperTerminal setting

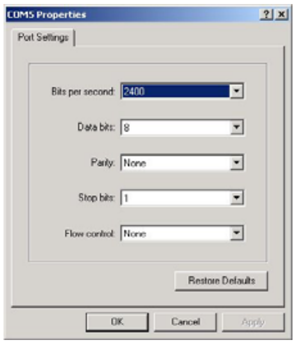


Figure 4 PC HyperTerminal setting

2.2 Operation & Observation

1. Set the PC to any baud rate (1200, 4800, 9600, or19200)
2. Hit the <RETURN> key
3. LCD display show Baud Rate detection & a stream of data (“BAUD DETECT”) is send back to the PC
4. Press any key
    a. The key pressed will be display on the LCD and send back to the PC
5. Press ‘a’ to abort
6. Goto step 1
    a. Click on [Call/ Disconnect] in HyperTerminal
    b. Click on [File/ Property/Configure] to re-configure to a new Baud rate.

3. Code listing

The attached code is generated using HEW project generator targeting at H8/38024 SLP MCU. The toolchain used is the free SLP/TINY toolchain.

The main routines of the auto baud rate detection are attached as follows.

In summary, the code provides a basic framework for users to have a quick start. The code is made to be readable, and thus it is not optimized. The Bprintf() function, (customized printf()) is detailed under the Application Note "Writing a printf function to LCD & serial port".

```
// autobaud.c
#include "iodefine.h"
#include "auto_baud.h"
#include <machine.h>

int lcd_cursor_pos=8;

void main(void)
{
   init_io();
   init_lcd();

   while(1)
   {
      init_sci();
      auto_detect();
   }
}

unsigned char auto_detect(void)
{
   static unsigned char sequence='1';
   unsigned char b_data, b_data2;
   unsigned int baudrate,i;

   // Start up message with sequence numering
   lcd_cursor_pos=8;
   Bprintf("BAUD %c ", (BYTE)sequence,(DWORD)SPACE);
   sequence++;

   b_data             = sci_charget();
   P_SCI3.SCR3.BIT.RE = 0x0;           //disable
   P_SCI3.SSR.BYTE    = 0x00;          //84 clear error

   if (b_data==0x00)                   //low baud rate
   {
      // start timer
      // watch for sci_charget()
      // Measure time
      // Determine new Baud
      lcd_cursor_pos=8;
      Bprintf("LOW BR", (BYTE)baudrate,(DWORD)b_data);
      P_SCI3.SCR3.BIT.RE = 0x1;        //enable
      P_SCI3.SSR.BYTE = 0x00;          //84 clear error

      //clear leftover data
      for(i=0;i<10000;i++);            // delay
      if ((P_SCI3.SSR.BIT.RDRF) == 0)
         b_data2=P_SCI3.RDR;
      return(0);
   }

   //to prevent latching in of the leftover data
   //dummy read to clear the initial unwanted data
```

```
    for(i=0;i<10000;i++);                  // delay
    if ((P_SCI3.SSR.BIT.RDRF) == 0)
       b_data2=P_SCI3.RDR;

    if (b_data == 0x80)                    // 1200 bps
    {  baudrate = BR1200;
       P_SCI3.SMR.BYTE |=BR12_CKS;
       P_SCI3.BRR     = BR12_BRR;
    }

    else if (b_data == 0x78)               //2400 bps
    {  baudrate = BR2400;
       P_SCI3.SMR.BYTE |=BR24_CKS;
       P_SCI3.BRR     = BR24_BRR;
    }

    else if (b_data == 0xE6)               // 4800 bps
    {  baudrate = BR4800;
       P_SCI3.SMR.BYTE |=BR48_CKS;
       P_SCI3.BRR     = BR48_BRR;
    }

    else if (b_data == 0x0D)               // 9600 bps
    {  baudrate = BR9600;
       P_SCI3.SMR.BYTE |=BR96_CKS;
       P_SCI3.BRR     = BR96_BRR;
    }

    else if (b_data == 0xF2   ||
           b_data == 0xF3      ||
           b_data == 0xFA     ||
           b_data == 0xFB )                // 19200 bps
    {  baudrate = BR19200;
       P_SCI3.SMR.BYTE |=BR192_CKS;
       P_SCI3.BRR     = BR192_BRR;
    }
    else                                   // unknown Baud
    {  baudrate = 0xFF;
       lcd_cursor_pos=8;
       Bprintf("X %x ", (BYTE)b_data,(DWORD)SPACE);
       P_SCI3.SCR3.BIT.RE = 0x1; //enable
       P_SCI3.SSR.BYTE = 0x00; //84 clear error
       return(0);
    }

    lcd_cursor_pos=8;
    Bprintf("%x      ", (BYTE)baudrate,(DWORD)SPACE);

    P_SCI3.SCR3.BIT.RE = 0x1;              //enable
    P_SCI3.SSR.BYTE = 0x00;                //84 clear error

    sci_putstr("BAUD ");
    sci_charput('D');
    sci_charput('E');
```

```
   sci_charput('T');
   sci_charput('E');
   sci_charput('C');
   sci_charput('T');
   sci_charput(' ');

   while(1)    // send back receive character based on new Baudrate
   {
      b_data= sci_charget();
      lcd_cursor_pos=8;
      Bprintf("GET  %c%c ", (BYTE)b_data,(DWORD)SPACE);
      sci_charput(b_data);

      if(b_data=='a' || b_data=='A')
         break;
   }
}

void sci_charput(char OutputChar)      //Serial Port
{
   while ((P_SCI3.SSR.BIT.TDRE) == 0);
   P_SCI3.TDR = OutputChar;
   P_SCI3.SSR.BIT.TDRE = 0;
}


unsigned char sci_charget(void)        //Serial Port
{
   while ((P_SCI3.SSR.BIT.RDRF) == 0);
   return(P_SCI3.RDR);
}

void sci_putstr(char *str)
{
    while(*str !='\0')
      sci_charput(*str++);
}

void init_sci(void)
{
   P_SCI3.SCR3.BYTE  = 0x30;
   P_SCI3.SMR.BYTE   = 0x00;
   P_SCI3.BRR        = BR96_BRR;
   P_SCI3.SPCR.BYTE  = 0xE0;
   P_SCI3.SSR.BYTE   = 0x84;
}
```

```
// autobaud.h
///////////////////////DEFINE////////////////////////////
// For auot baud rate
//#define OSC_16M
#define OSC_98304M

#define BR19200   0x19
#define BR9600    0x96
#define BR4800    0x48
#define BR2400    0x24
#define BR1200    0x12

#ifdef OSC_16M
#define BR192_CKS 0
#define BR192_BRR 12
#define BR96_CKS  0
#define BR96_BRR  25
#define BR48_CKS  0
#define BR48_BRR  51
#define BR24_CKS  0
#define BR24_BRR  103
#define BR12_CKS  0
#define BR12_BRR  207
#endif

#ifdef OSC_10M
#define BR192_CKS 0
#define BR192_BRR 12
#define BR96_CKS  0
#define BR96_BRR  0
#define BR48_CKS  0
#define BR48_BRR  0
#define BR24_CKS  0
#define BR24_BRR  64
#define BR12_CKS  0
#define BR12_BRR  129
#endif

#ifdef OSC_98304M
#define BR192_CKS 0
#define BR192_BRR 7
#define BR96_CKS  0
#define BR96_BRR  15
#define BR48_CKS  0
#define BR48_BRR  31
#define BR24_CKS  0
#define BR24_BRR  63
#define BR12_CKS  0
#define BR12_BRR  127
#endif
```

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Sep.03	—	First edition issued