PERSISTOR
Instruments Inc.

254-J Shore Road, Bourne, MA 02532, USA
Tel: 508-759-6434 Fax: 508-759-6436
www.persistor.com info@persistor.com

Data Acquisition and Storage Solutions for Industry and Science

CF2 TPU UART FUNCTIONS

Overview

PRELIMINARY

Table of Contents

OVERVIEW

This is an overview of the TPU UART functions found in PicoDOS for the Persistor Instruments CF2.

The TPU has the ability to perform the function of an asynchronous serial transmitter or receiver. It is possible to use a single TPU pin as an asynchronous receiver of transmitter. You can also pair two TPU pins and use them together as a serial port.

Functions are provided to access the capabilities of the TPU UART. Below is a short list with a description of the functions provided. A more detailed description is provided further along in the document.

INITIALIZATION	
TUInit	Initialize the TPU UART module
TURelease	Close all ports then release all memory and resources allocated to TPU UARTs
TUOpen	Open a TPU UART port for serial communications
TUConfigure	Set the baud rate and parity
TUClose	Close the specified port and release its memory
TUSetDefaultParams	Setup new default TPU UART open parameters
TUGetDefaultParams	Return the default TPU UART open parameters
TUNotifyPreClockChange	Prepare the TPU UART for a pending clock change
TUNotifyPostClockChange	Update the TPU UART baud rate from a clock change
TUBlockDuration	Return expected block duration in ms at current baud
RECEIVE FUNCTIONS	
TURxGetByte	Wait for, and return the next word
TURxGetByteWithTimeout	Return next word or -1 on timeout
TURxGetBlock	Receive a block of bytes with timeout
TURxPeekByte	Fetch Nth byte in receive queue without deleting
TURxQueuedCount	Return the number of words in the receive queue
TURxFlush	Delete any data in the receive queue
TRANSMIT FUNCTIONS	
TUTxPutByte	Send byte
TUTxPrintf	Transmit using standard printf conventions

TUTxPutBlock	Transmit a block of bytes with timeout
TUTxWaitCompletion	Wait for all transmissions to complete
TUTxQueuedCount	Return the number of words in the transmit queue
TUTxFlush	Delete any data in the transmit queue

Warning

A couple of words of warning about using TPU UARTs.

Flow Control

There is no hardware flow control for a TPU UART. You may implement a software scheme by sending XON and XOFF characters if you are ambitious. But, be warned, if the equipment that you are connected to is limited in its ability to sustain the baud rate you are using then you will surely suffer a loss of data.

Loading

The Motorola document Asynchronous Serial Interface TPU Function (UART_AsyncSerial_tpup07r1.pdf) can be found here in C:\Program Files\Persistor\MotoCross Support\CFX\Docs\pdf\TPU. This document contains all of the information about the TPU UARTs. A question that is often asked is something like, "How many channels can I transmit on/Receive on/both transmit AND receive at the same time". The best information is on page 9 of the above-mentioned document. Here is the excerpt:

> **All examples assume that only the UART function is running. Examples are for absolute worst case, e.g. all receivers receive a stop bit at the same time, since this is the longest state.**
>
> **When only transmitters are running, maximum baud rate for all channels combined is 360 kbaud. This can be one transmitter with 360 kbaud, or nine with 38.4 kbaud, or any other combination.**
>
> **When only receivers are running, maximum baud rate is 233 kbaud. This can be one receiver with 233 kbaud, or six with 38.4 kbaud, or any other combination.**
>
> **When both receivers and transmitters are running, maximum baud rate is 142 kbaud. This can be one pair running at 142 kbaud, or three pair at 38.4 kbaud, or seven pair at 19.2 kbaud, or any other combination.**

Again, for the sake of clarity, "All examples assume that only the UART function is running". This is almost never the case. Your CF2 program will probably be doing a lot of other tasks besides servicing the TPU UARTs. This means that the maximum number of channels at some mix of baud rates that YOU will be able to achieve will depend on what you are doing. TEST, TEST, TEST!

Structures

The TUChParams structure is defined in cfxpico.h. This structure holds parameters for initializing a TPU UART port.

```
typedef struct
    {
    short  bits;      // data bits exclusive of start, stop, parity
    short  parity;    // parity: 'o','O','e','E', all else is none
    short  autobaud;  // automatically adjust baud when clock changes
    long   baud;      // baud rate
    short  rxpri;     // receive channel TPUPriority
    short  txpri;     // transmit channel TPUPriority
    short  rxqsz;     // receive channel queue buffer size
    short  txqsz;     // transmit channel queue buffer size
    short  tpfbsz;    // transmit channel printf buffer size
    } TUChParams
```

Functions

TUInit

Initialize the TPU UART module. This must be called before any other TPU UART functions are called. It provides the TPU UART functions with access to the preferred memory management functions.

