TECHNICAL NOTE

TN218a

Implementing a Serial Download Manager for a 256K Byte Flash

(For Dynamic C version 7.31 and Later)

Disclaimer

*The programs described in this note are provided as a sample **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 all users' needs. Sample systems are provided as a starting point for programmers to implement their own field reprogramming solutions.*

Introduction

This technical note describes a method of implementing a serial Download Manager (DLM) and downloaded program (DLP) as a pair of coresident programs on a Rabbit 2000 based board having a single 256K byte flash chip. The DLM/DLP system is included in `Samples/DOWNLOAD`.

The method described herein requires that pin A18 on the Rabbit processor be connected to pin A17 on the flash chip. Many Rabbit board level and core module products have a jumper for switching this line. The Rabbit Memory Interface Unit (MIU) has a feature that allows A18 to be inverted for all memory accesses within selected quadrants of the 1M physical address space by setting a bit in an internal I/O register. The method explained here combines the physical rerouting of A18 in hardware with the MIU inversion of A18 to allow both co-resident programs to be compiled to address 0x0000, but actually reside in separate halves of the 256K byte flash.

Board Types Supported

Many Rabbit Semiconductor board types with a single 256K flash have a jumper for connecting A18 on the bus to A17 on the flash. The jumper (0 Ω resistor) is set in the normal A18-A18 position in the factory and needs to be resoldered to implement this DLM. A factory option for shipping with the A18-A17 configuration would be considered for a volume order.

Why Use this Method?

The most robust way to field reprogram a remote Rabbit target is to have a separate device such as the Rabbitlink receive the program and reprogram the target using the Rabbit 2000's bootstrap mode. That solution may be too expensive for some applications where low cost and small size are critical. Another option is to use a board with two flash chips, and keep the unchanging resident DLM in one of the flash chips. Starting with Dynamic C version 7.05, a fully functional sample DLM for two-flash boards comes with Dynamic C in the subdirectory `Samples\DOWN_LOAD`. That method may also be too expensive in parts cost and size for some applications.

For an application using a board with a single 256K flash, the fact that Dynamic C does not create relocatable programs is a problem because the BIOS must start at 0x0000 so that board and program initialization code runs correctly. But sharing the BIOS between two coresident programs creates potential problems with Dynamic C version compatibility if the permanently resident DLM compiled with an older version remains unchanged, while a secondary program which may be built using later versions shares the BIOS with the DLM. The method described in this note allows the DLP and DLM to both start at logical address 0x0000, each with their own BIOS.

Internet Protocol Alternatives

The techniques used in this serial method are adaptable to an Internet protocol based DLM/DLP system. It is unnecessary to transmit the DLP in HEX format with checksums when a reliable transport protocol is used; a BIN image file suffices. Using a TCP-based DLM removes much of the complexity in the current sample, since the details of reliable transmission are hidden in the Rabbit provided TCP/IP libraries.

Sample DLM/DLP systems for one and two flash boards that work over TCP/IP are provided with Dynamic C starting with version 7.32. See Technical Note 224, "Implementing a TCP-Based Download Manager," for instructions on using these samples. The TCP DLM is a tight fit into 128K of split flash and requires reducing the memory for the System ID block, so we recommend customers use two 256K flash chips in their solution.

Functional Overview

The Download Manager (DLM) is the primary program, the program that runs when the target is reset or powered on. The DLP (or secondary program) is transmitted using RS232 (over serial port B or C in these examples) in an Intel Hex file format. (See "Appendix: The Intel HEX File Format" at the end of this document for details.)

Both the DLM and DLP monitor the channel for non-ASCII characters (bytes with the high bit set) in order to detect a predefined/user-definable command string of user-definable length that is understood to be the restart command. When the restart command is detected, the DLM starts. A menu is displayed on a terminal emulator screen, giving the user a choice of actions. The first action either must be to enter a password or restart the DLP. These are the only actions that may be performed without entering a password first. If no password is entered within a user-specifiable time limit, the DLM checks for the presence of a valid DLP and runs it if one is present, or restarts itself if one is not. The user can set the initial password at DLM compile time, and specify whether the password will be run-time changeable.

Other Software Needed

A terminal emulation program with the ability to send a raw ASCII or binary file, handle ANSI escape sequences and use software flow control (XON/XOFF) is needed. A good terminal program called Tera Term was used to test these samples.

```
http://hp.vector.co.jp/authors/VA002416/teraterm.html
```

(In the author's opinion, HyperTerminal is notoriously cranky and should not be used.)

Software flow control is required so that receiving the program data can be paused while interrupts are turned off during a write to a flash sector. This is necessary to prevent interrupt service routines from running code in flash while the flash is being written.

