

Michael Tangy

ECE 381 – Spring 2015

Lab 5

RS-232 Interfacing

The purpose of this lab was for us to gain an understanding of how RS-232 Interfacing works in general as well as how to implement it with our PSoC microcontrollers. The lab consisted of us asking the user to either write or read from 4 different string, while reading the strings the program gives you the option of printing the strings to either the prompt window or the LCD display. The program also allows the user to backspace and delete characters while writing the strings, it also beeps if the user try's to enter to many characters.

Parts Needed:

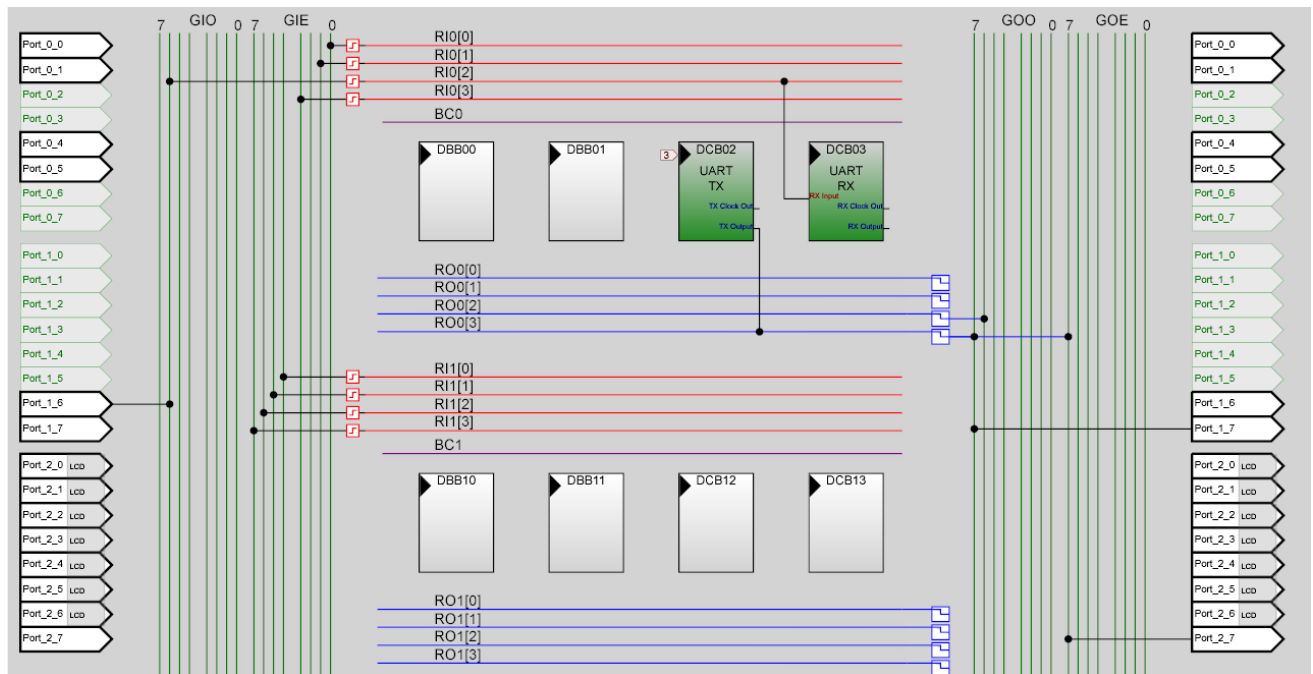
- 1- Psoc Microcontroller
- 1- DB-9 to DB-9 cable or DB-9 to USB serial cable
- 1- PSoC LCD Display

Wiring:

This lab doesn't require any wiring besides hooking the LCD display to port two which should already be done. The UART cable also needs to be hooked up to the computer it's communicating with. This requires hooking the cable up to the PSoC's DB9 input and then hooking the other end to your computers USB or DB9 input.

