

# CORE FILE: SERIAL PORT COMMUNICATION BETWEEN MATLAB AND TI TMS320C54X\*

Douglas L. Jones  
Swaroop Appadwedula  
Matthew Berry  
Mark Haun  
Dima Moussa  
Daniel Sachs

This work is produced by OpenStax-CNXCNX and licensed under the  
Creative Commons Attribution License 1.0<sup>†</sup>

## Abstract

The core file provides two macros, READSER and WRITSER, for communicating with a PC via the serial port. One can create graphical user interfaces in MATLAB to control the DSP in real time.

## 1 Using the Serial Port

The core file supports the serial port installed on the TI TMS320C54x DSP. The serial port on the EVM is connected with a cable to COM2 on the PC. Before jumping to your code, the core file initializes the EVM's serial port to 38,400 bits per second (bps) with no parity, eight data bits, and one stop bit (but it may be necessary to restart the DSP completely if the serial port does not work.) It then accepts characters received from the PC by the **UART (Universal Asynchronous Receiver/Transmitter)** and buffers them in memory until your code retrieves them. It also can accept a block of bytes to transmit and send them to the UART in sequence.

Two macros are used to control the serial port: READSER and WRITSER. Both accept one parameter. READSER *n* reads up to *n* characters from the serial input buffer (the data coming from the PC) and places them in memory starting at \*AR3. (AR3 is left pointing one past the last memory location written.) The actual number of characters read is left in AR1. If AR1 is zero, then no characters were available in the input buffer.

WRITSER *n* adds *n* characters starting at \*AR3 to the serial output buffer; in other words, it queues them to be sent to the PC. AR3 is left pointing one location after the last memory location read.

---

\*Version 2.7: Feb 25, 2004 11:54 am +0000

<sup>†</sup><http://creativecommons.org/licenses/by/1.0>

Note that `READSER` and `WRITSER` modify registers `ARO`, `AR1`, `AR2`, `AR3`, and `BK`, as well as the flag `TC`. Be sure you restore these registers after calling `READSER` and `WRITSER` if you need them later in your code.

Note also that the core file allows up to 126 characters to be stored in the input and output buffers. Neither the DSP hardware nor the core file protect against serial-buffer overflows, so you must be careful not to allow the input and output buffers to overflow. (The length of the buffers can be changed by editing `ser_rxlen` and `ser_txlen` values in `core.asm`<sup>1</sup>.) The buffers are 127 characters long; however, the code cannot distinguish between a completely-full and completely-empty buffer. Therefore, only 126 characters can be stored in the buffers.

It is easy to check if the input or output buffers in memory are empty. The input buffer can be checked by comparing the values stored in the memory locations `srx_head` and `srx_tail`; if both memory locations hold the same value, the input buffer is empty. Likewise, the output buffer can be checked by comparing the values stored in memory locations `stx_head` and `stx_tail`. The number of characters in the buffer can be computed by subtracting the head pointer from the tail pointer; add the length of the buffer (normally 127) if the resulting distance is negative.

The following example shows the minimal amount of code necessary to echo received data back through the serial port. It is available as `ser_echo.asm`<sup>2</sup>.

```

1 .copy "core.asm"
2
3 .sect ".data"
4 hold .word 0
5
6 .sect ".text"
7 main
8 stm #hold,AR3      ; Read to hold location
9
10 READSER 1 ; Read one byte from serial port
11
12 cmpm AR1,#1 ; Did we get a character?
13 bc main,NTC ; if not, branch back to start
14
15 stm #hold,AR3 ; Write from hold location
16 WRITSER 1 ; ... one byte
17
18 b main

```

*Line 8* sets `AR3` to point to the location `hold` so that `READSER` will store serial data there. On *Line 9*, `READSER 1` reads one serial byte into `hold`; the byte is placed in the low-order bits of the word, and the high-order bits are zeroed. If a byte was read, `AR1` will be set to 1. `AR1` is checked in *Line 12*; *Line 13* branches back to the top if no byte was read. Otherwise, `AR3` is reset to `hold` (since `READSER` moved it), then on *Line 16*, `WRITSER` sends the word received. Finally, *Line 18* branches back to the start to receive another character.

<sup>1</sup><http://cnx.rice.edu/author/workgroups/90/m10017/core.asm>

<sup>2</sup>[http://cnx.org/content/m10821/latest/ser\\_echo.asm](http://cnx.org/content/m10821/latest/ser_echo.asm)

## 2 Using MATLAB to Control the DSP

MATLAB allows you to create a visual interface with standard **graphical user-interface (GUI)** controls such as sliders, checkboxes, and radio buttons to call MATLAB scripts. The following scripts can be used to create a sample interface:

- `ser_set.m`<sup>3</sup> : Initializes the serial port and user interface
- `wrt_slid.m`<sup>4</sup> : Called when sliders are moved to send new data

