Summary

This app note demonstrates how to format serial data.

Description

In this example, data is coming in and going out of serial port 2. The incoming data is format the same way as the outgoing data. The format is as follows:

1 character representing a command
1 comma character
5 characters representing an double integer
1 comma character
5 characters representing an double integer
1 comma character
5 characters representing an double integer
1 carriage return

SEND.COMMAND takes a double integer and a character from the stack and sends out to the serial port the data packet.

RECEIVE.COMMAND **waits** for a data packet from the serial port and when it gets it, it puts it in a string. The SERIAL2.BUFFER string looks like this:

Char #	Contents
0	String count
1	Command character
2	Comma
3-7	First parameter (double number)
8	Comma
9-13	Second parameter (double number)
14	Comma
15-19	Third parameter (double number)
20	Carriage return

This example uses serial 1 as the primary serial port and serial 2 as the communications port. It switches from serial 1 to serial 2 during execution. This allows the user to program and debug using serial 1 and to connect serial 2 to the serial device. The final application program can use either serial 1 or serial 2 for communications. The user must change all serial 2 commands to serial 1 commands if serial 1 is to be used.

Error checking during receiving packets is not done, therefore improper packets or input data will not be detected. Error checking can be easily added and it is recommended to avoid program crashing. For example, the input string can be checked for commas in the right places. Also, the NUMBER word returns 1 of 3 flags when it converts a string to a number:

0	non-convertable character encountered in string
1	number converted to a 16 bit number
2	number converted to a 32 bit (double) number

