- **Asynchronous Serial Communications**
- The MC9S12 Serial Communications Interface (SCI)

Asynchronous Data Transfer

• In asynchronous data transfer, there is no clock line between the two devices

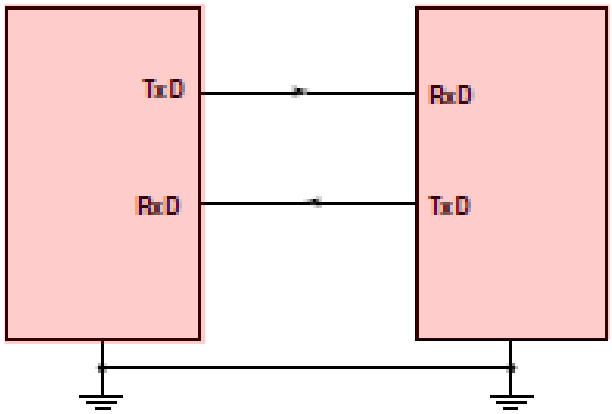
• Both devices use internal clocks with the same frequency

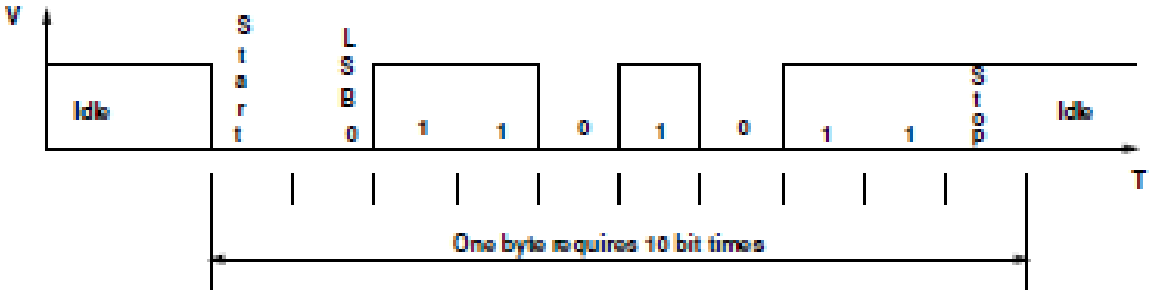
• Both devices agree on how many data bits are in one data transfer (usually 8, sometimes 9)

• A device sends data over an TxD line, and receives data over an RxD line
  – The transmitting device transmits a special bit (the start bit) to indicate the start of a transfer
  – The transmitting device sends the requisite number of data bits
  – The transmitting device ends the data transfer with a special bit (the stop bit)

• The start bit and the stop bit are used to synchronize the data transfer

Asynchronous Serial Communications





0xD6

11010110

Asynchronous Data Transfer

• The receiver knows when new data is coming by looking for the start bit (digital 0 on the RxD line).

• After receiving the start bit, the receiver looks for 8 data bits, followed by a stop bit (digital high on the RxD line).

• If the receiver does not see a stop bit at the correct time, it sets the Framing Error bit in the status register.

• Transmitter and receiver use the same internal clock rate, called the Baud Rate.

• At 9600 baud (the speed used by D-Bug12), it takes 1/9600 seconds for one bit, for a total of 10/9600 seconds, or 1.04 ms, for one byte.

Parity in Asynchronous Serial Transfers

• The HCS12 can use a parity bit for error detection.

– There are two types of parity – even parity and odd parity

* With even parity, and even number of ones in the data clears the parity bit; an odd number of ones sets the parity bit. The data transmitted will always have an even number of ones.

* With odd parity, and odd number of ones in the data clears the parity bit; an even number of ones sets the parity bit. The data transmitted will always have an odd number of ones.

Asynchronous Data Transfer

• The HCS12 has two asynchronous serial interfaces, called the SCI0 and SCI1 (SCI stands for Serial Communications Interface)

