

## M16C/26

### Using UARTs in Asynchronous Mode

#### 1.0 Abstract

The following article introduces and describes how to use the UART's of the M16C/26 (M30262) Flash microcontroller (MCU) for asynchronous communications. A sample program, using UART0 in asynchronous mode, was written for the MSV30262-SKP connected to a PC using a serial cable. Data sent by the MSV30262 SKP board can be displayed in Hyperterminal application.

#### 2.0 Introduction

The Renesas M30262 is a 16-bit MCU based on the M16C/60 series CPU core. The MCU features include up to 64KB of Flash ROM, 2KB of RAM, and 4KB of Virtual EEPROM. The peripheral set includes 10-bit A/D, Timers, DMA, GPIO, and 3 UART's. Any of the three UART's (UART0, UART1, and UART2) can be used in asynchronous mode and in synchronous mode. In addition, UART2 also supports I2C and SPI. The block diagrams of the UARTs are shown in Figure 1, Figure 2 and Figure 3.

UART (**U**niversal **A**synchronous **R**eceive/**T**ransmit) is a common form of serial communications. Without handshaking or flow control, will only require 2 lines, receive and transmit. If handshaking is needed, another 2 lines, CTS (**C**lear-**T**o-**S**end) and RTS (**R**quest-**T**o-**S**end), are required.

The remainder of this document shows how to configure a UART, i.e. UART0, in asynchronous mode.

(UART0)

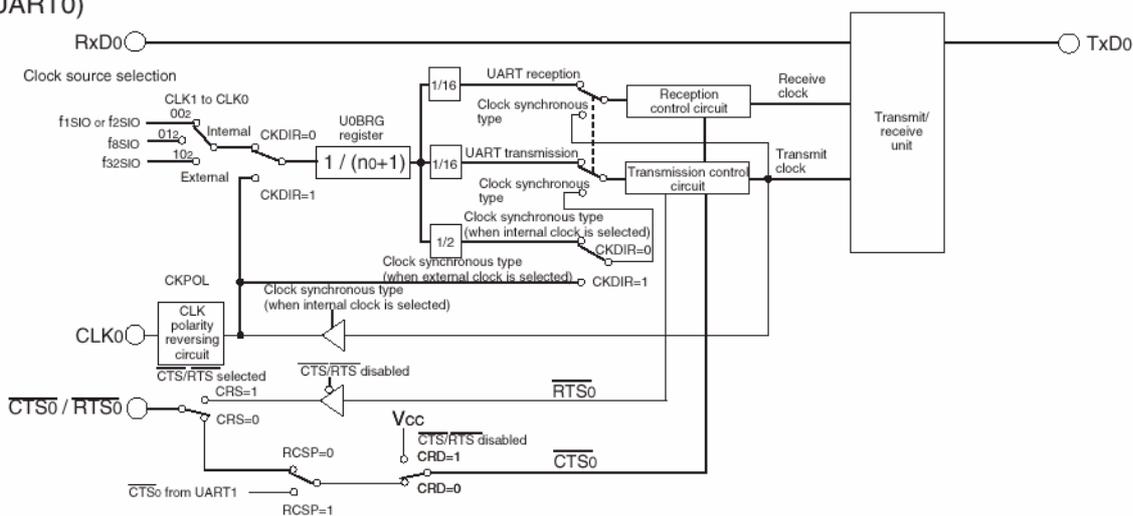


Figure 1 UART0 Block Diagram

(UART1)

