TECHNICAL NOTE

TN224

Implementing a TCP-Based Download Manager

Disclaimer

The programs described in this document are provided as a sample field reprogramming method ***only,*** *with no guarantees that they are fail-safe. How fail-safe a system needs to be is obviously application dependent. It is unlikely this sample or future samples will meet the needs of all users. Samples systems are provided as a starting point for programmers to implement their own field reprogramming solutions.*

Introduction

This document describes a method of implementing a TCP-based download manager (DLM) and downloaded application program (DLP) to be coresident on a Rabbit-based board having two 256K flash memory chips. The DLM resides in the flash connected to /CS0. The DLP resides in the flash connected to /CS2.

Rabbit Semiconductor manufactures many board types that have two flash chips. Some examples are:

- TCP/IP DevKit Board
- RCM2100
- SmartStar

The sample DLM and DLP may also be run on boards that have only one 256K flash device: the DLM occupies the lower half and the DLP occupies the upper half. For details about using one 256K flash, please see the comments in `Samples/DOWN_LOAD/DLM_TCP.c` under "One 256K Flash Memory Overview."

An attached ZIP file, `TN224.zip`, has replacement files needed for specific situations when using Dynamic C 7.32 and a single flash chip. Someone using 2 flash chips and the Dynamic C flash file system will also want to examine the ZIP files. The file `read_me_first.txt` describes the replacement files.

Why Use this Method?

The most robust way to field reprogram a remote Rabbit target is to have a separate device such as the Rabbit Link or DeviceMate receive the program and reprogram the target using the Rabbit's bootstrap mode. This "master" board could also monitor the target Rabbit board and possibly detect when it is functioning improperly. However, that solution may be too expensive for some applications where low cost and small size are critical.

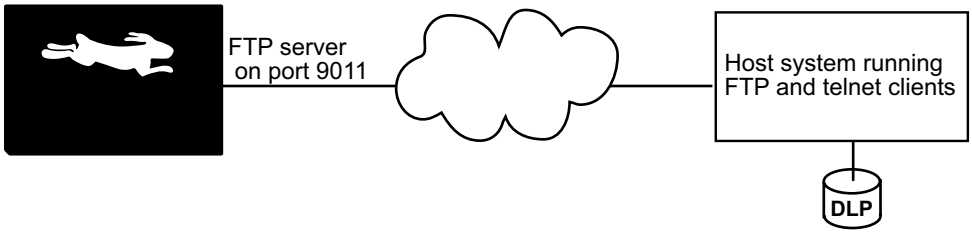
Internet Protocol Advantage

Technical Note 220 describes using a serial port to download an application program. The binary image of the DLP must be in Intel (printable) hex records. The serial port must be dedicated to downloading and monitoring for the restart signal. In contrast, when using TCP, the application image can be downloaded directly as a binary file and the Ethernet connection is not dedicated to downloading and monitoring for the restart signal. The Ethernet connection may be used in the DLP as part of the user application.

Functional Overview

The DLM is the primary program. It is loaded to the target in the normal way; i.e., using Dynamic C and the programming cable. It runs when the target is reset or powered on. When the DLM begins, it waits for an FTP client connection. The programmer sets the initial password for the FTP connection at DLM compile time. If the FTP connection becomes idle, the connection will time-out (defaults to 15 seconds) and be aborted. The DLM will then check for the presence of a valid DLP and run it if one is present.

A DLP (the secondary program) is sent to the target using FTP. A special file on the DLM side can be written to cause a restart of the DLP. Once started, the DLP monitors a telnet connection for a pass phrase telling it to restart the DLM. When the restart is executed, the DLM waits for an FTP client connection. The remote user may then run an FTP client, and, after giving the correct name and password, may upload a new DLP image and/or command a restart of the DLP.



This document makes references to memory quadrants. The primary Flash, selected by /CS0, is mapped to the first quadrant (Quadrant 0). The secondary Flash, selected by /CS2, is mapped to the second quadrant (Quandrant 1). Static RAM lives in the third and fourth quadrants. For more information on memory quadrants (aka memory banks), please see Technical Note 202, "Rabbit Memory Management in a Nutshell."