voidTUInit(Callocf *callocf, Freef *freef);

Arguments: Pointers to memory allocation/free functions

Returns: Nothing

Example:

```
TUInit(calloc, free);
```

TURelease

This function closes all open TPU UART ports and then releases all memory and resources that were allocated to TPU UARTs. This is done automatically when your program quits. If you need to use it during program execution remember that in order to re-use any previously configured TPU UARTs you will need to dynamically reconfigure those ports.

voidTURelease(void);

Arguments: None

Returns: Nothing

TUOpen

Opens a TPU UART port for serial communications. Specify separate valid TPU channels (1 to 15) for receive (rxch) and transmit (txch). If you use -1 as an argument for rxch then the port will be a transmit only port. Likewise, a value of -1 for txch will make the port receive only.

TUPort *TUOpen(short rxch, short txch, long baud, TUChParams *tp);

Arguments:

rxch – TPU channel 1-15

txch – TPU channel 1-15

baud – baud rate

*tp – pointer to TUChParams structure (pass a NULL to use default parameters).

Returns: A pointer to the port structure. You will use this pointer in subsequent function calls.

Example: Open a TPU UART port receiving on TPU 10 and transmitting on TPU 11. The macro TPUChanFromPin will convert a CF2 pin number to its corresponding TPU number. We are passing 0 for the parameters so that we use the defaults but we can specify a different baud rate without having to use the TUChParams structure.

```
TUPort *tuport;

TUInit(calloc, free);

tuport = TUOpen(TPUChanFromPin(31), TPUChanFromPin(32), 19200L, 0);
```

TUConfigure

Set the baud rate and parity.

voidTUConfigure(TUPort *tup, TUChParams *tp)

Arguments:

tup – Pointer returned from TUOpen.

*tp – pointer to TUChParams structure.

Returns: Nothing.

Example:

```
TUChParams MyParams;

MyParams.bits = TUGetDefaultParams()->bits;

MyParams.parity = TUGetDefaultParams()->parity);

MyParams.autobaud = TUGetDefaultParams()->autobaud);

MyParams.baud = 19200L;

MyParams.rxpri = TUGetDefaultParams()->rxpri);

MyParams.txpri = TUGetDefaultParams()->txpri);

MyParams.rxqsz = TUGetDefaultParams()->rxqsz);

MyParams.txqsz = TUGetDefaultParams()->txqsz);
```

```
MyParams.tpfbsz = TUGetDefaultParams()->tpfbsz);

TUConfigure(tuport,& MyParams);
```

TUClose

Close the specified port and release its memory

short TUClose(TUPort *tup)

Arguments:

tup – Pointer returned from TUOpen.

Returns: Zero if successful and non-zero if an error occurred.

Example:

```
if(!TUClose(tuport))
    cprintf("Port Closed!\n");
else
    cprintf("Error closing port!\n");
```

TUSetDefaultParams

Setup new default TPU UART open parameters. The new settings will only apply to ports opened after the set call.

voidTUSetDefaultParams(TUChParams *rp)

Arguments:

*tp – pointer to TUChParams structure.

Returns: Nothing

Example: All subsequent calls to TUOpen will use 19200 as the default baud rate.

```
TUChParams MyParams;

MyParams.bits = TUGetDefaultParams()->bits;

MyParams.parity = TUGetDefaultParams()->parity);

MyParams.autobaud = TUGetDefaultParams()->autobaud);

MyParams.baud = 19200L;

MyParams.rxpri = TUGetDefaultParams()->rxpri);

MyParams.txpri = TUGetDefaultParams()->txpri);

MyParams.rxqsz = TUGetDefaultParams()->rxqsz);
```

```
MyParams.txqsz = TUGetDefaultParams()->txqsz);

MyParams.tpfbsz = TUGetDefaultParams()->tpfbsz);

TUSetDefaultParams(&MyParams);
```

TUGetDefaultParams

Return pointer to the default TPU UART open parameters.

TUChParams*TUGetDefaultParams(void)

Arguments: None

Returns: Pointer to default TUChParams structure.

Example: Copy default values to our own TUChParams structure.

```
TUChParams MyParams;

memcpy(&MyParams, TUGetDefaultParams(), sizeof(TUChParams));
```

TUNotifyPreClockChange

Prepare the TPU UART for a pending clock change. You must do this before changing the system clock rate or the TPU baud rates will change. It must be followed with a call to TUNotifyPostClockChange.

voidTUNotifyPreClockChange(TUPort *tup)

Arguments:

tup – Pointer returned from TUOpen.

Returns: Nothing

Example:

```
TUNotifyPreClockChange(tuport);
```

TUNotifyPostClockChange

Update the TPU UART baud rate following a clock change.

voidTUNotifyPostClockChange(TUPort *tup)

Arguments:

tup – Pointer returned from TUOpen.