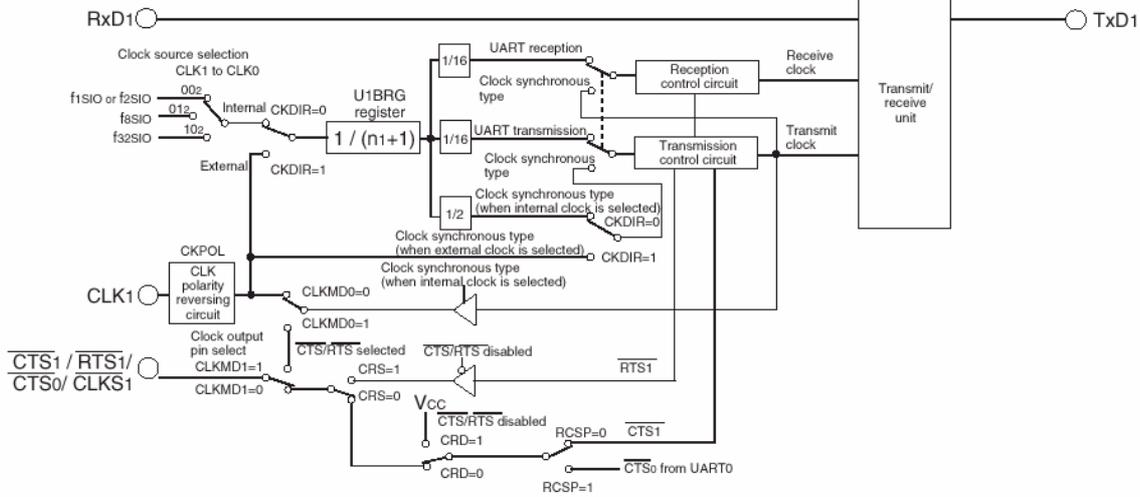


Figure 2 UART1 Block Diagram

(UART2)

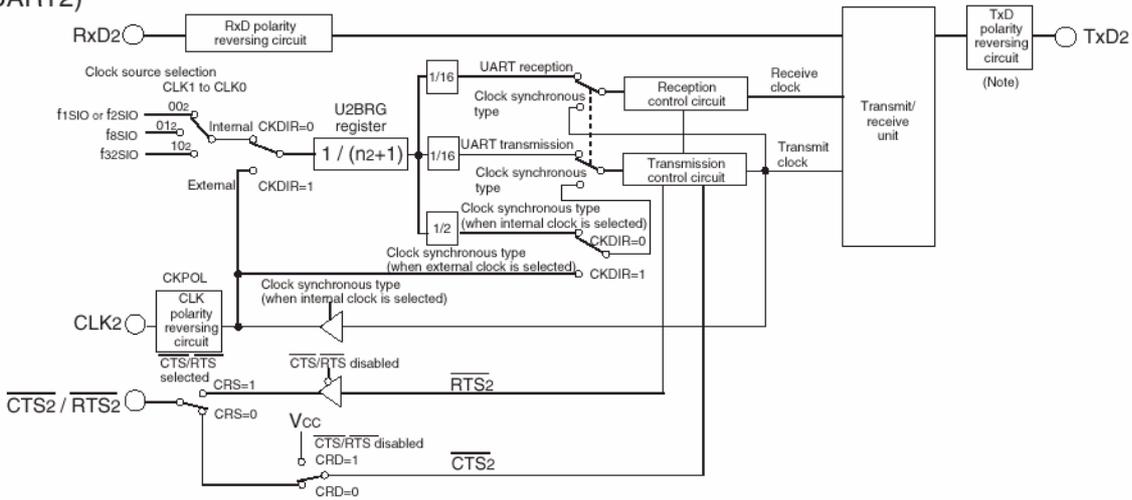


Figure 3 UART2 Block Diagram

## 3.0 UART

### 3.1 Parameters

The M16C/26 UART's are programmable and so given parameters are necessary to be able to configure the UART's. These parameters, which are shown below, will vary from one application to another.

- **Baud Rate** - The baud rate is the speed at which the data is transmitted or received. Examples are 19.2Kbps, 9,600bps, 4,800bps, or 2,400bps. Higher speeds of 115Kbps and 57.6Kbps are available in some applications. The baud rate is used to calculate the value for the bit rate generator. Use the equation below to calculate the value for the UARTi Bit Rate Generator, UiBRG.

$$\text{UiBRG} = ((\text{clock source} / 16) / \text{baud rate}) - 1$$

Where clock source could be f1, f8, or f32. The clock source is selected from UARTi transmit/receive control register 0, UiC0.

- **Data Length** - There are three data lengths to choose from: 7, 8, or 9 bits.
- **Stop Bit** - There are two options for Stop bits: 1 or 2.
- **Parity Bit** - There are three options: odd, even, or no parity.
- **Transmission/Reception (Handshaking or Flow) Control** - The available options are: CTS, RTS, or no control. Having some form of transmission/reception control will minimize data errors because data is only sent when one party is ready to receive.