The System ID block, located at the top of the primary Flash memory, is shared between the DLM and the DLP. When the DLP starts running from the secondary flash, its copy of the BIOS will access the primary Flash to read the System ID block. Starting with Dynamic C 7.32, the User block is also shared between the DLM and the DLP. Please see the *Rabbit 2000 Designer's Handbook* or the *Rabbit 3000 Designer's Handbook* for more information about the System ID and User blocks.

Other Software Needed

To upload an application requires an FTP client program on the host that can connect to an arbitrary port. To get the DLP to restart the DLM requires a telnet client that can connect to an arbitrary port. UNIX (and UNIX-like) systems have this ability. This author tested with RedHat Linux and Microsoft Windows NT, and both their telnet and FTP clients worked fine.

Running the DLM and DLP

The sample code was included with Dynamic C version 7.32. The following files are used for this sample system:

- Samples/DOWN_LOAD/DLM_TCP.c (the download manager)
- Samples/DOWN_LOAD/DLP_TCP.c (the downloaded application program)

Configuring and Compiling the DLP

The DLP (DLP_TCP.c) is a small test program that will listen for a telnet connection on TCP port 9013.

Use Dynamic C's File menu to open DLP_TCP.c. Set TCPCONFIG[1] to access the network addresses defined in either tcp_config.lib or custom_config.lib. The IP address and netmask must be defined to the appropriate values for the Rabbit board. The IP addresses for the gateway and the nameserver should also be defined if they are needed.

Open the Options | Define target configuration dialog box. Make sure the Board ID matches the board type where the DLP will reside.

Open the Options | Compiler dialog box from the main menu. Use the "Defines" button on the lower left corner to open the Defines dialog box. Type in the following macros:

```
COMPILE_PRIMARY_PROGx ; COMPILE_SECONDARY_PROG
```

> **NOTE:** Just adding an 'x' to the macro names when they are not needed saves a lot of typing. To save these settings without interfering with other programs, use the File menu to save a new project file.

These macros are used in the BIOS source code to split the RAM and Flash between the DLM and DLP. The second macro (the two are separated by a semi-colon) tells the BIOS the program will be in the Flash off /CS2, and the RAM is only half the size the compiler says it is.

If you don't want to split the RAM between the DLM and the DLP, you must also add the macro

```
DONT_SPLIT_RAM
```

using the Defines text box as you did above. Defining this macro gives all of the RAM to whichever program is running (useful on 256K RAM systems). If DONT_SPLIT_RAM is defined in the DLM, it must be defined in the DLP and vice-versa.

Next, make sure the menu option Compile | Compile to a .bin file | Include BIOS is checked. Now compile DLP_TCP.c by selecting Compile | Compile to a .bin file, then choose the option "Compile with defined target configuration." When compilation successfully finishes there will be a file named DLP_TCP.bin in the same directory as DLP_TCP.c. This is the secondary program that will be coresident with the DLM. If your development computer is on the same subnet as your Rabbit, you can FTP the file from where it is. If not, copy DLP_TCP.bin to the computer that is running the FTP client you will use.

---

1. Alternatively, you may hardcode the network addresses in any application, using MY_IP_ADDRESS etc.

Configuring and Compiling the DLM

To download the DLM, the development computer must be connected to the Rabbit target computer.

To configure the DLM, open DLM_TCP.c. Set TCPCONFIG to access the network addresses defined in either tcp_config.lib or custom_config.lib. The IP address and netmask must be defined to the appropriate values for the Rabbit board. The IP address for the gateway should also be defined if needed.

Use the **Options | Compiler** dialog box and the **Defines** button to change the defined macros to this:

```
COMPILE_PRIMARY_PROG; COMPILE_SECONDARY_PROGx
```

If you don't want to split the RAM between the DLM and the DLP, you must also add the macro

```
DONT_SPLIT_RAM
```

using the **Defines** text box, as you did above. Defining this macro gives all of the RAM to whichever program is running (useful on 256K RAM systems). If DONT_SPLIT_RAM is defined in the DLM, it must be defined in the DLP and vice-versa. Recompile the BIOS with these new defines by pressing **<Ctrl-Y>**.

Modify the following macros at the top of the DLM source code to change the program's default behavior. See the source code for more details.