Running the DLM and DLP

Look in Samples/DOWN_LOAD for these files:

- `DLM_256kFlash.C` - The primary, or DLM program.
- `DLP_256kFlash.C` - A simple secondary, or DLP program.
- `RESTART.BIN` - contains the DLM restart command which when sent to the target will signal either the DLM or DLP to restart the DLM.

The following numbered list walks through the hardware and software setup.

1. Hardware Set Up (Modemless)

Make a 3-wire connection between the target board and the PC by connecting TX of the serial channel being used on the Rabbit target board to the RX line of the PC COMM port being used. RX on the target goes to the TX line of the PC, and a common ground is the third connection. The samples are set up to use serial port B or C, uncomment the appropriate macro near the tops of the DLM and DLP samples to select the port.

```
//*** uncomment one only! ***
#define USE_SERIAL_PORT_B
//#define USE_SERIAL_PORT_C
```

Find the 0 Ω jumper on your board that connects A18 from the processor to A18 on the flash and change it to connect A18 on the processor to A17 on the flash.

2. Running the Terminal Emulator

Set your terminal emulation program for 57600 baud, one stop bit, no parity, ANSI terminal emulation, XON/XOFF flow control, a transmit delay of 1 ms/line and the appropriate COM port.

Slower boards may require a slower baud rate. Change the baud rate with this macro in `DLP_256kFlash.c` and `DLM_256kFlash.c`:

```
#define SERIAL_BAUDRATE 57600ul
```

3. Compiling DLP_256kFlash.C

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

```
COMPILE_PRIMARY_PROGx ; COMPILE_SECONDARY_PROG ;
INVERT_A18_ON_PRIMARY_FLASH ; ROUTE_A18_TO_FLASH_A17
```

Hint: Just adding an 'x' to the macro names when they are not needed saves a lot of typing. Save a new project file for added convenience later on.

These macros are used in the BIOS source code to split the RAM between the DLM and DLP, and to invert A18 when compiling the DLP and the BIOS so that they work correctly when located in the top half of the 256K flash chip.

To give the DLP access to all of the available RAM (i.e., no sharing of RAM with the DLM) add

```
DONT_SPLIT_RAM
```

to the macros listed in the Defines text box. If this macro is defined in the DLP, it must be defined in the DLM also.

Open `DLP_256kFlash.C` and compile it by selecting Compile | Compile to a .bin file. Choose the option, Compile with defined target configuration rather than Use attached target. When compilation finishes there should be a file called `DLP_256kFlash.HEX` in the same directory as `DLP_256kFlash.C`. This is the secondary program that will be coresident with the DLM.

4. Compiling DLM_256kFlash.C

To compile the DLM open DLM_256kFlash.C. Use the Options | Compiler dialog box and the Defines button to change the macros to this:

```
COMPILE_PRIMARY_PROG; COMPILE_SECONDARY_PROGx;
INVERT_A18_ON_PRIMARY_FLASH ; ROUTE_A18_TO_FLASH_A17
```

To give the DLM access to all of the available RAM (i.e., no sharing of RAM with the DLP) add

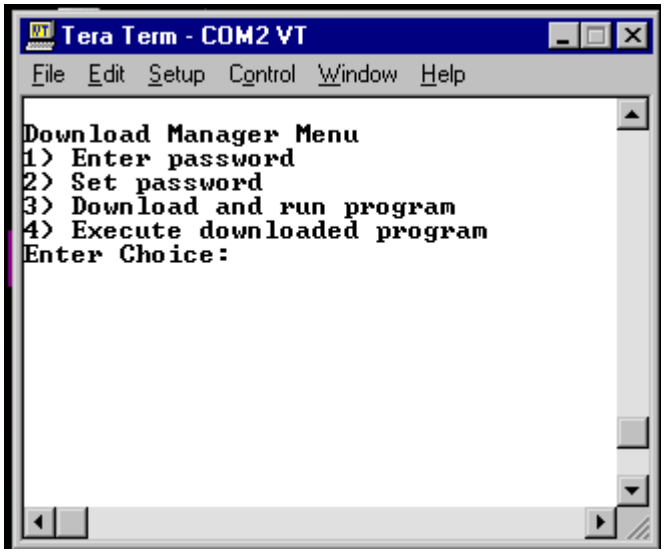
```
DONT_SPLIT_RAM
```

to the macros listed in the Defines text box. If this macro is defined in the DLM, it must be defined in the DLP also.

Recompile the BIOS with <Ctrl-Y>. Now compile DLM_256kFlash.C to the target. After it loads, power down the target, remove the programming cable and repower the board.

Using the Download Manager Menu via the Terminal Emulator

The following menu should appear in the terminal emulator window.



Type “1” and at the prompt enter the default password “123” followed by <Enter>.