Pin-Out Configuration:

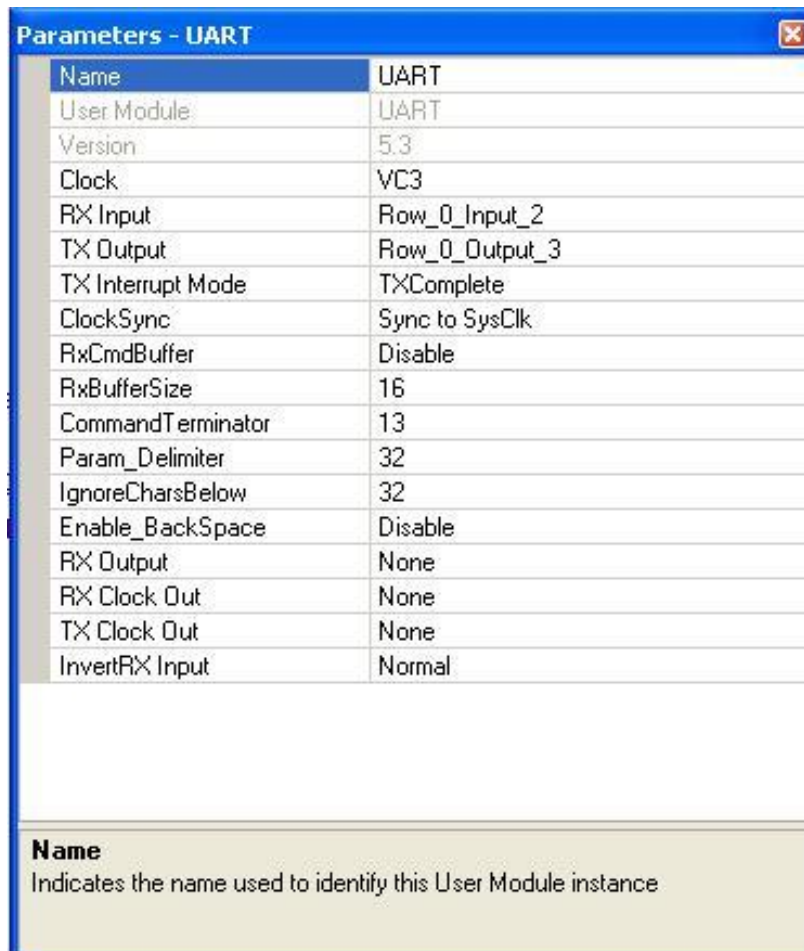
After hooking your PSoC up with your computer you need to assign your PSoC port 1 pin 6 (P1[6]) to the UART's RX line and port 1 pin 7 (P1[7]) to the UART's TX line, also change the variable names at those pins. This allows you to view the binary bit streams you're sending and receiving from your PSoC. This can be done from the chip level view of your project by shift-clicking on the UART modules outputs and assigning them to the appropriate pins. Below is how the chip level view of your project should look.



Next you need to configure a clock source to drive the UART. Since the required baud rate is 9600 we need to set our source clock to three times that (76,800Hz). In the UART data sheet we discovered the best way to do this was to divide the system clock by 8 and 39 and set it to VC3 as shown below:

Global Resources - lab5	
Power Setting [Vcc	5.0V / 24MHz
CPU Clock	SysClk/8
32K Select	Internal
PLL Mode	Disable
Sleep Timer	512_Hz
VC1= SysClk/N	8
VC2= VC1/N	3
VC3 Source	VC1
VC3 Divider	39
SysClk Source	Internal
SysClk*2 Disable	No
Analog Power	SC On/Ref Low
Ref Mux	(Vdd/2)+/-BandGap
AGndBypass	Disable
Op-Amp Bias	Low
A Buff Power	Low
SwitchModePump	OFF
Trip Voltage [LVD (: 4.81V (5.00V)	
LVDThrottleBack	Disable
Watchdog Enable	Disable

Next you need to change the properties of your UART module to make sure everything's configured properly. The clock should be set to VC3 and the RxCmdBuffer should be disabled the UART properties window should look like this:



The screenshot shows a window titled "Parameters - UART" with a table of configuration parameters. The parameters are listed in two columns: Name and Value. The values are as follows:

Name	Value
User Module	UART
Version	5.3
Clock	VC3
RX Input	Row_0_Input_2
TX Output	Row_0_Output_3
TX Interrupt Mode	TXComplete
ClockSync	Sync to SysClk
RxCmdBuffer	Disable
RxBufferSize	16
CommandTerminator	13
Param_Delimiter	32
IgnoreCharsBelow	32
Enable_BackSpace	Disable
RX Output	None
RX Clock Out	None
TX Clock Out	None
InvertRX Input	Normal

Below the table, there is a section for the "Name" property, which indicates the name used to identify this User Module instance.