- FTP_CMDPORT: The port on which the DLM will listen for FTP requests. It defaults to 9011. If this macro is commented out, well-known port 21 will be used. Using port 9011 is a weak attempt at security: it is uncommon, and thus hides itself through obfuscation. To download the DLP to the Rabbit, the FTP client must use whatever number is chosen in the DLM, either FTP_CMDPORT, or if the macro is not defined, the DLM will be listening on port 21.

- IDLE_TIMEOUT_SECS: The number of seconds that may elapse before the DLM starts running the currently loaded DLP. Defaults to 15. If no DLP is present when the DLM times out, the DLM will restart itself. During testing you will probably want to increase the value of this macro.

- USE_DHCP: Whether to use DHCP or have the IP address hard-coded. The default is to not use DHCP. If you do use it, you must know what IP address will be assigned so that you can FTP to the DLM later on. The easiest method is to set up the DHCP server to obtain a permanent lease for the MAC address that corresponds to your Rabbit board.

- USER_NAME and USER_PASSWORD: User name and password requested by the FTP server. The default name is "Rabbit" and the default password is "123." Currently, the password cannot be changed at runtime.

The function UserTaskTick() is called by the DLM while it is idle. Currently the function does nothing, but you may modify it to do whatever you want. See the source code for detailed specifications.

Now compile DLM_TCP.c to the target. After the program loads, power down the target (or hard reset it), remove the programming cable and repower the board. Disregard the error this causes Dynamic C to report when it loses target communications.

Using the Download Manager via FTP

When the DLM begins execution a timer starts running. After 15 seconds of no activity (defined by IDLE_TIMEOUT_SECS), the DLM will try to restart the DLP. If the stored CRC does not match the computed CRC value of the DLP, the DLM restarts itself instead. A remote host has 15 seconds to connect with FTP and do some file operation, such as a directory listing. If the connection is ever idle for 15 seconds, the DLM will attempt a restart of the DLP.

Once the FTP connection is made, you can upload a new DLP image to the target. First, make sure the transfer mode is BINARY. Then "put" the file on the remote end. The FTP session will look similar to:

```
C> ftp
ftp> open 10.10.6.1 9011
Welcome to Z-World TinyFTP!
Login (unknown): Rabbit
Password: 123

ftp> dir
   . . .
ftp> binary
200 IMAGE mode set

ftp> put DLP_TCP.BIN image.bin
   . . .
226 Transfer OK. Got 90408 bytes
90408 bytes sent in 5.09 secs (17 Kbytes/sec)

ftp> get reboot.me
   . . .
ftp> put reboot.me
   . . .
421 Service not available, remote server has closed connection
```

NOTE: Many FTP clients do not accept a port number on the command line. That is why the interactive `open` command is used. Also, notice that the put command was used to rename `DLP_TCP.bin` to `image.bin` on the remote end.

Troubleshooting

If you receive an error message trying to upload the `reboot.me` file (i.e. "`get reboot.me`"), the DLP image may be corrupted. Upload `reboot.me` again. If the error repeats, upload the file named `status` for a description of the problem. For instance, type the command "`get status -`" at the FTP prompt. This causes the contents of `status` to display on Stdout. The contents of `status` provides information on the last file action attempted.

If you notice the link status LED going off every 15 seconds or so, that is the DLM trying to reboot, but detecting an invalid DLP. Try uploading the DLP to the target again.

If the board reboots but the DLP doesn't appear to run, it might be your DLP has not been compiled for the correct target board. From the GUI, select **Options | Define target configuration** to set the board type correctly, then follow the rest of the directions for "Configuring and Compiling the DLP" to create a new image to upload to the target.

Restarting the DLM

If the above FTP session was successful, the DLM restarted after receiving `reboot.me`. The contents of this file are irrelevant: the restart signal is the receiving of a file named `reboot.me`, not what it contains. An appropriately small file is provided as a convenience. The "get reboot.me" command stores the file on your computer so that you can "put" it back on the Rabbit to signal the DLM to restart itself.

A restart is also initiated by simply waiting for `IDLE_TIMEOUT_SECS` seconds to pass. On a restart, the DLM will close the FTP connection, power down the Ethernet controller chip and jump to the DLP program.

