

MSC1210 ROM Routines

Russell Anderson

Data Acquisition Products -- Microsystem

ABSTRACT

The MSC1210 analog-to-digital converter (ADC) with 8051 microcontroller has 2K bytes of ROM that enable serial and parallel programming. Several MSC1210 ROM routines are useful and necessary for application programs. This application report describes these ROM routines and how to use them.

Some of the MSC1210 ROM routines are required for user applications. For example, a write to flash memory cannot execute from flash memory; the flash write routine must execute from the ROM or other non-flash memory. To maintain compatibility with future versions of the ROM code, a jump table is provided at fixed addresses. Convenient access to these routines is shown in Table 1. Note that some routine names have changed from previous versions of the ROM.a51 file.

Table 1. ROM Routines

Address	Routine	C declarations	Description
FFD5	put_string	void put_string(char code *string);	Output string
FFD7	page_erase	char page_erase (int faddr, char fdata, char fdm);	Erase flash page
FFD9	write_flash	Assembly only; DPTR = address, R5 = data	Fast flash write
FFDB	write_flash_chk	char write_flash_chk (int faddr, char fdata, char fdm);	Write flash byte, verify
FFDD	write_flash_byte	char write_flash_byte (int faddr, char fdata, char fdm);	Write flash byte
FFDF	faddr_data_read	char faddr_data_read(char faddr);	Read HW config byte from addr
FFE1	data_x_c_read	char data_x_c_read(int faddr, char fdm);	Read xdata or code byte
FFE3	tx_byte	void tx_byte(char);	Send byte to UART0
FFE5	tx_hex	void tx_hex(char);	Send hex value to UART0
FFE7	putok	void putok(void);	Send "OK" to UART0
FFE9	rx_byte	char rx_byte(void);	Read byte from UART0
FFEB	rx_byte_echo	char rx_byte_echo(void);	Read and echo byte on UART0
FFED	rx_hex_echo	char rx_hex_echo(void);	Read and echo hex on UART0
FFEF	rx_hex_int_echo	Int rx_hex_int_echo(void);	Read int as hex and echo: UART0
FFF1	rx_hex_rev_echo	Int rx_hex_rev_echo(void);	Read int reversed as hex and echo: UART0
FFF3	autobaud	void autobaud(void);	Set baud with received CR
FFF5	putspace4	void putspace4(void);	Output 4 spaces to UART0
FFF7	putspace3	void putspace3(void);	Output 3 spaces to UART0
FFF9	putspace2	void putspace2(void);	Output 2 spaces to UART0
FFFB	putspace1	void putspace1(void);	Output 1 space to UART0
FFFD	putcrl	void putcrl(void);	Output CR, LF to UART0
F97D ⁽¹⁾	cmd_parse	void cmd_parser(void)	See SBAA076B.pdf
FD3B ⁽¹⁾	monitor_isr	void monitor_isr() interrupt 6	Push registers and call cmd_parser

Note: 1) These addresses only relate to version 1.0 of the MSC1210 Boot ROM

Autobaud

Summary: void autobaud(void);

Description: Sets up UART0 for serial communications. Waits for a carriage return (0x0A) and calculates the baud rate of the host device. UART0 is then setup with Timer 2 for that baud rate. Compatible with C procedure printf/scanf.

Before serial communications, the baud rate needs to be set or autobaud needs to be called to perform that function.

If autobaud is to be called again, the following registered need to be cleared first: SCON0, TR2, T2CON, THL2

Inputs: None

Return Value: None. Sets timer 2 values for the correct baud rate.