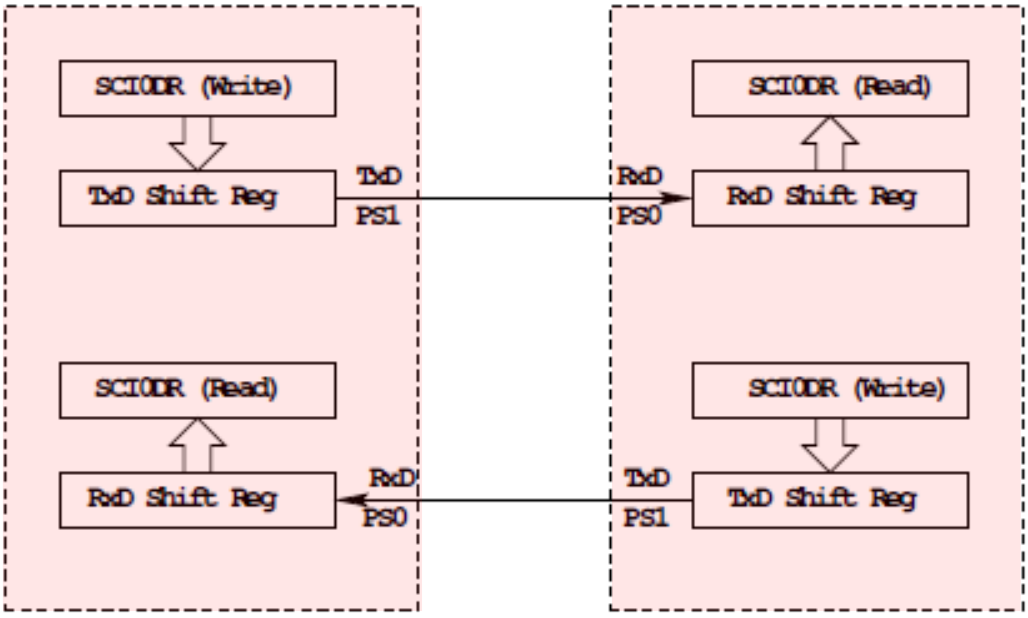
• SCI0 is used by D-Bug12 to communicate with the host PC

• When using D-Bug12 you normally cannot independently operate SCI0 (or you will lose your communications link with the host PC)

• The SCI0 TxD pin is bit 1 of Port S; the SCI1 TxD pin is bit 3 of Port S.

• The SCI0 RxD pin is bit 0 of Port S; the SCI1 RxD pin is bit 2 of Port S.

• In asynchronous data transfer, serial data is transmitted by shifting out of a transmit shift register into a receive shift register.



SCI0DR receive and transmit registers are separate registers.
distributed into two 8-bit registers, SCI0DRH and SCI0DRL

An overrun error is generated if RxD shift register filled before SCI0DR read

Timing in Asynchronous Data Transfers

• The BAUD rate is the number of bits per second.

• Typical baud rates are 1200, 2400, 4800, 9600, 19,200, and 115,000

• When not transmitting the TxD line is held high.

• When starting a transfer the transmitting device sends a start bit by bringing TxD low for one bit period (104 μs at 9600 baud).

• The receiver knows the transmission is starting when it sees RxD go low.

• The receiver checks the data three times for each bit. If the data within a bit is different, there is an error. This is called a noise error.

• The transmitter ends the transmission with a stop bit, which is a high level on TxD for one bit period.

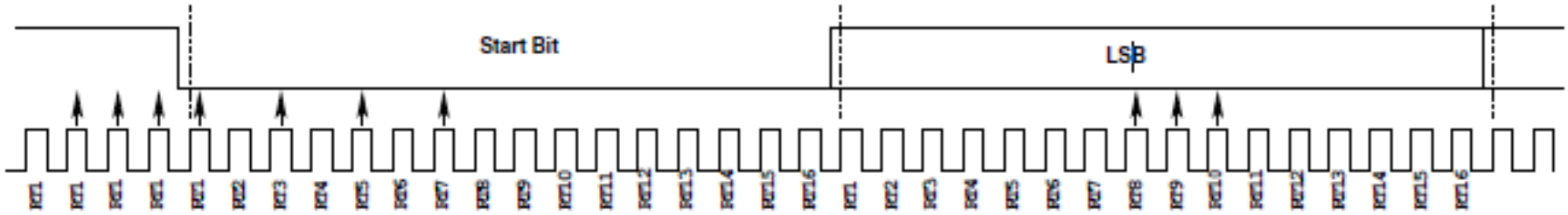
• If the receiver sees a start bit, but fails to see a stop bit, there is an error. Most likely the two clocks are running at different frequencies (generally because they are using different baud rates). This is called a framing error.

• The transmitter clock and receiver clock will not have exactly the same frequency. The transmission will work as long as the frequencies differ by less 4.5% (4% for 9-bit data).

Timing in Asynchronous Data Transfers

ASYNCHRONOUS SERIAL COMMUNIATIONS

Baud Clock = 16 x Baud Rate



Start Bit – Three 1's followed by 0's at RT1, 3, 5, 7
(Two of RT3, 5, 7 must be zero –
If not all zero, Noise Flag set)

If no stop bit detected, Framing Error Flag set

Baud clocks can differ by 4.5% (4% for 9 data bits)
with no errors.