If all went well, your downloaded application (the DLP) is running. The DLP is in the second Flash (connected to the /CS2 line), and the SRAM memory it uses is now the fourth quadrant. (Unless `DONT_SPLIT_RAM` was defined, which means that the DLP is using all of the RAM.)

Note that both the DLM and DLP will share the same System ID block.

If you upload the DLP image in ASCII mode, the image will be invalid. However, the CRC will be correct because it reflects what was transferred, not the original file contents read by the FTP client. The DLP will be started, but the program is invalid and interrupts will likely remain disabled. Hopefully, the watchdog timer will expire and the board will restart the DLM.

Telnet to the DLP

The DLP is contacted using telnet. The sample DLP listens on port `USE_TELNET_PORT`. Whenever a connection occurs, the DLP generates a challenge puzzle and presents it to the client. For the sample code, you type back the 8 digit number (without the comma). If the response is correct, the DLP will shutdown and the target will reboot back to the DLM. This allows a different DLP image to be downloaded. If the response isn't correct or `PUZZLE_MASTER_TIMEOUT` seconds have elapsed, the DLP will close the connection and listen for a new telnet connection.

There is no direct feedback to the telnet client saying whether or not the puzzle was solved. If it was solved you will be able to FTP to the DLM. If it was not solved you will only be able to telnet to the DLP to try the puzzle again. If you attempt to open an FTP connection to the DLM, it will be refused.

Functional Details

Some of the details involved in having coresident programs for field reprogramability are discussed here.

Program Verification

As the FTP server receives the DLP program, a running 16-bit CRC is maintained. The program size, starting address (always 0x40000 from the point of view of the DLM), and CRC are stored in flash in the User block area. A CRC check is computed on the stored DLP program before attempting to run it. If the computed CRC is different, the DLM will be restarted instead.

BIOS Changes

The following code is in BIOS\RABBITBIOS.C. If the DLP is being compiled, a check is made for a single flash device. Unless told not to, the BIOS splits the RAM in half, giving the first half to the DLM and the following half, or upper half, to the DLP.

```
#ifdef COMPILE_SECONDARY_PROG
   #ifdef INVERT_A18_ON_PRIMARY_FLASH     // single flash, split in half
      #undef MB0CR_INVRT_A18
      #define MB0CR_INVRT_A18 1
      #undef FLASH_SIZE                   //  assume 256K flash
      #define FLASH_SIZE 0x20             //  128K partition for DLP
      #undef CS_FLASH
      #define CS_FLASH 0x00
   #else                                  //  2 flash chips; one here, one there
      #undef  CS_FLASH
      #define CS_FLASH   0x02
      #undef  CS_FLASH2
      #define CS_FLASH2  0x00
   #endif
   #ifndef DONT_SPLIT_RAM                 //  RAM should be split, determine
      #if (_RAM_SIZE_ == 0x80)            //  RAM size, i.e., number of 4K
         #undef _RAM_SIZE_                //  RAM pages available on board
         #define _RAM_SIZE_ 0x40          //  take half from the DLM
      #else
         #if (_RAM_SIZE_ == 0x40)
            #undef _RAM_SIZE_
            #define _RAM_SIZE_ 0x20
         #else
            #if (_RAM_SIZE_ == 0x20)
               #undef _RAM_SIZE_
               #define _RAM_SIZE_ 0x10
            #else
               #error "unknown RAM size"
            #endif
         #endif
      #endif
      #undef RAM_SIZE
      #define RAM_SIZE _RAM_SIZE_         //  give DLP half of RAM
      #undef RAM_START                    //  locate DLP's RAM in the
      #define RAM_START 0x80+RAM_SIZE     //  upper half.
   #endif
#endif
```

The above redefined macro values are used later in the BIOS to set up memory mapping information in origin directives used by the compiler. A similar block of compiler directives exists for compiling the DLM:

```
#ifdef  COMPILE_PRIMARY_PROG
  ...
```

The rest may be viewed in the BIOS source code. COMPILE_PRIMARY_PROG and COMPILE_SECONDARY_PROG also show up in IDBLOCK.LIB; they affect how the System ID block. is accessed.

Monitoring for the Restart Signal

The DLP must monitor for the condition that signals a DLM restart. The DLM is also set up to do this, but it is less critical since the DLM will time-out and restart either itself or the DLP if a valid DLP exists. The DLP uses telnet and a challenge-response scheme to verify that the remote user is allowed to restart the DLM. When a telnet connection is requested, the DLP issues the challenge.