### 3.2 Configuring the UART

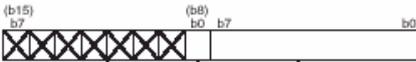
The steps necessary to configure a UART are shown below. An example using these steps can be found in the `init_uart()` routine of the sample program.

1. Initialize the UARTi transmit/receive mode register, **UiMR**, based on parameters discussed in section 3.1 Parameters.
2. Initialize UARTi transmit/receive control register 0, **UiC0**, based on baud rate and flow control.
3. Initialize UARTi bit rate generator, **UiBRG**, using the calculated value based on baud rate and clock source selected in **UiC0**.
4. Initialize UART transmit/receive control register 2, **UcON**, based on flow control.
5. Initialize UARTi interrupt vector **SiTIC** for transmit and **SiRIC** for receive.
6. Enable transmit and receive using UARTi transmit control register 1, **UiC1**.

There are several registers used to configure the UART. To simplify the article, we are only going to show the registers used for configuring UART0 (or UART1), which are shown in Figure 4 to Figure 6.

After initialization, you can start sending and receiving data.

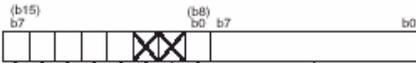
### UARTi transmit buffer register (Note)



Symbol	Address	When reset
U0TB	03A3 <sub>16</sub> , 03A2 <sub>16</sub>	Indeterminate
U1TB	03AB <sub>16</sub> , 03AA <sub>16</sub>	Indeterminate
U2TB	037B <sub>16</sub> , 037A <sub>16</sub>	Indeterminate

Function	R	W
Transmit data	X	O
Nothing is assigned. In an attempt to write to these bits, write 0. The value, if read, turns out to be indeterminate.	—	—

Note: Use MOV instruction to write to this register.



Symbol	Address	When reset
U0RB	03A7 <sub>16</sub> , 03A6 <sub>16</sub>	Indeterminate
U1RB	03AF <sub>16</sub> , 03AE <sub>16</sub>	Indeterminate
U2RB	037F <sub>16</sub> , 037E <sub>16</sub>	Indeterminate

Bit symbol	Bit name	Function (During UART mode)	R	W
—	—	Receive data	O	X
Nothing is assigned. In an attempt to write to these bits, write 0. The value, if read, turns out to be 0.				
ABT	Arbitration lost detecting flag (Note 2)	Invalid	O	O
OER	Overrun error flag (Note 1)	0: No overrun error 1: Overrun error found	O	X
FER	Framing error flag (Note 1)	0: No framing error 1: Framing error found	O	X
PER	Parity error flag (Note 1)	0: No parity error 1: Parity error found	O	X
SUM	Error sum flag (Note 1)	0: No error 1: Error found	O	X

Note 1: Bits 15 through 12 are set to 0 when the serial I/O mode select bit (bits 2 to 0 at addresses 03A0<sub>16</sub>, 03A8<sub>16</sub> and 0378<sub>16</sub>) are set to 000<sub>2</sub> or the receive enable bit is set to 0.

(Bit 15 is set to 0 when bits 14 to 12 all are set to 0.) Bits 14 and 13 are also set to 0 when the lower byte of the UARTi receive buffer register (addresses 03A6<sub>16</sub>, 03AE<sub>16</sub> and 037E<sub>16</sub>) is read out.

Note 2: Arbitration lost detecting flag is allocated to U2RB and only '0' must be written. Nothing is assigned in bit 11 of U0RB and U1RB. Set to '0' when writing. If read, value is '0'.

### UARTi bit rate generator (Note 1, 2)



Symbol	Address	When reset
U0BRG	03A1 <sub>16</sub>	Indeterminate
U1BRG	03A9 <sub>16</sub>	Indeterminate
U2BRG	0379 <sub>16</sub>	Indeterminate

Function	Values that can be set	R	W
Assuming that set value = n, BRGi divides the count source by n + 1	00 <sub>16</sub> to FF <sub>16</sub>	X	O

Note 1: Write a value to this register while transmit/receive halts.  
Note 2: Use MOV instruction to write to this register.