See Also: putchar, putok, putstring, rxbyte_echo, rx_hex_echo, rx_hex_int_echo, rx_hex_rev_echo, tx_byte, tx_hex

Example:

C Program

```
extern void autobaud(void);
#include <stdio.h>
main()
{
    autobaud();
    printf("Hello, World");
    while(1) ;
}
```

Assembly Program

```
AUTOBAUD EQU 0FFF3H
PUTSTR EQU 0FFD5H
World: DB 'Hello World', 0DH,0AH,0

TEST: LCALL AUTOBAUD
      MOV R6,#HIGH World
      MOV R7,#LOW World
      LCALL PUTSTR
END
```

Caution

Before autobaud can be called a second time the following registers must be cleared.

```
SCON0 = 0;
TR2 = 0;
T2CON = 0;
THL2 = 0;
```

Data_x_c_read

Summary:

char data_x_c_read (int faddr, char fdm);

Description:

Read byte from Program/Data Flash

For reading flash data/code memory a pointer can be used (recommended) instead of this routine.

Inputs:

R6: [faddr] Flash address high byte

R7: [faddr] Flash address low byte

R4: R5 are not used but must be passed to correctly set R3

R3: [fdm] if fdm is equal to 1 then Data Memory, otherwise Program memory

Return Value:

R7: Byte from Flash address

See Also:

faddr_data_read, page_erase

Example:
C Program

```
#include <stdio.h>
extern char data_x_c_read(int faddr, int, char fdm);
extern void tx_hex(char t);
extern void autobaud(void);
main()
{
    char xdata * data dVal;    // Pointer to Xdata
    char code * data cVal;     // Pointer to Code
    dVal = cVal = 0x0043;
    autobaud();
    printf("From 0x0043 (Data/Prog Flash)\n");
    tx_hex(data_x_c_read(0x0043, 0, 1)); // Read from data Mem.
    tx_hex(data_x_c_read(0x0043, 0, 0)); // Read from Prog. Mem.
    tx_hex(*dVal); tx_hex(*cVal);
    while(1) ;
}
```

Assembly Program

```
AUTOBAUD    EQU    0FFF3H
TX_HEX      EQU    0FFE5H
PUTSTR      EQU    0FFD5H
X_C_READ    EQU    0FFE1H

TEST:       LCALL          AUTOBAUD
            MOV            R6,#HIGH Mesg
            MOV            R7,#LOW Mesg
            LCALL          PUTSTR
            MOV            R6,#04H
            MOV            R7,#43H
            MOV            R3,#1                ; FLASH DATA MEMORY
            LCALL          X_C_READ
            LCALL          TX_HEX                ; TRANSMIT FLASH DATA BYTE
            MOV            R6,#04H
            MOV            R7,#43H
            MOV            R3,#0                ; FLASH PROGRAM MEMORY
            LCALL          X_C_READ
            LCALL          TX_HEX                ; TRANSMIT FLASH PROGRAM BYTE
            AJMP           $

Mesg:       DB 'From 0x0443 (Data/Prog Flash)', 0DH,0AH,0
CSEG AT 0807FH
            DB 0FEH                ; 1K Data Flash
            END
```

Faddr_data_read

Summary: char faddr_data_read(char address);

Description: Read a byte from the 128 bytes of configuration memory

Inputs: R7: [address] PS1 address

Return Value: R7: Byte at that address

See Also: data_x_c_read, page_erase

Example:

C Program

```
extern char faddr_data_read(char address);
extern void autobaud(void);
extern void tx_hex(char);

main()
{
    autobaud();
    tx_hex(faddr_data_read(0x7E)); // Print HCR1 value
    while(1) ;
}
```

Assembly Program

```
FADDR_R      EQU    0FFDFH
AUTOBAUD     EQU    0FFF3H
TX_HEX       EQU    0FFE5H

TEST:
    LCALL    AUTOBAUD
    MOV      R7,#07EH      ; ADDRESS OF HCR1
    LCALL    FADDR_R       ; VALUE RETURNED IN R7
    LCALL    TX_HEX        ; HEX VALUE OF R7 SENT
    AJMP     $
END
```

Page_erase

Summary: char page_erase (int faddr, char fdata, char fdm);

Description: Erases 128 bytes located at 16-bit address faddr. The least significant 7 bits of the address are ignored. fdata is required, but the value is ignored.