Next you should make sure your LCD is set to port two through its module properties. The Real Term software used in interface with the UART needs to be configured with its Baud rate set to 9600 and its port set 1 if your UART is connected to COMA or COM1 or set to port 2 if its connect to COMB or COM2.

Software Description:

After configuring the pin-out settings, chip level settings and the real term settings you're ready to write your program. All your variables, pointers and constant strings should be declared globally above your main method. Your main method should initialize your strings to null as well as start you UART and set its parity to none. From there your program should print your initial message (asking if they want to R/W) to the UART's console and read in the character they input, then the program should enter an infinite while loop. Your infinite while loop houses all your nested if-statements that direct your program to the correct place depending on what the user inputs. The first if-statement is going to check if there is an inputted variable. Next it's going to print your second message (asking which string) and save the inputted character. After that, the program checks your first input and directs you to either the read or write portions of the program. The read portion contains two different switch statements one that prints to the LCD and one that prints to the console. Each of those cases have their own switch statements that check the second input entered and prints the appropriate string. The write portion has a similar switch statement that checks the second inputted character and directs you to the appropriate string to write to. In order to do this each case should have an infinite while that reads until a character is inputted then checks if that character exceeds the max of 16 (the 17th is left NULL) is so it writes the bell to the console. Next it checks if the character is "enter" if so it prints the initial message and jumps to the first infinite while. After that it checks for a "backspace" or "delete" character and sets the previous character to a blank space if it is. If neither of those conditions are met it assigns the inputted character to the first element of the string prints the string and increments the index variable of the string.

CODE:

```
#include <m8c.h>           // part specific constants and macros
#include "PSoCAPI.h"       // PSoC API definitions for all User Modules
#include "string.h"

char string0[17];          // string decleration
char string1[17];
char string2[17];
char string3[17];
//void Start(void);

char buffer= '\0';        //buffer variable initializtions
char buffer2= '\0';
char buffer3= '\0';
char buffer4= '\0';
char buffer5= '\0';
int placeCounter = 0;
int count;

const char promptMessage1[67]= "Would you like to read or write a string? R
for read & W for write";    //const strings and their pointers
const char promptMessage2[28]= "Not a valid input. Goodbye";
const char promptMessage3[55]= "Which string? Type a for 1, b for 2, c for 3,
d for 4";
const char promptMessage4[65]= "Do you want to print to Port or LCD? p for
port & l for LCD";
const char* promptMessage1P = promptMessage1;
const char* promptMessage2P = promptMessage2;
const char* promptMessage3P = promptMessage3;
const char* promptMessage4P = promptMessage4;

void main(void)
{

//UART_CPutString("\r\n1\r\n");
string0[0] = '\0';    // string initilizations
string1[0] = '\0';
string2[0] = '\0';
string3[0] = '\0';

LCD_Start();
LCD_Position(0,0);
LCD_PrCString("                ");
LCD_Position(1,0);
LCD_PrCString("                ");

//LCD_PrCString(promptMessage1P);

UART_Start(UART_PARITY_NONE);    // start UART function

UART_CPutString(promptMessage1P); // initial prompt message
//UART_CPutString("\r\n");
UART_CPutString("\r\n");
```

```

UART_PutCRLF();
while(1)
{
Start:
buffer= '\0';           //re-initialize buffer vars
buffer2= '\0';
buffer3= '\0';
buffer4= '\0';
buffer5= '\0';
placeCounter = 0;

buffer = UART_cReadChar();    // read in first input char


if(buffer != 0x00)           // waits until first char is enter
{

UART_CPutString(promptMessage3P); // "Which String?"   prompt
UART_CPutString("\r\n");
UART_CPutString("\r\n");
buffer2 = UART_cReadChar();    // reads in second char
while (buffer2 == 0x00){buffer2 = UART_cReadChar();}
if(buffer2 != 0x00)    // waits until second char is enter
{
if(buffer == 'r')           // read string portion
{

UART_CPutString(promptMessage4P);    // print to console or LCD?

UART_CPutString("\r\n");
buffer4 = UART_cReadChar();    // reads third input char
while (buffer4 == 0x00){buffer4 = UART_cReadChar();} // waits until input

if (buffer4 == 'l'){ //print to LCD portion
switch(buffer2)
{
case 'a':
LCD_Position(0,0);
LCD_PrCString("                ");
LCD_Position(1,0);
LCD_PrCString("                ");
LCD_Position(0,0);
LCD_PrCString(string0);
buffer3 = UART_cReadChar();
while (buffer3 == 0x00){buffer3 = UART_cReadChar();}
if (buffer3 == 0x0d){
UART_CPutString("\r\n");
UART_CPutString("\r\n");
UART_CPutString(promptMessage1P);
UART_CPutString("\r\n");
UART_CPutString("\r\n");
goto Start;
}
break;