Returns: Nothing

Example:

```
TUNotifyPreClockChange(tuport);

TMGSetSpeed(8000L);
```

```
TUNotifyPostClockChange(tuport);
```

TUBlockDuration

Returns the expected block duration in milliseconds at current baud rate.

longTUBlockDuration(TUPort *tup, long bytes)

Arguments:

tup – Pointer returned from TUOpen.

bytes – The number of bytes in the block to be sent.

Returns: A long integer representing the estimated transmission time for the block in milliseconds.

Example:

```
cprintf("The time to send 32 bytes is: %ld ms.\n",TUBlockDuration(tuport,32));
```

TURxGetByte

Wait for, and return the next word.

short TURxGetByte(TUPort *tup, bool block)

Arguments:

tup – Pointer returned from TUOpen.

block – true if you wish the function to wait until a byte is received or false to return if no byte is available.

Returns: A short containing the byte received in the lower 8 bits.

Example:

```
cprintf("The value received: %04x\n",TURxGetByte(tuport, true));
```

TURxGetByteWithTimeout

Return next word or -1 on timeout.

short TURxGetByteWithTimeout(TUPort *tup, short millisecs);

Arguments:

tup – Pointer returned from TUOpen.

millisecs – The number of milliseconds to wait before returning without a value from the port.

Returns: A short containing the byte received in the lower 8 bits or -1 if a timeout occurred.

Example: Wait 10 milliseconds for a value and then timeout (return -1).

```
cprintf("The value received: %04x\n", TURxGetByteWithTimeout (tuport, 10));
```

TURxGetBlock

Receive a block of bytes with timeout.

longTURxGetBlock(TUPort *tup, uchar *buffer, long bytes, short millisecs)

Arguments:

tup – Pointer returned from TUOpen.

buffer – A pointer to the buffer where the bytes will be written.

bytes – The number of bytes in the block to be received.

millisecs – The number of milliseconds to wait before returning without receiving 'bytes' from the port.

Returns: A long indicating the number of bytes received.

Example:

```
uchar mybuffer[128];

//Wait 500 milliseconds to receive 128 bytes

TURxGetBlock(tuport, mybuffer, 128, 500));
```

TURxPeekByte

Fetch Nth byte in receive queue without deleting.

short TURxPeekByte(TUPort *tup, short index);

Arguments:

tup – Pointer returned from TUOpen.

index – Offset into the receive queue (zero for the first byte, 1 for the second, etc).

Returns: The byte at the offset (index) specified or -1 for an error (e.g. if the nth byte doesn't exist).

Example:

```
cprintf("The value at offset 5 is: %04x\n", TURxPeekByte(tuport, 5));
```

TURxQueuedCount

Return the number of words in the receive queue.

short TURxQueuedCount(TUPort *tup);

Arguments:

tup – Pointer returned from TUOpen.

Returns: The number of words in the receive queue.

Example:

```
cprintf("The queue has: %d bytes waiting.\n", TURxQueuedCount(tuport));
```

TURxFlush

Delete any data in the receive queue.

voidTURxFlush(TUPort *tup)

Arguments:

tup – Pointer returned from TUOpen.

Returns: Nothing

Example:

```
TURxFlush(tuport);
```

TUTxPutByte

Send byte.

short TUTxPutByte(TUPort *tup, ushort data, bool block)

Arguments:

tup – Pointer returned from TUOpen.

data – The word to send (byte in the bottom 8 bits).

block – true if you wish the function to wait until a byte is sent or false just queue the byte (not wait for it to be sent).

Returns: The word sent.

Example: Send an ASCII ‘1’ to the port.

```
TUTxPutByte(tuport, 0x0031, false);
```

TUTxPrintf

Transmit using standard printf conventions.

short TUTxPrintf(TUPort *tup, char * str, ...)

Arguments:

tup – Pointer returned from TUOpen + Standard printf arguments.

Returns: The number of characters printed (sent) or -1 if an error occurred.

Example:

```
short i = 3;

TUTxPrintf(tuport, “pi is greater than %d”,i);
```

TUTxPutBlock

Transmit a block of bytes with timeout.

longTUTxPutBlock(TUPort *tup, uchar *buffer, long bytes, short millisecs)

Arguments:

tup – Pointer returned from TUOpen.

buffer – A pointer to the buffer of bytes to send.

bytes – The number of bytes in the block to be sent.

millisecs – The number of milliseconds to wait before returning without sending ‘bytes’to the port.

Returns: A long indicating the number of bytes sent.

Example:

```
uchar mybuffer[128];

//Wait 500 milliseconds to send 128 bytes

TUTxPutBlock(tuport, mybuffer, 128, 500));
```

TUTxWaitCompletion

Wait for all transmission to complete.

voidTUTxWaitCompletion(TUPort *tup)

Arguments:

tup – Pointer returned from TUOpen.