### 2.1 Creating a MATLAB user interface

The following code (`ser_set.m`<sup>5</sup>) initializes the serial port COM2, then creates a minimal user interface consisting of three sliders.

```

1 % ser_set: Initialize serial port and create three sliders
2
3 % Set serial port mode
4 !mode com2:38400,n,8,1
5
6 % open a blank figure for the slider
7 Fig = figure(1);
8
9 % open sliders
10
11 % first slider
12 sld1 = uicontrol(Fig,'units','normal','pos',[.2,.7,.5,.05],...
13 'style','slider','value',4,'max',254,'min',0,'callback','wrt_slid');
14
15 % second slider
16 sld2 = uicontrol(Fig,'units','normal','pos',[.2,.5,.5,.05],...
17 'style','slider','value',4,'max',254,'min',0,'callback','wrt_slid');
18
19 % third slider
20 sld3 = uicontrol(Fig,'units','normal','pos',[.2,.3,.5,.05],...
21 'style','slider','value',4,'max',254,'min',0,'callback','wrt_slid');
```

*Line 4* of this code uses the Windows `mode` command to set up serial port COM2 (which is connected to the DSP) to match the serial port settings on the DSP evaluation board: 38,400 bps, no parity, eight data bits, and one stop bit. *Line 7* then creates a new MATLAB figure for the controls; this prevents the controls from being overlaid on any graph you may have already created.

*Lines 12 through the end* create the three sliders for the user interface. Several parameters are used to specify the behavior of each slider. The first parameter, `Fig`, tells the slider to create itself in the window we created in *Line 7*. The rest of the parameters are property/value pairs:

- units** - `normal` tells MATLAB to use positioning relative to the window boundaries.
- pos** - Tells MATLAB where to place the control.

<sup>3</sup>[http://cnx.rice.edu/author/workgroups/90/m10821/ser\\_set.m](http://cnx.rice.edu/author/workgroups/90/m10821/ser_set.m)

<sup>4</sup>[http://cnx.rice.edu/author/workgroups/90/m10821/wrt\\_slid.m](http://cnx.rice.edu/author/workgroups/90/m10821/wrt_slid.m)

<sup>5</sup>[http://cnx.rice.edu/author/workgroups/90/m10821/ser\\_set.m](http://cnx.rice.edu/author/workgroups/90/m10821/ser_set.m)

- style** - Tells MATLAB what type of control to place. `slider` creates a slider control.
- value** - Tells MATLAB the default value for the control.
- max** - Tells MATLAB the maximum value for the control.
- min** - Tells MATLAB the minimum value for the control.
- callback** - Tells MATLAB what script to call when the control is manipulated. `wrt_slid.m`<sup>6</sup> is a MATLAB file that reads the values of the sliders and sends them to the DSP via the serial port.

### 2.1.1 User-interface callback function

Every time a slider is moved, the file `wrt_slid.m`<sup>7</sup> is called:

```

1 % wrt_slid: write values of sliders out to com port
2
3 % open com port for data transfer
4 fid = fopen('com2:', 'w');
5
6 % send data from each slider
7 v = round(get(sld1, 'value'));
8 fwrite(fid, v, 'uint8');
9
10 v = round(get(sld2, 'value'));
11 fwrite(fid, v, 'uint8');
12
13 v = round(get(sld3, 'value'));
14 fwrite(fid, v, 'uint8');
15
16 % send reset pulse
17 fwrite(fid, 255, 'uint8');
18
19 % close com port connection
20 fclose(fid);

```

*Line 4* of `wrt_slid.m`<sup>8</sup> opens COM2 for writing. (It has already been initialized by `ser_set.m`<sup>9</sup>.) Then *Line 7* reads the value of the first slider using MATLAB's `get` function to retrieve the `value` property. The value is then rounded off to create an integer, and the integer is sent as an 8-bit quantity to the DSP in *Line 8*. (The number that is sent at this step will appear when the serial port is read with `READSER` in your code.) Then the other two sliders are sent in the same way.

*Line 17* sends `0xFF` (255) to the DSP, which can be used to indicate that the three previously-transmitted values represent a complete set of data points. Your code can check for the value 255 to detect and correct synchronization errors.

*Line 20* closes the serial port. Note that MATLAB buffers the data being transmitted, and data is often not sent until the serial port is closed. Make sure you close the port after writing a data block to the serial port.

<sup>6</sup>[http://cnx.rice.edu/author/workgroups/90/m10821/wrt\\_slid.m](http://cnx.rice.edu/author/workgroups/90/m10821/wrt_slid.m)

<sup>7</sup>[http://cnx.rice.edu/author/workgroups/90/m10821/wrt\\_slid.m](http://cnx.rice.edu/author/workgroups/90/m10821/wrt_slid.m)

<sup>8</sup>[http://cnx.rice.edu/author/workgroups/90/m10821/wrt\\_slid.m](http://cnx.rice.edu/author/workgroups/90/m10821/wrt_slid.m)

<sup>9</sup>[http://cnx.rice.edu/author/workgroups/90/m10821/ser\\_set.m](http://cnx.rice.edu/author/workgroups/90/m10821/ser_set.m)