Inputs:
R6: [faddr] flash address high byte
R7: [faddr] flash address low byte
R5: [fdata] (value ignored)
R3: [fdm] (0: Flash Program Memory, 1: Flash Data Memory)

Return Value: R7: (0: pass, non-zero: fail)

See Also: data_x_c_read, faddr_data_read

Example:

C Program

```
extern void put_string (char code *string);
extern void autobaud (void);
extern char page_erase (int faddr, char dat, char fdm);
extern void tx_hex (char t);
main()
{
    autobaud();
    put_string("Page Erase Result: ");
    tx_hex(page_erase(0x400, 0, 1));    // first page data memory
    while(1) ;                        // 00=success
}
```

Assembly Program

```
AUTOBAUD EQU 0FFF3H
TX_HEX EQU 0FFE5H
PAGE_ERASE EQU 0FFD7H
PUTSTRING EQU 0FFD5H

TEST:    LCALL    AUTOBAUD
        MOV      R6,#HIGH Msg
        MOV      R7,#LOW Msg
        LCALL    PUTSTRING
        MOV      R7,#04H          ; Address = 0x0400
        MOV      R6,#00H
        MOV      R3,1             ; Select Data memory
        LCALL    PAGE_ERASE
        LCALL    TX_HEX           ; Output Results of Erase 0=good
        AJMP     $
Msg:     DB       0DH,0AH,'Page Erase Result: ',0
END
```

Putcr

Summary: void putcr(void);

Description: Writes a carriage return (0x0D) & line feed (0x0A) to UART0

Inputs: None

Return Value: None

See Also: autobaud, tx_byte

Example:

C Program

```
extern void putcr(void);
extern void put_string(char code *string);
extern void autobaud(void);

main()
{
    autobaud();
    putcr();
    put_string("Hello,");
    putcr();
    put_string("World");
    while(1) ;
}
```

Assembly Program

```
PUTSTR    EQU    0FFD5H
PUTCR     EQU    0FFFDH
AUTOBAUD  EQU    0FFF3H

TEST:     LCALL   AUTOBAUD
          MOV     R6,#HIGH World      ; "Hello World "
          MOV     R7,#LOW World
          LCALL   PUTSTR
          LCALL   PUTCR
          MOV     R6,#HIGH World      ; "Hello World "
          MOV     R7,#LOW World
          LCALL   PUTSTR
          AJMP    $
World:    DB 'Hello World', 0DH,0AH,0
END
```

Putok

Summary: void putok(char passfail);

Description: Checks the contents of R7 (pass_fail), if it is equal to 0x00 then it outputs 'ok' to UART0, otherwise it outputs 'x' to UART0.

Inputs: R7: [passfail] (0: pass, 1: fail)

Return Value: None

See Also: tx_byte, autobaud

Example: C Program

```
extern void putok(char passfail);
extern void autobaud(void);
extern char page_erase(int faddr, char fdata, char fdm);
#include <stdio.h>

main()
{
    autobaud();
    printf("Testing Page Erase\n");
    putok(page_erase(0x2345, 0, 0));    // display page_erase result
    printf("Done");
    while(1) ;
}
```

Assembly Program

```
PUTOK      EQU    0FFE7H
PAGE_ERASE EQU    0FFD7H
PUTSTR     EQU    0FFD5H
AUTOBAUD   EQU    0FFF3H

TEST:      LCALL AUTOBAUD
           MOV     R6, #HIGH Msg      ; "Page Erase Result: "
           MOV     R7, #LOW Msg
           LCALL PUTSTR
           MOV     R7, #04H           ; Address = 0x0400
           MOV     R6, #00H
           MOV     R3, 1               ; Select Data memory
           LCALL PAGE_ERASE
           LCALL PUTOK
           AJMP    $
Msg:        DB     0DH, 0AH, 'Page Erase Result: ', 0
CSEG AT 0807FH
DB         0FEH           ; 1K Data Flash

END
```

Putspace1, putspace2, putspace3, putspace4

Summary: void putspace1(void)

Description: Writes one (two/three/four) space character(s) (0x20) to UART0

Inputs: None

Return Value: None

See Also: autobaud, tx_byte

Example:

C Program

```
extern void putspace1(void);    /* or putspace2, 3 or 4 */
extern void autobaud(void);
extern void putok(char);

main()
{
    autobaud();
    putok(0);
    putspace1();                /* or putspace2, 3 or 4 */
    putok(1);
    while(1) ;
}
```

Assembly Program

```
PUTOK      EQU    0FFE7H
PUTSPACE4  EQU    0FFF5H
AUTOBAUD   EQU    0FFF3H

TEST:      LCALL  AUTOBAUD
           MOV    R7,#0
           LCALL  PUTOK
           LCALL  PUTSPACE4
           MOV    R7,#0
           LCALL  PUTOK
           AJMP   $

END
```


Put_string

- Summary:** void put_string(char code * data msg);
Writes a null terminated character string to UART0
- Description:** Addresses 0x0A and 0x0B are used by put_string.
- Inputs:** R6: [msg] high byte of address
R7: [msg] low byte of address
- Return Value:** None
- See Also:** autobaud, tx_byte

Example:

C Program

```
extern void put_string(char code *string);
extern void autobaud(void);
data char ROM_temp0 _at_ 0x000a;    // Restrict compiler so it
data char ROM_temp1 _at_ 0x000b;    // doesn't use these addresses

main()
{
    autobaud();
    put_string("Hello, World");
    while(1) ;
}
```

Assembly Program

```
PUTSTR      EQU    0FFD5H
AUTOBAUD    EQU    0FFF3H

TEST:       LCALL AUTOBAUD
            MOV    R6,#HIGH World
            MOV    R7,#LOW World
            LCALL PUTSTR
            AJMP   $
World:      DB     'Hello World', 0DH,0AH,0
END
```

Caution:

The put_string routine was designed to print strings that are referenced when the Boot ROM is located at 0x0000 and also at 0xF800. This means that it forces the location of the string to match the same 2K segment the program is located in. This will lead to strange behavior if the string address is located in a different 2K segment. For this reason it is suggested that you use the following code instead:

```
void putstring(char code * data msg){
    while (*msg != 0) {
        tx_byte((unsigned char) *msg);
        if (*msg=='\n') tx_byte('\r');
        msg++;
    }
}
```

Rx_byte

Summary: char rx_byte (void);

Description: Receives a byte from UART0
It waits for RI bits, reads value and clears RI bit.

Inputs: None

Return Value: Byte received from UART0 in R7

See Also: autobaud, tx_byte

Example:

C Program

```
extern char rx_byte(void);
extern void autobaud(void);
extern void tx_byte(char x);

main()
{
    autobaud();
    while(1)
        tx_byte(rx_byte());    // Echo Character Received.
}
```

Assembly Program

```
TXBYTE    EQU    0FFE3H
RXBYTE    EQU    0FFE9H
AUTOBAUD   EQU    0FFF3H

          LCALL AUTOBAUD
TEST:     LCALL RXBYTE
          LCALL TXBYTE
          AJMP  TEST
END
```

Rx_byte_echo

Summary: char rx_byte_echo(void)

Description: Receives a character from UART0 and then echos the character if bit PSW.F1 is zero.

Inputs: None

Return Value: Received byte

See Also: autobaud, tx_byte, rx_byte

Example: C Program

```
extern char rx_byte_echo(void);
extern void autobaud(void);

main()
{
    char inchar;
    autobaud();
    inchar = rx_byte_echo();
}
```

Assembly Program

```
AUTOBAUD EQU 0FFF3H
RXECHO EQU 0FFEBH

LCALL AUTOBAUD
TEST: LCALL RXECHO
AJMP TEST

END
```

Rx_hex_echo

Summary: char rx_hex_echo (void);

Description: Receives two ASCII characters from UART0, echos each character, also interprets them as Hex characters (high character first) and returns a byte of that value.

Inputs: None

Return Value: Byte with value of two received Hex characters.