Returns: Nothing

Example: Wait for all data to be sent before returning.

```
TUTxWaitCompletion(tuport);
```

TUTxQueuedCount

Return the number of words waiting to be sent in the transmit queue.

short TUTxQueuedCount(TUPort *tup);

Arguments:

tup – Pointer returned from TUOpen.

Returns: The number of words in the transmit queue.

Example:

```
cprintf("The queue has: %d bytes waiting.\n", TUTxQueuedCount(tuport));
```

TUTxFlush

Delete any data in the transmit queue.

voidTUTxFlush(TUPort *tup)

Arguments:

tup – Pointer returned from TUOpen.

Returns: Nothing

Example:

```
TUTxFlush(tuport);
```

Examples

The source code for the following examples can be downloaded from the Persistor Instruments web site. Go to http://www.persistor.com and click on **programming examples**. You should see them listed under examples for the CF2.

GPS/NMEA TPU UART Example

This is an example of using the TPU UART on the CF2 to receive NMEA data from a GPS. The Persistor Instruments R216AU Recipe Card has two 1/8 inch stereo jacks and a Maxim RS232 driver so that two serial ports can be added. This example uses one of those ports (the one furthest from the front edge using TPU 10 (pin 31) for receiving and TPU 11 (pin 32) for transmitting.

We parse all received NMEA strings and verify the checksum. Valid strings are tested looking for $GPGGA strings which contain latitude and longitude information. All valid $GPGGA strings have the degrees and minutes.seconds separated and printed in formatted strings. It would be easy to modify this program to extract data from other NMEA strings.

Send a break (F8 under Motocross) to exit.