```

```

case 'b':
LCD_Position(0,0);
LCD_PrCString("                ");
LCD_Position(1,0);
LCD_PrCString("                ");
LCD_Position(0,0);
LCD_PrString(string1);
buffer3 = UART_cReadChar();
while (buffer3 == 0x00){buffer3 = UART_cReadChar();}
if (buffer3 == 0x0d){
UART_CPutString("\r\n");
UART_CPutString("\r\n");
UART_CPutString(promptMessage1P);
UART_CPutString("\r\n");
UART_CPutString("\r\n");
goto Start;
}
break;

case 'c':
LCD_Position(0,0);
LCD_PrCString("                ");
LCD_Position(1,0);
LCD_PrCString("                ");
LCD_Position(0,0);
LCD_PrString(string2);
buffer3 = UART_cReadChar();
while (buffer3 == 0x00){buffer3 = UART_cReadChar();}
if (buffer3 == 0x0d){
UART_CPutString("\r\n");
UART_CPutString("\r\n");
UART_CPutString(promptMessage1P);
UART_CPutString("\r\n");
UART_CPutString("\r\n");
goto Start;
}
break;

case 'd':
LCD_Position(0,0);
LCD_PrCString("                ");
LCD_Position(1,0);
LCD_PrCString("                ");
LCD_Position(0,0);
LCD_PrString(string3);
buffer3 = UART_cReadChar();
while (buffer3 == 0x00){buffer3 = UART_cReadChar();}
if (buffer3 == 0x0d){
UART_CPutString("\r\n");
UART_CPutString("\r\n");
UART_CPutString(promptMessage1P);
UART_CPutString("\r\n");
UART_CPutString("\r\n");
goto Start;
}
}

```



```

default:
UART_CPutString(promptMessage2P);
}
}

if (buffer4 == 'p'){                                     //print to console portion
switch(buffer2)
{
case 'a':
UART_CPutString("\r\n");
UART_PutString(string0);
buffer3 = UART_cReadChar();
while (buffer3 == 0x00){buffer3 = UART_cReadChar();}
if (buffer3 == 0x0d){
UART_CPutString("\r\n");
UART_CPutString("\r\n");
UART_CPutString(promptMessage1P);
UART_CPutString("\r\n");
UART_CPutString("\r\n");
goto Start;
}

break;

case 'b':

UART_CPutString("\r\n");
UART_PutString(string1);
buffer3 = UART_cReadChar();
while (buffer3 == 0x00){buffer3 = UART_cReadChar();}
if (buffer3 == 0x0d){
UART_CPutString("\r\n");
UART_CPutString("\r\n");
UART_CPutString(promptMessage1P);
UART_CPutString("\r\n");
UART_CPutString("\r\n");
goto Start;
}

break;

case 'c':

UART_CPutString("\r\n");
UART_PutString(string2);
buffer3 = UART_cReadChar();
while (buffer3 == 0x00){buffer3 = UART_cReadChar();}
if (buffer3 == 0x0d){
UART_CPutString("\r\n");
UART_CPutString("\r\n");
UART_CPutString(promptMessage1P);
UART_CPutString("\r\n");
UART_CPutString("\r\n");
goto Start;
}

break;

```

```

case 'd':

UART_CPutString("\r\n");
UART_PutString(string3);
buffer3 = UART_cReadChar();
while (buffer3 == 0x00){buffer3 = UART_cReadChar();}
if (buffer3 == 0x0d){
UART_CPutString("\r\n");
UART_CPutString("\r\n");
UART_CPutString(promptMessage1P);
UART_CPutString("\r\n");
UART_CPutString("\r\n");
goto Start;
}

break;

default:
UART_CPutString(promptMessage2P);
}
}

}

if(buffer == 'w') // write to string portion
{
buffer2 = UART_cReadChar(); // reads second inputted char
while (buffer2 == 0x00){buffer2 = UART_cReadChar();}
switch(buffer2)
{
case 'a':
UART_PutChar(0x3e);
while(1){
buffer3 = UART_cReadChar(); // reads inputted char
while (buffer3 == 0x00){buffer3 = UART_cReadChar();} // waits until inputted
if (placeCounter<16){ // checks its not
over 17 chars long
while (buffer3 == 0x00){buffer3 = UART_cReadChar();}

if (buffer3 == 0x0d){ //checks if enter was inputted
UART_CPutString("\r\n");
UART_CPutString("\r\n");
UART_CPutString(promptMessage1P);
UART_CPutString("\r\n");
UART_CPutString("\r\n");
goto Start;
}
if (buffer3 == 0x08 || buffer3 ==0x7f){ // checks if backspace was inputted
placeCounter--;
string0[placeCounter] = ' ';

}
string0[placeCounter] = buffer3; // saves to string
UART_PutChar(string0[placeCounter]); //mirrors back to console
placeCounter++; // goes to next char in string