Now type “3” and see the prompt “Send HEX file as raw ASCII or Binary.” Use the terminal emulator to send the file DLP_256kFlash.HEX. After it loads, you will see a message that a valid DLP was found and is starting, then a continuous stream of ‘X’s.

To restart the DLM, use the terminal emulator to send the file RESTART.BIN. This file has the default byte sequence that the DLM and DLP interpret as a restart command. The menu will be refreshed on the terminal afterwards. If you do nothing for 10 seconds, the DLP will start again. If you choose option 4, the DLP will also run again. Or you can enter the password again and send another program or change the password.

Configuring the DLM

The following properties can be easily modified by changing macros at the top the DLM source code, see the source code for details:

- Password time-out period - the time allowed to enter the password after the DLM starts before running the currently loaded DLP, if present.
- Default password - the password used when the DLM is run for the first time.
- Minimum password length allowed
- Maximum password length allowed
- Whether the password can be changed at run-time
- The DLM restart command sequence
- General time-out period - the period of inactivity allowed after the password is entered before the currently DLP is run.
- Error message delay - the time which error messages are displayed for.

In addition, the following functions may be redefined by the user to adapt the DLM for different serial channels or other communication means. See the source code for detailed specifications:

```
PutChar(), GetChar(), UserTask()
InitComm(), CloseComm(), PauseComm(), ResumeComm()
```

Functional Details

This section discusses some of the implementation details and issues involved in a Download Manager system.

BIOS Changes

The following conditional macro redefinitions are in `BIOS\RabbitBios.c` in Dynamic C 7.31.

```
#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 a 256K flash
      #define FLASH_SIZE 0x20               // 128K partition for DLP
      #undef  CS_FLASH
      #define CS_FLASH   0x00
   #else                                    // 2 flash; one here, one there
      #undef  CS_FLASH
      #define CS_FLASH   0x02
   #endif
   #ifndef DONT_SPLIT_RAM
      #if (_RAM_SIZE_==0x80)                // The rest of the directives
         #undef _RAM_SIZE_                  //  split RAM and make the start of
         #define _RAM_SIZE_ 0x40            //  RAM for the DLP begin after
      #else                                 //  the RAM used for the DLM
```

```
            #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
         /* Locate the DLP's RAM in upper half of it. */
         #undef  RAM_SIZE
         #define RAM_SIZE _RAM_SIZE_
         #undef  RAM_START
         #define RAM_START 0x80+RAM_SIZE
      #endif   // ifndef DONT_SPLIT_RAM
   #endif
```

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

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

The code can be viewed in `RabbitBios.c`. The directives cause the DLM to lose half of the available flash to accommodate the DLP. The RAM may or may not be divided in half, depending on the existence of `DONT_SPLIT_RAM`.

Memory Management

Technical Note 202 explains how the Rabbit Memory Management Unit (MMU) and the Memory Interface Unit (MIU) work. The trickiest part of the method used here is the inversion of A18 in quadrant 0. What this inversion does is cause A18 to be asserted for all accesses in physical memory quadrant 0 (addresses 00000-3FFFF). Since A18 is connected to A17 on the flash, when this inversion is in effect, all accesses in quadrant zero will cause A17 on the flash to be asserted, and therefore address 00000-3FFFF will really access the top half of the 256K flash, since 2^{17} = 128K (131072) . So when we are running the DLM in the lower half of flash and we want to start the DLP in the upper half, all we have to do is call a function that sets the Memory Bank Control Register for quadrant 0 (MB0CR) to invert A18 (it has a bit for this purpose) and then jump to address 0. Of course this function must be run from RAM, or else as soon as the MB0CR A18 inversion is enabled, execution from flash would proceed in the upper half of flash, and it is unlikely the correct jump instruction will happen to be there!

Likewise, when we are running the DLP in the upper half of flash and want to start the DLM, we run a RAM resident function that clears the MB0CR A18 inversion bit and jumps to address 0x0000.

When the DLM has to access the top half of the flash to write the DLP or other data, the method used will be to make sure quadrant 1 (40000-7FFFF) addresses CS0/OE0/WE0 (which the flash is connected to), and use quadrant 1 without enabling A18 inversion in the MIU. Since address bit 18 is set in the

quadrant 1 address range, and A18 is crossed with A17, those addresses will really correspond to offsets 20000-3FFFF in the flash chip.