Data Bit – Check at RT8, 9, 10
(Majority decides value)
(If not all same, noise flag set)

Even parity -- the number of ones in data word is even
Odd parity -- the number of ones in data word is odd
When using parity, transmit 7 data + 1 parity, or 8 data + 1 parity

Baud Rate Generation

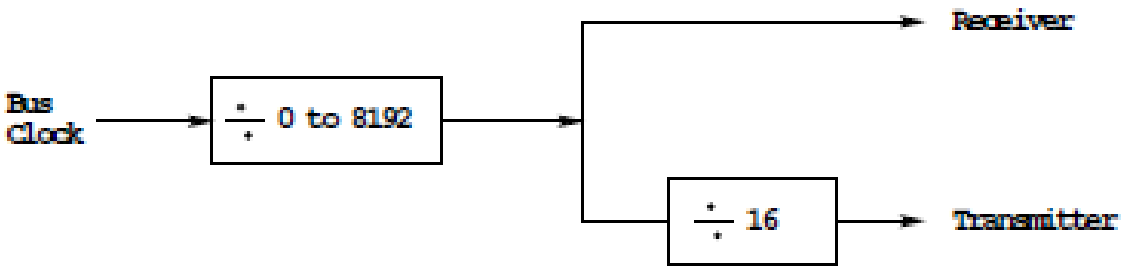
• The SCI transmitter and receiver operate independently, although they use the same baud rate generator.

• A 13-bit modulus counter generates the baud rate for both the receiver and the transmitter.

• The baud rate clock is divided by 16 for use by the transmitter.

• The baud rate is

SCIBaudRate = Bus Clock/(16 × SCI1BR[12:0])



• With a 24 MHz bus clock, the following values give typically used baud rates.

Bits SBR[12:0]	Receiver Clk (Hz)	Transmitter Clk(Hz)	Target Baudrate	Error (%)
39	615385	38462	38400	0.16
78	307692	19231	19200	0.16
156	153846	9615	9600	0.16
312	76923	4808	4800	0.16

SCI Registers

• Each SCI uses 8 registers of the HCS12. In the following we will refer to SCI1.

0	0	0	SBR12	SBR11	SBR10	SBR9	SBR8	SCI1BDH – 0x00D0
SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0	SCI1BDL – 0x00D1

LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT	SCI1CR1 – 0x00D2
TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK	SCI1CR2 – 0x00D3

TDRE	TC	RDRF	IDLE	OR	NF	FE	PF	SCI1SR1 – 0x00D4
0	0	0	0	0	BRK13	TXDIR	RAF	SCI1SR2 – 0x00D5

R8	T8	0	0	0	0	0	0	SCI1DRH – 0x00D5
R7/T7	R6/T6	R5/T5	R4/T4	R3/T3	R2/T2	R1/T1	R0/T0	SCI1DRL – 0x00D7

1. SCI Baud Rate Registers (SCI BDH/L)

SBR12 – SBR0: SCI Baud Rate Bits
The baud rate for the SCI is determined by these 13 bits.

2. SCI Control Register 1 (SCICR1)

M: Data Format Mode Bit
1 = One start bit, nine data bits, one stop bit
0 = One start bit, eight data bits, one stop bit

WAKE: Wakeup Condition Bit
A logic 1 (address mark) in the most significant bit position of a received data character, or a logic 0, an idle condition on the RXD

PE: Parity Enable Bit
1 = Parity function enabled
0 = Parity function disabled

PT: Parity Type Bit
1 = Odd parity
0 = Even parit

3. SCI Control Register 2 (SCICR2)

TIE: Transmitter Interrupt Enable Bit
1 = Transmit data register enable (TDRE) interrupt requests enabled
0 = TDRE interrupt requests disabled

RIE: Receiver Full Interrupt Enable Bit
1 = Receiver data register full (RDRF) enabled
0 = RDRF disabled

TE: Transmitter Enable Bit
1 = Transmitter enabled
0 = Transmitter disabled

RE: Receiver Enable Bit
1 = Receiver enabled
0 = Receiver disabled

RWU: Receiver Wakeup Bit Standby state
1 = RWU enables the wakeup function and inhibits further receiver interrupt requests. Normally, hardware wakes the receiver by automatically clearing RWU.
0 = Normal operation

4. SCI Status Register 1 (SCISR1)

TDRE: Transmit Data Register Empty Flag
1 = Byte transferred to transmit shift register; transmit data register empty
0 = No byte transferred to transmit shift register

RDRF: Receive Data Register Full Flag
1 = Received data available in SCI data register
0 = Data not available in SCI data register

OR: Overrun flag
1 = Overrun
0 = No overrun

NF: Noise Flag
1 = Noise
0 = No noise

FE: Framing Error Flag
1 = Framing error
0 = No framing error

PF: Parity Error Flag
1 = Parity error
0 = No parity error

5. SCI Status Register 2 (SCISR2)

BRK13: Break Transmit character length
1 = Break character is 13 or 14 bit long
0 = Break Character is 10 or 11 bit long

TXDIR: Transmitter pin data direction in Single-Wire mode.
1 = TXD pin to be used as an output in Single-Wire mode
0 = TXD pin to be used as an input in Single-Wire mode

6. SCI Data Registers (SCIDRH/L)

R8: R8 is the ninth data bit received when the SCI is configured for 9-bit data format (M = 1).