Figure 4 UARTi (i = 0, 1, 2) Registers

### UART<sub>i</sub> transmit/receive mode register

Bit	Symbol	Address	When reset
b7			
b6			
b5			
b4			
b3			
b2			
b1			
b0			
0	UIMR(i=0,1)	03A0 <sub>16</sub> , 03A8 <sub>16</sub>	00 <sub>16</sub>

Bit symbol	Bit name	Function (During UART mode)	R	W
SMD0	Serial I/O mode select bits	1 0 0 : Transfer data 7 bits long	○	○
SMD1		1 0 1 : Transfer data 8 bits long	○	○
SMD2		1 1 0 : Transfer data 9 bits long	○	○
		0 0 0 : Serial I/O invalid	○	○
		0 1 0 : Inhibited	○	○
		0 1 1 : Inhibited	○	○
		1 1 1 : Inhibited	○	○
CKDIR	Internal/external clock select bit	0 : Internal clock 1 : External clock (Note 1)	○	○
STPS	Stop bit length select bit	0 : One stop bit 1 : Two stop bits	○	○
PRY	Odd/even parity select bit	Valid when bit 6 = "1" 0 : Odd parity 1 : Even parity	○	○
PRYE	Parity enable bit	0 : Parity disabled 1 : Parity enabled	○	○
	Reserved	Must always be "0"	○	○

### UART<sub>i</sub> transmit/receive control register 0

Bit	Symbol	Address	When reset
b7			
b6			
b5			
b4			
b3			
b2			
b1			
b0			
	UIC0(i=0,1)	03A4 <sub>16</sub> , 03AC <sub>16</sub>	08 <sub>16</sub>

Bit symbol	Bit name	Function (During UART mode)	R	W
CLK0	BRG count source select bits	0 0 : f <sub>1SIO</sub> or f <sub>2SIO</sub> is selected (Note 2)	○	○
CLK1		0 1 : f <sub>3SIO</sub> is selected 1 0 : f <sub>2SIO</sub> is selected 1 1 : Inhibited	○	○
CRS	CTS/RTS function select bit	Valid when bit 4 = "0" 0 : CTS function is selected (Note 1) 1 : RTS function is selected (Note 3)	○	○
TXEPT	Transmit register empty flag	0 : Data present in transmit register (during transmission) 1 : No data present in transmit register (transmission completed)	○	✗
CRD	CTS/RTS disable bit	0 : CTS/RTS function enabled 1 : CTS/RTS function disabled (P6 <sub>0</sub> and P6 <sub>4</sub> function as programmable I/O port)	○	○
NCH	Data output select bit	0 : TXD <sub>i</sub> pin is CMOS output 1 : TXD <sub>i</sub> pin is N-channel open-drain output	○	○
CKPOL	CLK polarity select bit	Must always be "0" - When bit 4 = "0", transfer clock and receive data is input at falling edge. - Transmit data is output at rising edge of transfer clock and receive data is input at falling edge.	○	○
UFORM	Transfer format select bit	Must always be "0"	○	○

Note 1: Set the corresponding port direction register to "0".

Note 2: Peripheral clock select register bit PCLK1 is used to select between f<sub>1SIO</sub> and f<sub>2SIO</sub>.

Note 3: The settings of the corresponding port register and port direction register are invalid.

Figure 5 UART<sub>i</sub> (i = 0, 1) Registers

### UART<sub>i</sub> transmit/receive control register 1 (i=0, 1)

b7	b6	b5	b4	b3	b2	b1	b0	Symbol UiC1(i=0,1)	Address 03A5 <sub>16</sub> ,03AD <sub>16</sub>	When reset 02 <sub>16</sub>
Bit symbol	Bit name	Function (During UART mode)	R/W							
TE	Transmit enable bit	0 : Transmission disabled 1 : Transmission enabled	○ ○							
TI	Transmit buffer empty flag	0 : Data present in transmit buffer register 1 : No data present in transmit buffer register	○ ×							
RE	Receive enable bit	0 : Reception disabled 1 : Reception enabled	○ ○							
RI	Receive complete flag	0 : No data present in receive buffer register 1 : Data present in receive buffer register	○ ×							
Nothing is assigned. In an attempt to write to these bits, write 0. The value, if read, turns out to be 0.			— —							

