1.0 Custom RAM loader Instructions

A custom RAM loader would be needed for boards that are not a standard Z-World product. Compiling a custom RAM loader requires specific settings

1.1 Compile Custom Loader

If you need to generate a custom loader you will need to compile the serdownl2.c file. This is a new loader that supports the Rabbit-3000-based boards (and 2000-based also.) The old loader, serdownl.c is still supplied and may be used if desired.

The steps are as follows:

1. In the source change use one of the following defines:
In the top of the file is a #define USE_SERIAL_X

Change 'X' to A,B,C, or D to select the serial port

For port A, uncomment the #define SERA_USEPORTD if needed.

For port B, uncomment the#define SERB_USEPORTD if needed.

For the old loader, serdownl.c, if your board uses a single 512k flash chip you must add a define for: FLASH1x512. This is not needed for serdownl2.c

2. Make change to the flash write libs:

For DC 7.3x and later and serdownl2.c:

In libs\xmem.lib disable the following block of code starting at about line 516. This is done via an "#if 0" block as shown.

(part of WriteFlashArray())

```
#if 0    // disable check of flash
        // Check if we are outside the flash(es) area
#ifndef CS_FLASH
        #ifndef CS_FLASH2
        retval = -1;                // no flash at all!
        #endif  //CS_FLASH2
#endif  // CS_FLASH
#ifdef CS_FLASH
        if (flashnum == 1) {
                if (((mbXcrBegin & MBXCR_CS_MASK) != CS_FLASH) ||
```

```
                ((mbXcrEnd & MBXCR_CS_MASK) != CS_FLASH)) {
                    retval = -1;      // flash 1 is not mapped here!
                }
        }
#else   //ifndef CS_FLASH
        if (flashnum == 1) {
                retval == -1;             // no "primary" flash!
        }
#endif  //CS_FLASH
#ifdef CS_FLASH2
        if (flashnum == 2) {
                #if (RUN_IN_RAM_CS == 2)
                retval = -1;              // CS2 is RAM, not flash!
                #else
                if (((mbXcrBegin & MBXCR_CS_MASK) != CS_FLASH2) ||
                   ((mbXcrEnd & MBXCR_CS_MASK) != CS_FLASH2)) {
                    retval = -1;      // flash 2 is not mapped here!
                }
                #endif
        }
#else   //ifndef CS_FLASH2
        if (flashnum == 2) {
                retval == -1;             // no "secondary" flash!
        }
#endif  //CS_FLASH2
#endif //if 0
```

If you want to get rid of the warning about writing to flash when compiling to RAM, comment out the warnings at the beginning of WriteFlashArray()

```
//#if (RAM_COMPILE==1)
//#warnt "Write to flash will only be simulated in RAM mode"
//#warnt "Flash is not mapped, 0-7FFFFh is mapped to RAM"
//#endif
```

3. In compile options, select:

- ✍ Code and Data in RAM
- ✍ Check user defined BIOS file and select the rabbitbiosx.c file.
- ✍ Select optimize for speed.
- ✍ Turn off pointer and array indices checking
- ✍ Disable Separate I&D
- ✍ Turn of inline I/O
- ✍ In compiler defines set DATAORG=0x3000

4. Turn off “Include debug code/RST 28 instructions” in the compile menu.

5. Select the “Define target configuration” appropriate for your board.

6. Load the serdownl2.c file and compile to bin file (serdownl3.c for protected loader.)

The bin file will be called serdownl2.bin. This file must be less than 32768 bytes. The latest lib supports a length of up to 64k.

Use the “window” “Information” menu from DC and make sure that The root code area ends at less than 0x2f00. If this address is greater, change some of the functions from XXMEM (root) to xmem until the code size is less than 2f00. Getting this value closest to 2f00 but still less will minimize the .bin file size. The serial parameters are passed at 2f00.

You can save a bit more code space by turning off the Breakpoints, Watch and ASM single-step support in the Debugger options.

There is an option to use hardware handshaking for the modem. The code will work without it.

At the top of serdownl3.c there is a set of defines to enable port A debugging. This may be useful for debugging the modem communications. But the RAM .bin file will be much larger. The DATAORG will need to be changed to 0x4000 and the define:

#define SERIAL_PARAMS 0x2f00

will have to be changed to:

#define SERIAL_PARAMS 0x3f00

The same change will be needed in the lib source. The LIB and RAM loaders must use the same settings.

Rather than using port A, is easier to just run the RAM loader as a stand-alone program as shown in the next section.

1.2 Testing or Debugging the Loader

The RAM loader can not be debugged in a users program with the IDE. This is because the library resets the board and load the RAM loader. The DC IDE will stop the board and get communication errors.

To test the loader, compile it to RAM and run it from the debugger. The loader will show information on its progress in the stdio window.

To test the RAM loader, edit the following options in the serdownl.c file:

```
// 0=direct connect, 1= answer, 2=dial
#define MODE 0
```

```
#define ID_STRING "Serial RAM loader Test"
#define BAUD_RATE 38400
// disable echo, text status codes
#define MODEM_INIT "ATE0\rATV0\rATX0\r"
#define PHONE_NUMBER "5551234"
```

These parameters should match the parameters passed from your program (the .bin file will use the parameters passed from your program not any compiled in.)

Define the USE_SERIAL_X and optionally SERX_USEPORTD in the compile defines or in the source. (see step 1 in section 4.2.)

Then compile to target in RAM using the standard BIOS with debug RSTs turned on. The stdio window of the DC IDE will show progress. This is useful for debugging the modem init string. Once it is working, use the parameters in your main program