See Also: autobaud, rx_byte_echo, rx_hex_int_echo, rx_hex_rev_echo

Example:

C Program

```
#include <stdio.h>
extern char rx_hex_echo(void);
extern void autobaud(void);

main()
{
    char value;           // signed value
    autobaud();
    while (1){
        value = rx_hex_echo();
        printf(" [%bd]\n",value);
    }
}
```

Assembly Program

```
AUTOBAUD EQU 0FFF3H
RXHEXECHO EQU 0FFEDH
TXBYTE EQU 0FFE3H

        LCALL AUTOBAUD
TEST:    LCALL RXHEXECHO
        LCALL TXBYTE
        AJMP TEST

END
```

Rx_hex_int_echo

Summary: int rx_hex_int_echo(void);

Description: Receives four ASCII characters from UART0, echos each character, also interprets them as Hex characters (high characters first) and returns an address pointer (16-bits) of that value.

Inputs: None

Return Value: R6: integer high byte
R7: integer low byte

See Also: autobaud, rx_byte_echo, rx_hex_echo, rx_hex_rev_echo

Example:

C Program

```
#include <stdio.h>
extern int code *rx_hex_int_echo(void);
extern void autobaud(void);

main()
{
    char code * data ptr;
    autobaud();
    while (1){
        ptr = rx_hex_int_echo();
        printf("    Normal: [%bx] @ %x\n",*ptr, (int)ptr);
    }
}
```

Assembly Program

```
RXINTECHO EQU    0FFEFH
AUTOBAUD  EQU    0FFF3H

                LCALL    AUTOBAUD
TEST:         LCALL    RXINTECHO
                MOV      DPH,R7           ; Save Value
                MOV      DPL,R6
                AJMP     $
END
```

Rx_hex_rev_echo

Summary: `int *rx_hex_rev_echo(void);`

Description: Receives four ASCII characters from UART0, echos each character, also interprets them as Hex characters (low nibble first) and returns an address pointer (16-bits) of that value. This can be useful for pushing a value on the stack.

Inputs: None

Return Value: R6: pointer low byte
R7: pointer high byte

See Also: autobaud, rx_byte_echo, rx_hex_echo, rx_hex_int_echo

Example:

C Program

```
#include <stdio.h>
extern int code *rx_hex_int_echo(void);
extern int code *rx_hex_rev_echo(void);
extern void autobaud(void);

main()
{
    char code * data ptr;
    autobaud();
    while (1){
        ptr = rx_hex_int_echo();
        printf("    Normal: [%bx] @ %x\n",*ptr, (int)ptr);
        ptr = rx_hex_rev_echo();
        printf("  Reversed: [%bx] @ %x\n",*ptr, (int)ptr);
    }
}
```

Assembly Program

```
RXREVECHO EQU    0FFF1H
AUTOBAUD  EQU    0FFF3H

                LCALL    AUTOBAUD
TEST:         LCALL    RXREFECHO
                MOV      DPH,R6            ; Save Value
                MOV      DPL,R7
                MOV      A,@DPTR
                AJMP     $
END
```

Tx_byte

Summary: void tx_byte(char x);

Description: Writes a byte to UART0
Wait for TI to be set, clear bit and load sbuf with R7

Inputs: R7: data byte

Return Value: None

See Also: autobaud, tx_hex, rx_byte

Example:

C Program

```
extern void autobaud(void);
extern void tx_byte(char x);

main()
{
    autobaud();
    tx_byte('A');
}
```

Assembly Program

```
AUTOBAUD EQU 0FFF3H
TXBYTE EQU 0FFE3H
RXBYTE EQU 0FFE9H

TEST: LCALL AUTOBAUD
      LCALL RXBYTE
      LCALL TXBYTE
      AJMP TEST
END
```

Tx_hex

Summary: void tx_hex(char hvalue);

Description: Writes a two characters to UART0 that represent the hex equivalence of the input byte

Inputs: R7: [hvalue] data byte

Return Value: None

See Also: autobaud, tx_byte, rx_hex_byte

Example:

C Program

```
extern void autobaud(void);
extern void tx_hex(char hvalue);

main()
{
    autobaud();
    tx_hex(0x32);
}
```