### UART transmit/receive control register 2

b7	b6	b5	b4	b3	b2	b1	b0	Symbol UCON	Address 03B0 <sub>16</sub>	When reset X0000002
Bit symbol	Bit name	Function (During UART mode)	R/W							
U0IRS	UART0 transmit interrupt cause select bit	0 : Transmit buffer empty (TI = 1) 1 : Transmission completed (TXEPT = 1)	○ ○							
U1IRS	UART1 transmit interrupt cause select bit	0 : Transmit buffer empty (TI = 1) 1 : Transmission completed (TXEPT = 1)	○ ○							
U0RRM	UART0 continuous receive mode enable bit	Must always be 0	○ ○							
U1RRM	UART1 continuous receive mode enable bit	Must always be 0	○ ○							
CLKMD0	CLK/CLKS select bit 0	Invalid	○ ○							
CLKMD1	CLK/CLKS select bit 1 (Note)	Must always be 0	○ ○							
RCSP	Separate CTS/RTS bit	0 : CTS/RTS shared pin 1 : CTS/RTS separated	○ ○							
Nothing is assigned. In an attempt to write to this bit, write 0. The value, if read, turns out to be indeterminate.			— —							

Note: When using multiple pins to output the transfer clock, the following requirements must be met:  
\* UART1 internal/external clock select bit (bit 3 at address 03A8<sub>16</sub>) = 0.

Figure 6 UART<sub>i</sub> (i = 0, 1) Registers

#### **4.0 UART Demo Program**

The demo program was written to run on the MSV30262-SKP board with a null modem connected to the PC's RS-232C serial port. The Hyperterminal program (or any terminal program that can connect to the COM port) in Windows™ is used to view the data sent by the M16C/26. Like our parameters to configure UART0, the Hyperterminal must be configured to 19.2Kbps, one stop bit, no parity, and no flow control.

As soon as a connection is established, incrementing data, 0 to 9, is displayed on Hyperterminal window. Pressing the 'z' key on the keyboard will stop the M16C/26 from sending data. Pressing any other key will resume data transmission. An interrupt is generated every time a character is received.

The program was compiled using the KNC30 Compiler, which also came with the MSV30262-SKP. It can be modified to suit a user application.

#### **5.0 Conclusion**

Asynchronous mode offers simple communications using only two pins (when no handshaking is required). Due to this simple connectivity, it is easy to implement. Understanding the key parameters simplifies setup and usage.

#### **6.0 Reference**

##### **Renesas Technology Corporation Semiconductor Home Page**

<http://www.renesas.com>

##### **E-mail Support**

[support\\_apl@renesas.com](mailto:support_apl@renesas.com)

##### **Data Sheets**

- M16C/26 datasheets, M30262eds.pdf

##### **User's Manual**

- M16C/20/60 C Language Programming Manual, 6020c.pdf
- M16C/20/60 Software Manual, 6020software.pdf
- Interrupt Handler App Note, M16C26\_Interrupt\_Handlers\_in\_C.doc
- MSV30262-SKP Users Manual, Users\_Manual\_MSV30262.pdf

## 7.0 Software Code

```

/*****
*      File Name:  main_uart.c
*
*      Content: This program communicates from the M16C/26 to a terminal
*               program via UART0 and RS232.  UART0 configuration is:
*               19200 baud rate, 8-bit data, 1 stop bit, no parity, no flow control.
*
*               This program was written to run on the MSV30262 Board.
*               To see the data sent, use Hyperterminal configured as mentioned
*               above and connect a null modem cable between P1 and the PC's
*               com port.
*
*               An incrementing data (0 to 9) is sent to the Hyperterminal window.
*               To stop receiving data, press z on the keyboard. To resume, press
*               any key.
*
*      Version:  1.0
*      Date:    05-09-03
*
*      Copyright 2003 Renesas Technology America, Inc.
*      All rights reserved
*=====
*      $Log:$
*=====*/
/* Include files */
#include "..\common\sfr262.h"          // include M16C/26 header file

/* Setup interrupt routine for UART0 receive. This must also be setup in the
   vector table in SECT30_UART.INC */

#pragma INTERRUPT U0rec_ISR
void U0rec_ISR (void);

/* Function Prototypes */
void text_write (_far char * msg_string);
void mcu_init(void);
void uart_init(void);

/* Global variables */
char U0_in;                          // declare UART0 receive variable

/* String constants used for screen output *****/
const char cmd_clr_scr[] = {27, '[', '2', 'J', 0};
const char cmd_cur_home[] = {27, '[', 'H', 0};

/*****
Name:      Main
Parameters: none
Returns:   none
Description: This is the main program
*****/