This application prints the position ONLY when we are sure we have a valid 2D OR 3D position fix. We know what type of a fix we have by looking at the $GPGSA string. The second field is: 1=No fix, 2=2D, 3=3D. The program only prints the position when it has a valid 2D or better fix. It is easy to change the program so it only prints position when it has a 3D fix.

```
/*****************************************************************************\
**   GPSReceive.c      TPU NMEA at 4800
**
**   Description: This is an example of using the TPU UART on the CF2 to receive
**   NMEA data from a GPS.  We parse all received NMEA strings and verify the
**   checksum.  Valid strings are tested looking for $GPGGA strings which contain
**   latitude and longitude information. All valid $GPGGA strings have the degrees
**   and minutes separated and printed in formatted strings. It would be easy to
**   modify this program to extract data from other NMEA strings.
**
**   Send a break (F8 under Motocross) to exit.
**
**   Release:      2002/08/09
**   Updated:      2002/11/16
**
**   This application was updated to print the position ONLY when
**   we are sure we have a valid 2D OR 3D position fix. We know what type
**   of a fix we have by looking at $GPGSA string.  The second field
**   is: 1=No fix, 2=2D, 3=3D.
**
******************************************************************************
**
**   COPYRIGHT (C) 2002 PERSISTOR INSTRUMENTS INC., ALL RIGHTS RESERVED
**
**   Developed by: Thomas P. Sullivan for Persistor Instruments Inc.
**   254-J Shore Road, Bourne, MA 02532  USA
**   tpsully@persistor.com - http://www.persistor.com
**
**   Copyright (C) 2002 Persistor Instruments Inc.
**   All rights reserved.
**
******************************************************************************
**
**   Copyright and License Information
**
**   Persistor Instruments Inc. (hereafter, PII) grants you (hereafter,
**   Licensee) a non-exclusive, non-transferable license to use the software
**   source code contained in this single source file with hardware products
**   sold by PII. Licensee may distribute binary derivative works using this
**   software and running on PII hardware products to third parties without
```

```
**   fee or other restrictions.
**
**   PII MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF THE
**   SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
**   IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE,
**   OR NON-INFRINGEMENT. PII SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY
**   LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE OR
**   ITS DERIVATIVES.
**
**   By using or copying this Software, Licensee agrees to abide by the
**   copyright law and all other applicable laws of the U.S. including, but
**   not limited to, export control laws, and the terms of this license. PII
**   shall have the right to terminate this license immediately by written
**   notice upon Licensee's breach of, or non-compliance with, any of its
**   terms. Licensee may be held legally responsible for any copyright
**   infringement or damages resulting from Licensee's failure to abide by
**   the terms of this license.
**
\*****************************************************************************/

#include <cfxbios.h>       // Persistor BIOS and I/O Definitions
#include <cfxpico.h>       // Persistor PicoDOS Definitions

#include <assert.h>
#include <ctype.h>
#include <errno.h>
#include <float.h>
#include <limits.h>
#include <locale.h>
#include <math.h>
#include <setjmp.h>
#include <signal.h>
#include <stdarg.h>
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include <dirent.h>        // PicoDOS POSIX-like Directory Access Defines
#include <dosdrive.h>  // PicoDOS DOS Drive and Directory Definitions
#include <fcntl.h>         // PicoDOS POSIX-like File Access Definitions
#include <stat.h>      // PicoDOS POSIX-like File Status Definitions
#include <termios.h>       // PicoDOS POSIX-like Terminal I/O Definitions
#include <unistd.h>        // PicoDOS POSIX-like UNIX Function Definitions

typedef struct  {
    char Type[32];
    char Field[32][20];    //20 elements at 32 characters per field
} NMEAMessage;

#define NMEA_MAX_SIZE  132

bool StripFields(NMEAMessage *messageNMEA, char *strNMEA);
bool VerifyChkSum(char *strNMEA);

//Uncomment if you want all received NMEA strings to be printed
//#define PRINT_ALL_RECEIVED

//Comment the line below in order to receive NMEA ONLY!!!
//(no communications to the GPS)
#define RECEIVE_ONLY

/*****************************************************************************\
**   main
\*****************************************************************************/
int main(int argc, char **argv)
    {
    short    i;
    short    result = 0;       // no errors so far
    TUPort   *tuport;
    ushort   index = 0;
    short    numchars;
    TUChParams    NMEAParams;
    uchar    c;
    NMEAMessage   strCurrent;
    uchar    NMEAstring[NMEA_MAX_SIZE];
    uchar    strLatDeg[32];
    uchar    strLatMin[32];
    uchar    strLonDeg[32];
    uchar    strLonMin[32];
```

```
    bool ValidPosition = false;
    // Identify the progam and build
    printf("\nProgram: %s: %s %s \n", __FILE__, __DATE__, __TIME__);
    // Identify the device and its firmware
    printf("Persistor CF%d SN:%ld   BIOS:%d.%d   PicoDOS:%d.%d\n", CFX,
        BIOSGVT.CFxSerNum, BIOSGVT.BIOSVersion, BIOSGVT.BIOSRelease,
        BIOSGVT.PICOVersion, BIOSGVT.PICORelease);
    // Identify the arguments
    printf("\n%d Arguments:\n", argc);
    for (i = 0; i < argc; i++)
        printf("  argv[%d] = \"%s\"\n", i, argv[i]);

//  *************************
//  Enable the RS-232 receiver
//  *************************
//
//NOTE:  In order to use the R2AU/R212AU/R216AU RecipeCard for Auxiliary UARTs using the
//       TPU, you will need to do the following (follow along on the schematic for the RecipeCard):
//
//       1)  Make sure that the dip switch is closed for the TPU lines you are using. If,
//       for example, you are going to use AUX1 you must close 1 & 2 on the dip
//       switch in order to connect pins 34 (Tx) and 33 (Rx) to the Maxim driver chip.
//       If you are using AUX2 then you must close 3 & 4 on the dip switch.
//
//       2)  If you are transmitting only (using Tx only) then you must close dip switch 6 and drive
//       pin 29 (TPU8) high in software. To disable the driver, force pin 29 low. If you are not using the
//       transmit capability of the AUX UART (receive only) then leave dip switch 6 open and an internal
//       pulldown on /OFF will keep the driver disabled.
//
//       3)  In order to be able to receive serial data you must close dip switch 5 and drive
//       pin 30 low to enable the receive driver(s). Driving pin 30 high will disable the driver