Assembly Program

```
AUTOBAUD    EQU        0FFF3H
RXBYTE      EQU        0FFE9H
TXHEX       EQU        0FFE5H

                LCALL    AUTOBAUD
TEST:        LCALL    RXBYTE
                LCALL    TXHEX
                AJMP     TEST
END
```


write_flash

Summary: Write Flash routine, not callable from a C program

Description: Write a byte to Flash memory. This is the fastest Flash write routine. It writes 1 byte from the accumulator to the flash memory address specified by DPH:DPL. It starts by using `MOVX @DPTR,ACC` and exits when the Flash Busy bit in register FMCON is low. The duration of the procedure is about 60 uS which is the flash program time. Code or data write is selected by the MXWS bit in MWS sfr.

Inputs:
DPH: address high byte
DPL: address low byte
A: data byte

Return Value: None

See Also: page_erase, write_flash_byte, write_flash_chk

Example: C Program

None

Assembly Program

```
WRITEFLASH EQU 0FFD9H

TEST:      MOV  DPH,#12H
           MOV  DPL,#34H
           MOV  A,#0AAH
           LCALL WRITEFLASH
           AJMP $

END
```

Write_flash_byte

Summary: void write_flash_byte(int faddr, char fdata);

Description: The value in R6:R7 is moved to DPTR, R5 is moved to the accumulator and then the write_flash procedure described above is called. The duration of this procedure is about 50 uS (30-40 Flash program time + 10 uS for flash address setup + overhead). The FTCON register should be loaded before this routine is called.

Inputs: R6: faddr high byte
R7: faddr low byte
R5: fdata byte

Return Value: None

See Also: page_erase, write_flash, write_flash_chk

Example:

C Program

```
Extern void put_string(char code *string);
Extern void autobaud(void);
Extern void flash_write(int adr, char data);

Main()
{
    autobaud();
    write_flash(0x1234, 0x0ab);
    put_string("Value Written");
    while (1) {;}
}
```

Assembly Program

```
PUTSTR      EQU      0FFD5H
WRITEBYTE   EQU      0FFDDH
AUTOBAUD    EQU      0FFF3H

                LCALL    AUTOBAUD
TEST:        MOV       R6,#012
                MOV       R7,#034
                MOV       A,0ABH
                LCALL    WRITEBYTE
                MOV       R6,#HIGH Msg
                MOV       R7,#LOW Msg
                LCALL    PUTSTR
                AJMP      $
Msg:         DB         'Value Written', 0DH,0AH,0
END
```

Write_flash_chk

Summary:

char write_flash_chk(int faddr, char fdata, char fdm);

Description:

Writes data byte (fdata) to address (faddr) in either the flash program (fdm:0) memory or data (fdm:1) memory. This calls the write_flash procedure, then reads back from the same address to verify the results. If data matches the return value is 0, otherwise it will return a value of 1 for invalid comparison.

Inputs:

R6: faddr high byte
R7: faddr low byte
R5: fdata byte
R3: fdm (0: Flash Program Memory, 1:Flash Data Memory)

Return Value:

R7: (Pass:0, Fail: 1)

See Also:

page_erase, write_flash, write_flash_chk

Example:

C Program

```
extern void put_string(char code *string);
extern void autobaud(void);
extern char write_flash_chk(int adr, char dat, char fdm);
extern void putok(char);
#define DATA 1

Main()
{
    autobaud();
    putok(write_flash_chk(0x0034, 0x0ab, DATA));
    put_string(" Value Written (x=fail)\n");
    while (1) {;}
}
```

Assembly Program

```
AUTOBAUD    EQU        0FFF3H
PUTSTR      EQU        0FFD5H
WRBYTECHK   EQU        0FFDBH
TX_HEX      EQU        0FFE5H
            LCALL       AUTOBAUD
            MOV         0FBH, #0AH    ;Write timing for 11 MHz crystal
TEST:       MOV         R6, #004H
            MOV         R7, #034H
            MOV         R5, #0ABH
            MOV         R3, #01        ;Data Memory
            LCALL       WRBYTECHK
            LCALL       TX_HEX
            MOV         R6, #HIGH Msg
            MOV         R7, #LOW Msg
            LCALL       PUTSTR
            AJMP        $
Msg:         DB          ' = 00(Verified), 01(Error)', 0DH, 0AH, 0
CSEG AT 0807FH
            DB          0FEH    ; 1K Data Flash

END
```