```

```

main() {
    int count;           // declare count variable
    int convert;        // declare ASCII variable
    unsigned int delay; // declare delay variable
    int i;              // declare string pointer variable

    mcu_init();         // MCU initialization
    uart_init();        // UART initialization

    /* Text to be displayed at the beginning of the program output
       to the terminal window (\r\n = carriage return & linefeed) */

    text_write(cmd_clr_scr);           // clear screen
    text_write(cmd_cur_home);          // home cursor
    text_write("Renesas Technology America, Inc. \r\n");
    text_write("Renesas MSV30262-SKP UART demo. \r\n");
    text_write("Press z to stop, any key to continue. \r\n");

    /****** MAIN PROGRAM LOOP *****/
    while (1){

    // setup program loop to count from 0 to 9, stop when "z" is received

        while (U0_in != 'z'){           // count as long as "z" wasn't pressed
            text_write("\r\n");         // send carriage return and line feed

            for (count=0;(count<=9)&&(U0_in!='z');count++){ // count 0 to 9
                convert = count + 0x30; // convert count data to ASCII
                while(ti_u0c1 == 0);   // wait for previous transmission to complete
                u0tb = convert;        // put ASCII data in transmit buffer
                p7_0 = 1;              // turn off red LED
                p7_2 = ~p7_2;          // blink run LED D5
                for (delay=0x3fff; delay>0; delay--); // Count Delay
            }
        }
        asm("NOP");                    // Do nothing while stopped
    }                                   // (after "z" is pressed)
}

/*****
Name:      UART0 Receive Interrupt Routine
Parameters: none
Returns:   none
Description: Interrupt routine for UART0 receive
            Reads character received from keyboard and stores U0_in variable
*****/
void U0rec_ISR(void){
    while(ri_u0c1 == 0);               // make sure receive is complete
    U0_in = u0rb;                      // read in received data
    if (U0_in == 'z'){                 // If "z" was entered do the following:
        while(ti_u0c1 == 0);          // wait for previous transmission to complete
        u0tb = 'z';                   // echo "z" to screen
        p7_2 = 1;                     // turn off green LED
        p7_0 = 0;                     // turn on red LED
    }
}

```

```

/*****
Name:      text_write
Parameters: msg_string -> the text string to output
Returns:   none
Description: The following sends a text string to the terminal program
*****/
void text_write ( _far char * msg_string)
{
    char i;

    for (i=0; msg_string[i]; i++){ // This loop reads in the text string and
        while(ti_u0c1 == 0);      // puts it in the UART 0 transmit buffer
        u0tb = msg_string[i];
    }
}

/*****
Name:      mcu_init
Parameters: None
Returns:   None
Description: Initialization routine for the different MCU peripherals. See
            settings for details.
*****/
void mcu_init(void) {

    /* LED initialization */
    pd7_0 = 1;      // Change LED ports to outputs (connected to LEDs)
    pd7_1 = 1;
    pd7_2 = 1;

    p7 |= 0x7;     // turn off LEDs

    /* unused pins - configure as outputs to decrease power consumption */
    pd6 |= 0x9F;
    pd7 |= 0xF8;
    pd8_0 |= 0x2F;

    prc2 = 1;      // P9 is write protected - disable protection before writing to P9
    pd9 |= 0x0F;
    prc2 = 0;     // Write protect P9

    pd10 |= 0x1F; // upper bits are switch inputs
}

/*****
Name:      uart_init
Parameters: None
Returns:   None
Description: Uart0 initialization - 19200 baud, 8 data bits, 1 stop bit, no parity.
*****/