//       as will leaving dip switch 5 open because there is a pullup on /EN on the RecipeCard.
//
    PIOWrite(30,0);
    PIOWrite(29,1);

    TUInit(calloc, free);      // give TU manager access to our heap

//  Gather the default parameters
    memcpy(&NMEAParams,TUGetDefaultParams(),sizeof(TUChParams));

//  Set our baud
    NMEAParams.baud = 4800;    //NMEA at 4800 bps from my Garmin eTrex Venture

//  Uncomment (below) to print parameters
/*
    cprintf("NMEAParams.bits = %d\n",NMEAParams.bits);
    cprintf("NMEAParams.parity = %c\n",NMEAParams.parity);
    cprintf("NMEAParams.autobaud = %d\n",NMEAParams.autobaud);
    cprintf("NMEAParams.baud = %ld\n",NMEAParams.baud);
    cprintf("NMEAParams.rxpri = %d\n",NMEAParams.rxpri);
    cprintf("NMEAParams.txpri = %d\n",NMEAParams.txpri);
    cprintf("NMEAParams.rxqsz = %d\n",NMEAParams.rxqsz);
    cprintf("NMEAParams.txqsz = %d\n",NMEAParams.txqsz);
    cprintf("NMEAParams.tpfbsz = %d\n",NMEAParams.tpfbsz);
*/

//  Open the TPU UART (TPU 10 is pin 31 and TPU 11 is pin 32)
#ifdef RECEIVE_ONLY
    tuport = TUOpen(TPUChanFromPin(31), -1, 4800L, &NMEAParams);
#else
    tuport = TUOpen(TPUChanFromPin(31), TPUChanFromPin(32), 4800L, &NMEAParams);
#endif

    if(tuport == 0)
    {
        cprintf("\nCannot open TPU Channel %d for receive\n\n",TPUChanFromPin(25));
    }
    else
    {
        cprintf("\nOpened TPU Channel %d for receive\n",TPUChanFromPin(33));

//      Do this until a break is sent on the SCI (main UART)
        while(!SCIRxBreak(100))
        {
            numchars = TURxQueuedCount(tuport); //See if we have anything from the TPU UART
            if(numchars > 0)
            {
                c = TURxGetByte(tuport,false);
                if((c!=0x0A)&&(index<NMEA_MAX_SIZE-1))
                {
#ifdef PRINT_ALL_RECEIVED
```

```
                        cprintf("%c",c);
#endif
                        NMEAstring[index++] = (c!=0x0d)?(c):(0); //Add the next character (or NULL if 0x0D)
                    }
                    else
                    {
                        NMEAstring[index] = 0; //Add a NULL
                        index = 0;

                        //If we got a whole line (we got an 0x0A or max chars)
                        if(VerifyChkSum(NMEAstring))
                        {
                            //Clear before we use (just a precaution)
                            memset(&strCurrent,0,sizeof(NMEAMessage));
                            //Process string
                            StripFields(&strCurrent, NMEAstring);
                            //Display individual fields
                            if(strncmp("$GPGGA\0",strCurrent.Type,strlen("$GPGGA\0"))==0)
                            {
                                if(ValidPosition)
                                {
                                    //Split the field so degrees and minutes are seperate
                                    strncpy(strLatDeg,&strCurrent.Field[1][0],2);
                                    strLatDeg[2] = 0;
                                    strcpy(strLatMin,&strCurrent.Field[1][2]);

                                    strncpy(strLonDeg,&strCurrent.Field[3][0],3);
                                    strLonDeg[3] = 0;
                                    strcpy(strLonMin,&strCurrent.Field[3][3]);

                                    cprintf("\n Latitude:  %s %s
%s\n",strLatDeg,strLatMin,&strCurrent.Field[2][0]);
                                    cprintf("Longitude: %s %s
%s\n",strLonDeg,strLonMin,&strCurrent.Field[4][0]);

                                    //Force looking for valid position fix
                                    ValidPosition = false;

                                    cprintf("\n");
                                }
                                else
                                {
                                    cprintf("\nNo 3D Fix\n");
                                }
                            }
                            else if(strncmp("$GPGSA\0",strCurrent.Type,strlen("$GPGSA\0"))==0)
                            {
                                switch(strCurrent.Field[1][0])
                                {
                                    default:
                                    case '1':
                                        ValidPosition = false;
                                    break;
                                    case '2':
                                    case '3':
                                        ValidPosition = true;
                                    break;
                                }
                            }
                        }
                    }
                }
            }
        }
        TUClose(tuport);
    }

// Various exit strategies
//   BIOSReset();           // full hardware reset
//   BIOSResetToPBM();      // full reset, but stay in Pesistor Boot Monitor
//   BIOSResetToPicoDOS();  // full reset, but jump to 0xE10000 (PicoDOS)
    return result;

    }   //____ main() ____//


/*****************************************************************************
**   StripFields
**
**   Pass a NMEA string and individual NMEA message fields
**   are stripped and copied to the NMEAMessage structure. By
**   examining the Type field you can extract the various fields
**   of information from the NMEA string and manipulate them.
```

```
**
*******************************************************************************/
bool StripFields(NMEAMessage *messageNMEA, char *strNMEA)
{
     char *str = strNMEA;
     char *ptrtok;
     short i;

     ptrtok = strtok(str,",\n\0");
     strncpy(&messageNMEA->Type, ptrtok,31);

     for(i=0;i<20;i++)
     {
          ptrtok = strtok(NULL,",\n\0");
          if(ptrtok)
               strncpy(&messageNMEA->Field[i][0], ptrtok, 31);
          else
               messageNMEA->Field[i][0] = 0;
     }

     return true;
}


/*******************************************************************************
**   VerifyChkSum
**
**   Compute the checksum of the string passed and compare against
**   the received checksum. Return true if they match otherwise
**   return false.
**
**   A checksum of a NMEA string is an XOR of all the bytes compared against
**   the byte formed by the two characters after the asterisk ('*') at the end of
**   the sentence.
**
*******************************************************************************/
bool VerifyChkSum(char *strNMEA)
{
     uchar *str = strNMEA;
     uchar test[10];
     uchar chksum;
     bool result = false;

//   Get the first character
     chksum = *strNMEA;

     while(((*str!= '*')&&((str-strNMEA)<131)))
     {
          chksum = chksum ^ *str++;
     }

     sprintf(test,"%02X\0",chksum);

     if(strncmp(test,&str[1],2)==0)
          result = true;
     else
          cprintf("\nChecksum Error!!!\n\n");

     return result;
}
```