```

```

} // end if

} // end outer while

break;

case 'b':
UART_PutChar(0x3e);
while(1){

buffer3 = UART_cReadChar();
while (buffer3 == 0x00){buffer3 = UART_cReadChar();}
if (placeCounter<16){
while (buffer3 == 0x00){buffer3 = UART_cReadChar();}

if (buffer3 == 0x0d){
UART_CPutString("\r\n");
UART_CPutString("\r\n");
UART_CPutString(promptMessage1P);
UART_CPutString("\r\n");
UART_CPutString("\r\n");
goto Start;
}

if (buffer3 ==0x08){
placeCounter--;
string1[placeCounter] = ' ';

}

string1[placeCounter] = buffer3;
UART_PutChar(string1[placeCounter]);
placeCounter++;

}else{UART_PutChar(0x07);}
}

break;

case 'c':
UART_PutChar(0x3e);
while(1){

buffer3 = UART_cReadChar();
while (buffer3 == 0x00){buffer3 = UART_cReadChar();}
if (placeCounter<16){
while (buffer3 == 0x00){buffer3 = UART_cReadChar();}

if (buffer3 == 0x0d){
UART_CPutString("\r\n");
UART_CPutString("\r\n");
UART_CPutString(promptMessage1P);
UART_CPutString("\r\n");
UART_CPutString("\r\n");
goto Start;
}
}

```

```

if (buffer3 == 0x08 || buffer3 ==0x7f){
placeCounter--;
string2[placeCounter] = ' ';

}

string2[placeCounter] = buffer3;
UART_PutChar(string2[placeCounter]);
placeCounter++;

}else{UART_PutChar(0x07);}
}
break;

case 'd':
UART_PutChar(0x3e);
while(1){
buffer3 = UART_cReadChar();
while (buffer3 == 0x00){buffer3 = UART_cReadChar();}
if (placeCounter<16){
while (buffer3 == 0x00){buffer3 = UART_cReadChar();}

if (buffer3 == 0x0d){
UART_CPutString("\r\n");
UART_CPutString("\r\n");
UART_CPutString(promptMessage1P);
UART_CPutString("\r\n");
UART_CPutString("\r\n");
goto Start;
}

if (buffer3 == 0x08 || buffer3 ==0x7f){
placeCounter--;
string3[placeCounter] = ' ';

}

string3[placeCounter] = buffer3;
UART_PutChar(string3[placeCounter]);
placeCounter++;

}else{UART_PutChar(0x07);}
}
break;

default:
UART_CPutString(promptMessage2P);

}
}
}

}

```

Testing:

After the program is written it should be built and uploaded to the PSoC. Turn the PSoC on and go to the Real Term console and look for the initial message. After that try writing and reading to all the strings as follows:

```
Would you like to read or write a string? R for read & W for write
Which string? Type a for 1, b for 2, c for 3, d for 4
>test string 1
Would you like to read or write a string? R for read & W for write
Do you want to print to Port or LCD? p for port & l for LCD
test string 1
Would you like to read or write a string? R for read & W for write
Which string? Type a for 1, b for 2, c for 3, d for 4
>test string 2
Would you like to read or write a string? R for read & W for write
Which string? Type a for 1, b for 2, c for 3, d for 4
Do you want to print to Port or LCD? p for port & l for LCD
█
```

Try this for all strings to make sure they are all functioning properly. Also check LCD when printing to the LCD display to make sure it has the correct output:



Below are functions used from the LCD & UART data sheet's:

LCD_Start

Description:

Initializes LCD to use the multi-line 4-bit interface. This function should be called before all other LCD functions.