Generally, the algorithm used to correctly respond to the challenge is kept secret. The response itself should not provide clues to an eavesdropper as to what the algorithm is doing.

The following macros are in the DLP:

```
#define CRYPT(x) ((x) ^ 0x98765432L)
#define  ALGO(x)   (x)
```

The first macro provides an encryption/decryption function. This has the advantage that the challenge is not sent in cleartext. The second macro is an algorithm that solves the challenge. In this example, no change is made to "x." Obviously, the implementation in this sample system is not meant to be industrial strength; just enough to discourage casual access. For a more secure implementation, see RFC1760 for the S/KEY algorithm.

Sources of "x" may be the time of day, a string, or a hash function such as MD5. Replay attacks are discouraged because "x" is different on every attempt. The first few bytes of the cleartext challenge may be random to introduce a bit of entropy. If the challenge is long enough, the last byte can hold a checksum of either the cleartext or ciphertext.

There is no need to poll for a restart signal. That is only done when using a serial communication channel. Using a telnet connection, the remote end has a limited number of seconds to respond correctly to the challenge. When a correct response is received, the DLP closes the telnet connection and restarts the DLM. If the response is incorrect, the DLP closes the telnet connection, but instead of restarting the DLM, the DLP continues to run, listening for a new telnet connection. The DLP will stay in this endless loop until it receives the correct response to its challenge. If for some reason you can't give the correct response, you must reset the board by asserting the reset externally or by cycling power. Either one will cause the DLM to restart.

DLM Restart Details

The function RestartDLM() is in both the DLP and the DLM. In both cases the goal is the same: to execute the DLM code, which resides at 0x0000. For the DLM this is easy. In a small assembly block, interrupts are turned off and a call is made to address zero. That's it. The download manager is restarted.

For the DLP, reaching the goal is more complex. To run the DLM, the DLP must copy code to RAM that will switch to the beginning of the primary flash. This is done by allocating a root buffer to hold the code and then calling memcpy() to place the flash switching function into the buffer. The flash switching function sets bits in the memory bank control register (MB0CR) to access the beginning of the primary flash and then jumps to it to run the DLM code.

Whereas your average application does not need to be aware of separate I&D space, the DLP does need to be aware of it because it runs code in the data space. The strategy for this situation is to temporarily disable separate I&D space while the flash switching function is running.

Possible Complications

There are several situations to take into consideration when creating a DLM and/or a DLP.

Stuck in a Loop

If the following code fragment is run, the Rabbit board will be locked in a tight loop that cannot be exited without cycling power or asserting the reset pin.

```
#asm
   ipset 3          ; turn off interrupts
.tightLoop:         ; only a reset will get out of this!
   call hitwd
   jr .tightLoop
#endasm
```

Interrupts are turned off, so the periodic ISR will not run and virtual watchdogs cannot time-out and cause a needed reset. The hardware watchdog gets hit in the tight loop, so it can't time-out and cause a reset either. There's no way out of the tight loop except asserting the reset externally or cycling power. It would be silly to have a piece of code that did this in either the DLM or DLP, but careless programming could result in a more complex set of instructions that have the same result. The best way to avoid this would be to use virtual watchdogs in your programs, let the periodic interrupt take care of hitting the hardware watchdog timer, and take great care not to create situations where interrupts could be shut off permanently.

When the watchdog timer expires, the target board will be restarted in the DLM. If the DLP image is still good (computed CRC is same as stored value), then it will be restarted.

Power Failure

The possibility of a power failure at the wrong time is something that any field reprogramming method should take into consideration. In this sample, both the DLM and DLP write only to the top half of the flash. Even huge sector flashes generally don't have a sector that crosses the 128K boundary, so there is no possibility that the DLM will be corrupted by a power failure while writing (or after erasing, but before writing) a sector in the DLM.

References

To use a serial-based system:

See Technical Note 218A "Implementing a Serial Download Manager for a 256K Byte Flash" for a one flash chip solution.

See Technical Note 220 "Implementing a Serial Download Manager for Two 256K Byte Flash Memories" for a two flash chip solution.