Simple TPU UART Example

This is a very simple CF2 TPU UART example. We perform a 16 bit A/D conversion on 8 channels and write the results as hex to both the main UART and the TPU UART. If a character is sent from the TPU UART end it is displayed on the main UART. This example assumes that you are using a R216AU Recipe Card with 16 bit A/D converter and RS-232 driver for two TPU UARTS.

```
/*****************************************************************************\
**   TPU_UART.c        TPU UART example with A/D
**
**   Release:      2002/06/20
******************************************************************************
**
**   COPYRIGHT (C) 2002 PERSISTOR INSTRUMENTS INC., ALL RIGHTS RESERVED
**
**   Developed by: Thomas P. Sullivan for Persistor Instruments Inc.
**   254-J Shore Road, Bourne, MA 02532  USA
**   tpsully@persistor.com - http://www.persistor.com
**
**   Copyright (C) 2002 Persistor Instruments Inc.
**   All rights reserved.
**
******************************************************************************
**
**   Copyright and License Information
**
**   Persistor Instruments Inc. (hereafter, PII) grants you (hereafter,
**   Licensee) a non-exclusive, non-transferable license to use the software
**   source code contained in this single source file with hardware products
**   sold by PII. Licensee may distribute binary derivative works using this
**   software and running on PII hardware products to third parties without
**   fee or other restrictions.
**
**   PII MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF THE
**   SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
**   IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE,
**   OR NON-INFRINGEMENT. PII SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY
**   LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE OR
**   ITS DERIVATIVES.
**
**   By using or copying this Software, Licensee agrees to abide by the
**   copyright law and all other applicable laws of the U.S. including, but
**   not limited to, export control laws, and the terms of this license. PII
**   shall have the right to terminate this license immediately by written
**   notice upon Licensee's breach of, or non-compliance with, any of its
**   terms. Licensee may be held legally responsible for any copyright
**   infringement or damages resulting from Licensee's failure to abide by
**   the terms of this license.
**
\*****************************************************************************/

#include <cfxbios.h>       // Persistor BIOS and I/O Definitions
#include <cfxpico.h>       // Persistor PicoDOS Definitions

#include <assert.h>
#include <ctype.h>
#include <errno.h>
#include <float.h>
#include <limits.h>
#include <locale.h>
#include <math.h>
#include <setjmp.h>
#include <signal.h>
#include <stdarg.h>
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include <dirent.h>        // PicoDOS POSIX-like Directory Access Defines
#include <dosdrive.h>  // PicoDOS DOS Drive and Directory Definitions
#include <fcntl.h>         // PicoDOS POSIX-like File Access Definitions
#include <stat.h>      // PicoDOS POSIX-like File Status Definitions
#include <termios.h>       // PicoDOS POSIX-like Terminal I/O Definitions
#include <unistd.h>        // PicoDOS POSIX-like UNIX Function Definitions
```

```
/*
**   Change if you are using a different A/D or chip select
*/
#define  ADSLOT   NMPCS3            // The QPB slot number (0..14)
//#define     ADTYPE   ADisMAX146    // The A-D selector from <cf1AD.h> for the 12 bit MAX146
#define  ADTYPE   ADisADS8344       //The A-D selector from <cf1AD.h> for the 16 bit ADS8344
#define  VREF 2.5               // A-D reference voltage

//Note: I commented out the following lines in my ADExamples.h file. This works better as I
//decide which converter I am using in my application source file.
/*
#define  ADSLOT   NMPCS3            // The QPB slot number (0..14)
#define  ADTYPE   ADisMAX146        // The A-D selector from <cf1AD.h>
#define  VREF 2.5               // A-D reference voltage
#define  PRCPLG
*/

#include <ADExamples.h>

CFxAD    adbuf, *ad;               //For A/D sampling

ushort   *ADDataBuf;               //Hold A/D data

/*************************************************************************\
**   main
\*************************************************************************/
int main(int argc, char **argv)
     {
     short    i;
     short    result = 0;       // no errors so far
     TUPort   *tuport;
     ushort   thebyte = 0;
     short    numchars;

     // Identify the progam and build
