Application Report
SPRA757 - July 2001

Reading and Writing Binary Files on Targets With More Than 8-Bit Chars

Alan Davis *Software Development Systems*

ABSTRACT

The run-time support (RTS) library shipped with the code generation toolset includes functions such as fread() and fwrite() that can perform input/output (I/O) on host files. On some targets such as TMS320C2000™, TMS320C3000™, and TMS320C5000™, the size of a char on the target (16 or 32 bits) differs from the size of a byte on the host (8 bits). The tools have two basic alternatives for dealing with this mismatch: to pack host bytes into target words, or not to pack host bytes into target words. Either choice creates a different set of issues for reading and writing binary files. This application report describes the problem, the two alternative solutions, and the tradeoffs between them. It also explains our choice (not packed), the rationale for our choice, and the implications of that choice on application code.

- Affected tools: Code Generation Tools, Code Composer Studio™ Integrated Development Environment (IDE)
- Affected targets: TMS320C54x™ DSP, TMS320C55x™ DSP, TMS320C20x™ DSP, TMS320C24x™ DSP, TMS320C27x™ DSP, TMS320C28x™ DSP TMS320C3x™ DSP
- Applies to: fread() and fwrite() functions in RTS library
- Common customer problem reports:
  – “fread only reads a byte at a time”
  – “fread and fwrite don’t read the right number of bits”
  – “fwrite truncates data to 8 bits”

Contents

1 Background **1**
2 The Problem **2**
3 Workaround **2**

1 Background

The RTS library shipped with the code generation toolset includes functions that implement C’s standard input/output package (stdio). Since there is no standard framework for stream I/O on an embedded system, these functions rely on a host computer (PC, Sun, etc.) to act as an I/O server. The host uses software built into the Code Composer Studio IDE or other host application to handshake with the target and provide I/O services for it via a message-passing protocol. The studio functions are defined using streams of data of type ‘char’. The host provides I/O services using its own file system, which usually consists of streams of 8-bit bytes.

2 The Problem

A problem arises when the target's char is not the same size as the host's bytes. (On theC2000™ and C5000™ DSP platforms, a char is 16 bits; on the C3x™ DSP generation, a char is 32 bits.) To illustrate this problem, consider the following call to fwrite:

```
char data[] = { 0xAABB, 0xCCDD };
fwrite(data, 2, 1, filep); /* write 2 ’chars’ from data[] to file */
```

How does this come out in the output file? There are two options:

The first option is to maintain a one-to-one correspondence between host bytes and target chars. Since a host char is smaller than a target char, this has the disadvantage that values may be truncated when writing from the target to the host. In the example above, the result file would contain the two bytes (0xBB, 0xDD). However, it has the advantage that the number of bytes in the file equals the number of chars on the host. This advantage is most evident for programs that need be ported between the host and target since each can share the data files written on the other. For example, a file written as N chars on the host can be read by one that reads N chars on the target. *This option is the one implemented in the RTS library.*

The alternative option, which is *not* implemented in the RTS library, is to split each target char into multiple host bytes (when writing from target to host), and pack multiple host bytes into each target char (when reading from host to target). This has the advantage that the data is preserved intact without truncation. This advantage is evident for programs that run only on the target and need to read data files written by other programs that run only on the target. In the example above, the result file would contain the 4 bytes (0xAA, 0xBB, 0xCC, 0xDD). The primary disadvantage of this approach is that the number of bytes in the file does not equal the number of chars written.

3 Workaround

For target programs that need to read and write data without truncation, the best approach is to add pack/unpack code to the target application. The example above could be rewritten as:

```
char data[] = { 0xAABB, 0xCCDD };
for (i = 0; i < 2; ++i)
{
   /* write each target char out as two bytes */
   fputc(data[i] >> 8, filep);
   fputc(data[i] & 0xFF, filep);
}
```

The corresponding code to read the data would be:

```
char data[];
char buf[2];
for (i = 0; i < 2; ++i)
{
   /* pack two bytes from file into target char */
   int buf[2];
   fread(buf, 2, 1, filep);
   data[i] = (buf[0] << 8) | buf[1];
}
```

C2000, C5000 and C3x are trademarks of Texas Instruments.