```

```

void uart_init(void) {

    u0mr = 0x05;          // UART0 transmit/receive mode register
    /*
    00000101;           // 8-bit data, internal clock, 1 stop bit, no parity
    |||||_____Serial I/O Mode select bit, SMD0
    |||||_____Serial I/O Mode select bit, SMD1
    |||||_____Serial I/O Mode select bit, SMD2
    |||||_____Internal/External clock select bit, CKDIR
    ||||_____Stop bit length select bit, STPS
    |||_____Odd/even parity select bit, PRY
    ||_____Parity enable bit, PRYE
    |_____Reserved, set to 0 */

    u0c0 = 0x10;        // UART0 transmit/receive control register 1
    /*
    00010000;          // f1 count source, CTS/RTS disabled, CMOS output
    |||||_____BRG count source select bit, CLK0
    |||||_____BRG count source select bit, CLK1
    |||||_____CTS/RTS function select bit, CRS
    |||||_____Transmit register empty flag, TXEPT
    ||||_____CTS/RTS disable bit, CRD
    |||_____Data output select bit, NCH
    ||_____CLK polarity select bit, CKPOL           - set to 0 in UART mode
    |_____Transfer format select bit, UFORM         - set to 0 in UART mode
*/

    u0brg = 64;         // set UART0 bit rate generator
    /*
    bit rate can be calculated by:
    bit rate = ((BRG count source / 16)/baud rate) - 1

    in this example: BRG count source = f1 (20MHz)
                    baud rate = 19200
                    bit rate = ((20MHz/16)/19200) - 1 = 64

    ** one has to remember that the value of BCLK does not affect BRG count source */

    ucon = 0x00;        // UART transmit/receive control register 2
    /*
    00000000;          // transmit irq not used
    |||||_____UART0 transmit irq cause select bit, U0IRS
    |||||_____UART1 transmit irq cause select bit, U1IRS
    |||||_____UART0 continuous receive mode enable bit, U0RRM - set to 0 in
UART mode
    |||||_____UART1 continuous receive mode enable bit, U1RRM           - set
to 0 in UART mode
    ||||_____CLK/CLKS select bit 0, CLKMD0 - set to 0 in UART mode
    |||_____CLK/CLKS select bit 1, CLKMD1 - set to 0 in UART mode
    ||_____Separate CTS/RTS bit, RCSP
    |_____Reserved, set to 0 */

```

```
u0tb = u0rb;          // clear UART0 receive buffer by reading
u0tb = 0;             // clear UART0 transmit buffer

asm("FCLR I");        // disable irqs before setting irq registers
s0ric = 0x04;         // Enable UART0 receive interrupt, priority level 4
asm("FSET I");        // Enable all interrupts

u0c1 = 0x05;          // UART0 transmit/receive control register 1
/*
00000000;             // enable transmit and receive
|||||||_____Transmit enable bit, TE
|||||||_____Transmit buffer empty flag, TI
|||||||_____Receive enable bit, RE
|||||_____Receive complete flag, RI
||||_____Reserved, set to 0
|||_____Reserved, set to 0
||_____Reserved, set to 0
|_____Reserved, set to 0 */
}
```

## Keep safety first in your circuit designs!

- Renesas Technology Corporation puts the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage. Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of nonflammable material or (iii) prevention against any malfunction or mishap.

## Notes regarding these materials

- These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corporation product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corporation or a third party.
- Renesas Technology Corporation assumes no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
- All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor for the latest product information before purchasing a product listed herein.  
The information described here may contain technical inaccuracies or typographical errors. Renesas Technology Corporation assumes no responsibility for any damage, liability, or other loss arising from these inaccuracies or errors.  
Please also pay attention to information published by Renesas Technology Corporation by various means, including the Renesas Technology Corporation Semiconductor home page (<http://www.renesas.com>).
- When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corporation assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
- Renesas Technology Corporation semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
- The prior written approval of Renesas Technology Corporation is necessary to reprint or reproduce in whole or in part these materials.
- If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination.  
Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
- Please contact Renesas Technology Corporation for further details on these materials or the products contained therein.