If a 0 is returned, (see case statement), the data packet is invalid. The routine should be aborted and return an error flag.

```
DECIMAL
4 USE.PAGE
20 WIDTH !
ANEW SERIAL.COMMUNICATIONS

1  CONSTANT CNT.LEN             \ 1st byte of string is the string count
1  CONSTANT COMMAND.LEN         \ Length of command character
5  CONSTANT PARAMETER.LEN       \ Length of command parameter
20 CONSTANT PACKET.LEN          \ Length of packet
20 CONSTANT SERIAL2.BUFFER.LEN \ Length of serial2.buffer
6  CONSTANT TEMP.BUF.LEN        \ Temporary buffer for scratch work
3  CONSTANT X.OFFSET            \ X Parameter offset
9  CONSTANT Y.OFFSET            \ Y Parameter offset
15 CONSTANT Z.OFFSET            \ Z Parameter offset
10 CONSTANT LINE.FEED               \ ASCII number for line.feed
13 CONSTANT CARRIAGE.RETURN         \ ASCII number for carriage return
44 CONSTANT COMMA                   \ ASCII number for comma

4800 BAUD2

\ Set up serial2's buffer
VARIABLE SERIAL2.BUFFER          \ Defines variable (only reserves 2 bytes)
SERIAL2.BUFFER.LEN VALLOT        \ Reserves some more bytes in variable space

VARIABLE TEMP.BUFFER             \ Set up a temp buffer
SERIAL2.BUFFER.LEN VALLOT

\ SEND.COMMAND will take a command character and a command parameter (double
\ int) and send them out serial 2. Typing in the following will send the M with
\ x = 12345, y = 1000 and z = 5000
\
\      DIN 33333 DIN 22222 DIN 11111 ASCII M SEND.COMMAND
\
\ will send the command M to the position 5000, 1000, 12345.
\
\    M,11111, 22222, 33333<CR>
\
\ Parameter must be an integer less than 5 digits.  Parameter will be emitted
\ right justified, for example 1 will be sent out as ____1 where _ indicates a
\ space (ASCII 32)
: SEND.COMMAND ( d1\d2\d3\char -- | Z\Y\X\command char )
    4800 BAUD2 USE.SERIAL2        \ Set up serial 2
    EMIT2                         \ Emit command char - 1st item on stack
    COMMA EMIT2                   \ Emit comma to SERIAL2
    PARAMETER.LEN D.R             \ Emit 1st parameter (x), right justified
    COMMA EMIT2                   \ Emit comma to SERIAL2
    PARAMETER.LEN D.R             \ Emit 2nd parameter (y), right justified
    COMMA EMIT2                   \ Emit comma to SERIAL2
    PARAMETER.LEN D.R             \ Emit 3rd parameter (z), right justified
    CARRIAGE.RETURN EMIT2         \ Emit carriage return
    LINE.FEED EMIT2               \ Emit line feed
    USE.SERIAL1                   \ Return to serial 1 (only for debugging)
;
```

```
\ RECEIVE.COMMAND will receive a data packet and return a command character and
\ three parameters. There no error checking, therefore the inputs must be the
\ correct format or this will not work. The SERIAL2.BUFFER is used to store the
\ input string. To use, type the following:
\
\            RECEIVE.COMMAND EMIT D. D. D.
\
\ This will emit the command character and print the three double numbers
\
\            M 11111 22222 33333
\
: RECEIVE.COMMAND         ( -- char\d1\d2\d3 )
    LOCALS{ | d&output.parameter x&output.string }
    4800 BAUD2 USE.SERIAL2                      \ Set up serial port 2
    UEMIT X@ CFA.FOR DROP UEMIT X!              ( emit.xcfa -- ) \ kill the echo
    SERIAL2.BUFFER CNT.LEN XN+ PACKET.LEN EXPECT    \ Receives packet to buffer
    UEMIT X!                                    \ restore echo
\ Takes the Z parameter from string and leaves it on the stack as a double number
    PARAMETER.LEN 0 DO                          \ For each digit in double number do
        SERIAL2.BUFFER Z.OFFSET I + XN+ C@     \ Retrieve Ith character
        TEMP.BUFFER CNT.LEN I + XN+ C!         \ Store character in temp string
    LOOP
    BL TEMP.BUFFER TEMP.BUF.LEN XN+ C!          \ Store space bar at end of string
    PARAMETER.LEN TEMP.BUFFER C!                \ Store count at start of string
    TEMP.BUFFER NUMBER                          \ Convert string to number
    CASE
        0 OF        ENDOF                       \ Error condition - no conversion
        1 OF S>D    ENDOF                       \ Convert to double precision
    ENDCASE
\ Takes the Y parameter from string and leaves it on the stack as a double number
    PARAMETER.LEN 0 DO                          \ For each digit in double number do
        SERIAL2.BUFFER Y.OFFSET I + XN+ C@     \ Retrieve Ith character
        TEMP.BUFFER CNT.LEN I + XN+ C!         \ Store character in temp string
    LOOP
    BL TEMP.BUFFER TEMP.BUF.LEN XN+ C!          \ Store space bar at end of string
    PARAMETER.LEN TEMP.BUFFER C!                \ Store count at start of string
    TEMP.BUFFER NUMBER                          \ Convert string to number
    CASE
        0 OF        ENDOF                       \ Error condition - no conversion
        1 OF S>D    ENDOF                       \ Convert to double precision
    ENDCASE
\ Takes the X parameter from string and leaves it on the stack as a double number
    PARAMETER.LEN 0 DO                          \ For each digit in double number do
        SERIAL2.BUFFER X.OFFSET I + XN+ C@     \ Retrieve Ith character
        TEMP.BUFFER CNT.LEN I + XN+ C!         \ Store character in temp string
    LOOP
    BL TEMP.BUFFER TEMP.BUF.LEN XN+ C!          \ Store space bar at end of string
    PARAMETER.LEN TEMP.BUFFER C!                \ Store count at start of string
    TEMP.BUFFER NUMBER                          \ Convert string to number
    CASE
        0 OF        ENDOF                       \ Error condition - no conversion
        1 OF S>D    ENDOF                       \ Convert to double precision
    ENDCASE
\ Takes the command character from the input string & leaves it on the stack
    SERIAL2.BUFFER 1XN+ C@              \ Retrieve character and put it on the stack
    USE.SERIAL1                         \ Returns control to serial 1 (for debugging)
;
```


Mosaic Industries

5437 Central Ave Suite 1, Newark, CA 94560 Telephone: (510) 790-8222 Fax: (510) 790-0925