Appendix A. ROM.A51

This file defines the ROM routines and provides the entry point addresses for use by the linker. Some names have changed.

```

PUBLIC      _put_string      ; void put_string(char code *string);
PUBLIC      _page_erase     ; char page_erase (int faddr, char fdata, char fdm)
PUBLIC      write_flash     ; char write_flash (int faddr, char fdata, char fdm)
PUBLIC      _write_flash_chk ; char write_flash_chk (int faddr, char fdata, char fdm)
PUBLIC      _write_flash_byte ; char write_flash_byte (int faddr, char fdata, char fdm)
PUBLIC      _faddr_data_read ; char faddr_data_read(char);
PUBLIC      _data_x_c_read   ; char data_x_c_read(int addr);
PUBLIC      _tx_byte         ; void tx_byte(char);
PUBLIC      _tx_hex          ; void tx_hex(char);
PUBLIC      _putok           ; void putok(void);
PUBLIC      rx_byte          ; char rx_byte(void);
PUBLIC      rx_byte_echo     ; char rx_byte_echo(void);
PUBLIC      rx_hex_echo      ; char rx_hex_echo(void);
PUBLIC      rx_hex_int_echo   ; int rx_hex_int_echo(void);
PUBLIC      rx_hex_rev_echo   ; int rx_hex_rev_echo(void);
PUBLIC      autobaud         ; void autobaud(void);
PUBLIC      putspace4        ; void putspace4(void);
PUBLIC      putspace3        ; void putspace3(void);
PUBLIC      putspace2        ; void putspace2(void);
PUBLIC      putspace1        ; void putspace1(void);
PUBLIC      putcr            ; void putcr(void);

      CSEG AT 0FFD5H
_put_string:      ; void put_string(char *string);
      CSEG AT 0FFD7H
_page_erase:      ; char page_erase (int faddr, char fdata, char fdm)
      CSEG AT 0FFD9H
write_flash:      ; DPTR = address, acc = data (Not callable by C programs)
      CSEG AT 0FFDBH
_write_flash_chk: ; char write_flash_chk (int faddr, char fdata, char fdm)
      CSEG AT 0FFDDH
_write_flash_byte: ; char write_flash_byte (int faddr, char fdata, char fdm)
      CSEG AT 0FFDFH
_faddr_data_read: ; char faddr_data_read(char);
      CSEG AT 0FFE1H
_data_x_c_read:   ; char data_x_c_read(int addr);
      CSEG AT 0FFE3H
_tx_byte:         ; void tx_byte(char);
      CSEG AT 0FFE5H
_tx_hex:          ; void tx_hex(char);
      CSEG AT 0FFE7H
_putok:           ; void putok(void);
      CSEG AT 0FFE9H
rx_byte:          ; char rx_byte(void);
      CSEG AT 0FFEBH
rx_byte_echo:     ; char rx_byte_echo(void);
      CSEG AT 0FFEDH
rx_hex_echo:      ; char rx_hex_echo(void);
      CSEG AT 0FFEFH
rx_hex_int_echo:   ; char rx_hex_int_echo(void);
      CSEG AT 0FFF1H
rx_hex_rev_echo:   ; char rx_hex_rev_echo(void);
      CSEG AT 0FFF3H
autobaud:         ; void autobaud(void);
      CSEG AT 0FFF5H
putspace4:        ; void putspace4(void);
      CSEG AT 0FFF7H
putspace3:        ; void putspace3(void);
      CSEG AT 0FFF9H
putspace2:        ; void putspace2(void);
      CSEG AT 0FFFBH
putspace1:        ; void putspace1(void);
      CSEG AT 0FFFDH
putcr:            ; void putcr(void);

      END

```

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265