     printf("\nProgram: %s: %s %s \n", __FILE__, __DATE__, __TIME__);
     // Identify the device and its firmware
     printf("Persistor CF%d SN:%ld   BIOS:%d.%d   PicoDOS:%d.%d\n", CFX,
          BIOSGVT.CFxSerNum, BIOSGVT.BIOSVersion, BIOSGVT.BIOSRelease,
          BIOSGVT.PICOVersion, BIOSGVT.PICORelease);
     // Identify the arguments
     printf("\n%d Arguments:\n", argc);
     for (i = 0; i < argc; i++)
          printf("  argv[%d] = \"%s\"\n", i, argv[i]);

//   Turn on the reference for the A/D (drive -SD (shutdown) high)
     PIOWrite(28,1);

//   *************************
//   Enable the RS-232 receiver
//   *************************
//
//NOTE:  In order to use the R2AU/R212AU/R216AU RecipeCard for Auxiliary UARTs using the
//       TPU, you will need to do the following (follow along on the schematic for the RecipeCard):
//
//       1)   Make sure that the dip switch is closed for the TPU lines you are using. If,
//            for example, you are going to use AUX1 you must close 1 & 2 on the dip
//            switch in order to connect pins 34 (Tx) and 33 (Rx) to the Maxim driver chip.
//            If you are using AUX2 then you must close 3 & 4 on the dip switch.
//
//       2)   If you are transmitting only (using Tx only) then you must close dip switch 6 and drive
//       pin 29 (TPU8) high in software. To disable the driver, force pin 29 low. If you are not using the
//       transmit capability of the AUX UART (receive only) then leave dip switch 6 open and an internal
//       pulldown on /OFF will keep the driver disabled.
//
//       3)   In order to be able to receive serial data you must close dip switch 5 and drive
//            pin 30 low to enable the receive driver(s). Driving pin 30 high will disable the driver
//            as will leaving dip switch 5 open because there is a pullup on /EN on the RecipeCard.
//
     PIOWrite(30,0);
     PIOWrite(29,1);

//   Initialize the A/D
     ad = CFxADInit(&adbuf, NMPCS3, ADInitFunction);

     TUInit(calloc, free);      // give TU manager access to our heap
     cprintf("TUGetDefaultParams()->bits = %d\n",TUGetDefaultParams()->bits);
     cprintf("TUGetDefaultParams()->parity = %c\n",TUGetDefaultParams()->parity);
     cprintf("TUGetDefaultParams()->autobaud = %d\n",TUGetDefaultParams()->autobaud);
     cprintf("TUGetDefaultParams()->baud = %ld\n",TUGetDefaultParams()->baud);
     cprintf("TUGetDefaultParams()->rxpri = %d\n",TUGetDefaultParams()->rxpri);
     cprintf("TUGetDefaultParams()->txpri = %d\n",TUGetDefaultParams()->txpri);
```

```
    cprintf("TUGetDefaultParams()->rxqsz = %d\n",TUGetDefaultParams()->rxqsz);
    cprintf("TUGetDefaultParams()->txqsz = %d\n",TUGetDefaultParams()->txqsz);
    cprintf("TUGetDefaultParams()->tpfbsz = %d\n",TUGetDefaultParams()->tpfbsz);


//  Open the TPU UART
    tuport = TUOpen(TPUChanFromPin(31), TPUChanFromPin(32), 9600L, 0);
    if(tuport == 0)
    {
        cprintf("\nCannot open TPU Channel %d for receive\n\n",TPUChanFromPin(25));
    }
    else
    {
        cprintf("\nOpened TPU Channel %d for receive\n",TPUChanFromPin(33));


//      Until a break is sent on the SCI (main UART)
        while(!SCIRxBreak(10))
        {
            //Sample all eight channels of the A/D
            if(ADTYPE==ADisADS8344)
                CFxADSampleBlock(ad, 0, 8, 0, true, true, false);
            else
                CFxADSampleBlock(ad, 0, 8, 0, false, true, false);

            //Transfer the A/D data to our array
            ADDataBuf = CFxADQueueToArray(ad, (void *) QRR, 16);

            //Print all channels to SCI and TPU UART
            for(i=0;i<8;i++)
            {
                cprintf("Channel %d = %04x\n",i,ADDataBuf[i]);
                TUTxPrintf(tuport,"Channel %d = %04x\r\n",i,ADDataBuf[i]);
            }
            //To give us a little space
            cprintf("\n");
            TUTxPrintf(tuport,"\r\n");

            numchars = TURxQueuedCount(tuport);
            if(numchars > 0)
            {
                //Throw out any we get for now!!!
                cprintf("I got this from the TPU UART [%02x]\n",TURxGetByte(tuport,false));
            }

            //Two seconds between
            RTCDelayMicroSeconds(2000000L);
        }
        TUClose(tuport);
    }


    PIORead(28);

// Various exit strategies
//  BIOSReset();           // full hardware reset
//  BIOSResetToPBM();      // full reset, but stay in Pesistor Boot Monitor
//  BIOSResetToPicoDOS();  // full reset, but jump to 0xE10000 (PicoDOS)
    return result;

    }   //____ main() ____//
```