T8: T8 is the ninth data bit transmitted when the SCI is configured for 9-bit data format (M = 1).

R7-R0: Received bits seven through zero for 9-bit or 8-bit data formats

T7-T0: Transmit bits seven through zero for 9-bit or 8-bit formats

Example program using the SCI Transmitter

```
#include "derivative.h"
/* Program to transmit data over SCI port */

main()
{
        /*****************************************************************
        * SCI Setup
        *****************************************************************/
        SCI1BDL = 156; /* Set BAUD rate to 9,600 */
        SCI1BDH = 0;
        SCI1CR1 = 0x00; /*  0 0 0 0 0 0 0 0
                            | | | | | | | |
                            | | | | | | | \____ Even Parity
                            | | | | | | \_______ Parity Disabled
                            | | | | | \_________ Short IDLE line mode (not used)
                            | | | | \___________ Wakeup by IDLE line rec (not used)
                            | | | \_____________ 8 data bits
                            | | \_______________ Not used (loopback disabled)
                            | \_________________ SCI1 enabled in wait mode
                            \___________________ Normal (not loopback) mode
                            */

        SCI1CR2 = 0x08;   /* 0 0 0 0 1 0 0 0
                             | | | |  | | | |
                             | | | |  | | | \____ No Break
                             | | | |  | | \_______ Not in wakeup mode (always awake)
                             | | | |  | \_________ Receiver disabled
                             | | | |  \___________ Transmitter enabled
                             | | | \_____________ No IDLE Interrupt
                             | | \_______________ No Receiver Interrupt
                             | \_________________ No Transmit Complete Interrupt
                             \___________________ No Transmit Ready Interrupt
                            */
        /*****************************************************************
        * End of SCI Setup
        *****************************************************************/
```

```
        SCI1DRL = 'h'; /* Send first byte */
        while ((SCI1SR1 & 0x80) == 0) ; /* Wait for TDRE flag */

        SCI1DRL = 'e'; /* Send next byte */
        while ((SCI1SR1 & 0x80) == 0) ; /* Wait for TDRE flag */

        SCI1DRL = 'l'; /* Send next byte */
        while ((SCI1SR1 & 0x80) == 0) ; /* Wait for TDRE flag */

        SCI1DRL = 'l'; /* Send next byte */
        while ((SCI1SR1 & 0x80) == 0) ; /* Wait for TDRE flag */

        SCI1DRL = 'o'; /* Send next byte */
        while ((SCI1SR1 & 0x80) == 0) ; /* Wait for TDRE flag */
}
```

Example program using the SCI Receiver

```
/* Program to receive data over SCI1 port */

#include "derivative.h"
#include "vectors12.h"

#define enable() __asm(cli)

interrupt void sci1_isr(void);
volatile unsigned char data[80];
volatile int i;

main()
{
        /*****************************************************************
        * SCI Setup
        *****************************************************************/
        SCI1BDL = 156; /* Set BAUD rate to 9,600 */
        SCI1BDH = 0;
        SCI1CR1 = 0x00;  /* 0 0 0 0 0 0 0 0
                            | | | | | | | |
                            | | | | | | | \____ Even Parity
                            | | | | | | \_______ Parity Disabled
                            | | | | | \_________ Short IDLE line mode (not used)
                            | | | | \___________ Wakeup by IDLE line rec (not used)
                            | | | \_____________ 8 data bits
                            | | \_______________ Not used (loopback disabled)
                            | \_________________ SCI1 enabled in wait mode
                            \___________________ Normal (not loopback) mode
                            */
        SCI1CR2 = 0x04;  /* 0 0 1 0 0 1 0 0
                            | | | | |  | | |
                            | | | | |  | | \____ No Break
                            | | | | |  | \_______ Not in wakeup mode (always awake)
                            | | | | |  \_________ Receiver enabled
                            | | | | \____________ Transmitter disabled
                            | | | \_____________ No IDLE Interrupt
                            | | \_______________ Receiver Interrupts used
                            | \_________________ No Transmit Complete Interrupt
                            \___________________ No Transmit Ready Interrupt
                            */
```

```
        UserSCI1 = (unsigned short) &sci1_isr;
        i = 0;
        enable();

        /*********************************************************************
        * End of SCI Setup
        *********************************************************************/
        while (1)
        {
                /* Wait for data to be received in ISR, then do something with it */
        }
}

interrupt void sci1_isr(void)
{
        char tmp;
        /* Note: To clear receiver interrupt, need to read SCI1SR1, then read SCI1DRL.
        * The following code does that
        */

        if ((SCI1SR1 & 0x20) == 0) return;   /* Not receiver interrupt */
        data[i] = SCI1DRL;
        i = i+1;
        return;
}
```