C Prototype:

```
void LCD_Start(void);
```

Assembly:

```
lcall LCD_Start
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

LCD_PrString

Description:

Prints a null terminated RAM-based character string to the LCD at the present cursor location.

C Prototype:

```
void LCD_PrString(CHAR * sRamString);
```

Assembly:

```
mov    A,>sRamString    ; Load MSB part of pointer to RAM-based
null                                     ; terminated string.

mov    X,<sRamString     ; Load LSB part of pointer to RAM-based
null                                     ; terminated string.

lcall  LCD_PrString      ; Call function to display string at
current                  ; LCD cursor position.
```

Parameters:

sRamString: A pointer to a null-terminated string located in RAM.

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, the CUR_PP and IDX_PP page pointer registers are modified.

LCD_PrCString

Description:

Prints a null terminated ROM-based character string to the LCD at the present cursor location.

C Prototype:

```
void LCD_PrCString(const char * sRomString);
```

Assembly:

```
mov    A,>sRomString    ; Load MSB part of pointer to ROM-based
null                                     ; terminated string.
mov    X,<sRomString    ; Load LSB part of pointer to ROM-based
null                                     ; terminated string.
lcall  LCD_PrCString    ; Call function to display string at
current                                     ; LCD cursor position.
```

LCD_Position

Description:

Moves cursor to a location specified by the parameters. The upper left character is row 0, column 0. For a two-line by 16 character display, the lower right character is row 1, column 15.

C Prototype:

```
void LCD_Position(BYTE bRow, BYTE bCol);
```

Assembly:

```
mov    A,01h            ; Load Row
mov    X,02h            ; Load Column
lcall  LCD_Position
```

Parameters:

bRow: The row number at which to position the cursor. Zero specifies the first row.

bCol: The column number at which to position the cursor. Zero specifies the first (left most) column.

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

UART_Start

Description:

Sets the parity and enables the UART receiver and transmitter. When enabled, data can be received and transmitted.

C Prototype:

```
void UART_Start(BYTE bParitySetting)
```

Assembly:

```
mov    A, UART_PARITY_NONE
lcall  UART_Start
```

Parameters:

bParitySetting: One byte that specifies the transmit parity. Symbolic names are given in C and assembly, and their associated values are listed in the following table:

TX Parity	Value
UART_PARITY_NONE	0x00
UART_PARITY_EVEN	0x02
UART_PARITY_ODD	0x06

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

UART_CPutString

Description:

Sends constant (ROM), null terminated string out the UART TX port.

C Prototype:

```
void UART_CPutString(const char * szROMString)
```

Assembler:

```
mov    A,>szRomString    ; Load MSB part of pointer to ROM based null
                          ; terminated string.
mov    X,<szRomString     ; Load LSB part of pointer to ROM based null
                          ; terminated string.
lcall  UART_CPutString    ; Call function to send constant string out
                          ; UART TX
```

Parameters:

const char * szROMString: Pointer to string to be sent to the UART. MSB of string pointer is passed in the Accumulator and LSB of the pointer is passed in the X register.

Return Value:

None

Side Effects:

Program flow stays in this function, until the last character is loaded into the UART transmit buffer. The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

UART_PutChar**Description:**

Writes a single character to the UART TX port when the port buffer is empty.

C Prototype:

```
void UART_PutChar (CHAR cData)
```

Assembler:

```
mov  A,0x33          ; Load ASCII character "3" in A
lcall UART_PutChar    ; Call function to send single character to
                       ; UART TX port.
```

Parameters:

CHAR cData: The character to be sent to the UART TX port. Data is passed in the Accumulator.

Return Value:

None

Side Effects:

Program flow stays in this function until the data can be written to the UART TX buffer. The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

UART_cReadChar**Description:**

Reads UART RX port immediately if data is not available or an error condition exists, or zero is returned. Otherwise, the character is read and returned.

C Prototype:

```
CHAR UART_cReadChar (void)
```

Assembler:

```
lcall UART_cReadChar    ; Call function to read a character
cmp  A,0x00             ; Check for error
jz   ProcessError       ; If error, Process the error condition
mov  [CharBuffer],A     ; Store retrieved character in buffer
```

Parameters:

None

Return Value:

CHAR bData: Character read from UART RX port. ASCII characters from 1 to 255 are valid. A returned zero signifies an error condition or no data available.

Side Effects:

Function only accepts characters from 1 to 255 as valid. A 0x00 (null) character is detected as an error condition. The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx and CY8CLED16). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.