Monitoring for the DLM Restart Signal

It is the responsibility of the DLP to monitor for the string that signals a DLM restart. The DLM is also set up to do this, but this is less critical since the DLM will time-out and restart itself, or the DLP, if a valid restart string is present, if no password is entered or no data is received

The following macros are needed in the DLP.

```
#define RESTARTSIGNAL "\xaa\xbb\xcc\xbb\xaa"
#define RESTARTSIGNAL_TIMEOUT 30
```

The restart signal can be made longer or shorter, with the requirement that it be at least one byte long and all bytes have the high bit set. It can be sent from the terminal as keyboard combinations if the terminal program allows that, or as a file. An easy way to create a file containing the reset signal is to use the Dynamic C debug options to log Stdout (without appending) to a file, and run this program:

```
main(){
   printf("\xaa\xbb\xcc\xbb\xaa");
}
```

The following functions are also needed in the DLP.

```
void ProcessRestartCommand(char ch)
void RestartDLM()
```

The DLP sets up and monitors the download channel. It calls `ProcessRestartCommand()` each time it receives a byte with the high bit set. The sample DLP hits a virtual watchdog timer in the tasks that check the serial channel for input so that the board will reset if it is not entered periodically. `Process-RestartCommand()` is the same in both the DLM and the DLP. The function `RestartDLM()` is also in both the DLM and DLP and though the goal is the same—to execute the DLM code which resides at 0x0000—the means of reaching it differ.

For the DLM it's 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 address zero to run the DLM code.

Program Verification

The Intel HEX format file containing the program (DLP) has a one byte checksum for each 32 byte fragment of code. Verification of the checksum is performed as the data is received. An overall 16 bit CRC is also calculated as the data is received. The program size, starting address (always 40000h from the DLM's point of view), and CRC are stored in flash in the User ID Block area. A CRC check is made on the stored DLP program before attempting to run it. If the CRC is not valid, the DLM will be restarted instead.

Please see Appendix: The Intel HEX File Format for more information.

Virtual Watchdog

The DLM and DLP both enable a single virtual watchdog timer to ensure that the programs don't enter a "hung" state. The periodic interrupt ISR normally hits the hardware watchdog timer in Dynamic C programs.

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 recycling 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.

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.

When the password is changed and written to flash, the DLM uses the `writeUserBlock` function. Starting with Dynamic C version 7.20, this function will do redundant writes to separate sectors to store persistent data to protect against data loss if a power failure occurs at the wrong moment. The redundant copies in the User block are only done if the board has a version 3 (or later) System ID block. A version 3 (or later) ID block is highly recommended. The difference between version 3 and version 2 is the version number itself. It can be examined in the structure member `SysIDBlockType.tableVersion`. The System ID block may be rewritten to be a later version using `write_idblock.c`, which is included in the attached ZIP file, `TN218a.zip`.

Modem Failure

These samples make no attempt to exercise any control over any modem device. This is left to the user if it is needed. A danger of using a modem for field reprogramming is that the modem could get into a hung state. It might be a good idea to use a modem with an external reset, and have the DLP occasionally assert the reset if no data has been received for a long while. The worst that could happen would be that a reset would occur just as a remote user was starting to reprogram, and they might need a retry in order to establish the connection.

File System

The older version of the file system (`filesystem.lib`) has not been tested with this scheme and may have problems. To use the new file system (`FS2.lib`) requires changes to `RabbitBios.c` and `FS_DEV.LIB`. These files are included in the attached ZIP file, `TN218A.zip`. These replacement files are necessary if you are using Dynamic C 7.31 or 7.32. If you are using a later version of Dynamic C, the changes have been incorporated into the BIOS and the file system library.

Writing the User Block

The User block may be written if you have Dynamic C 7.32 and you replace the library `Flashwr.lib`. The replacement file is included in the attached ZIP file.

Appendix: The Intel HEX File Format

The Hex file format consists of ASCII records of the following format:

:NNAAAATTDD$_1$DD$_2$DD$_3$...DD$_N$CC

A colon starts every record. Each letter represents a hexadecimal nibble with the following meanings.

NN - Number of data bytes in record. For Dynamic C generated hex files, this always either 02 for extended address records, 20 for data records, or 00 for EOF records.

AAAA - 16 bit address. This is the offset portion off the destination address using the Intel real-mode addressing. The segment portion of the real-mode address is given by the last extended address record in the HEX file previous to the data record. The physical offset into the memory device is computed by shifting the segment left 4 bits and adding the offset.

TT - Type of record. For Dynamic C generated hex files, this always either 02 for extended address records, 00 for data records, or 01 for EOF records.

DD$_i$ - Data byte

CC - 8 bit checksum of all previous bytes int the record. The two’s complement of the checksum is used.

Examples

There are three types of records: extended address, data and end of file. Each record starts with a colon. The Address field is only meaningful for data records.

1. Extended Address Record

This is the first record in a Dynamic C generated HEX file, and applies to all addresses that come after it until another extended address record is found.

```
:020000020000FC
```

Table 1. Extended Address Record

Data Length	Address	Record Type	Data	Checksum
02	0000	02	0000	FC

2. Data record

Most of the records in a hex file will be of this type.

```
:200000003D183090DE3DBD803D7B803D7E80FFFF3DF3EE00000000003DF39000
   0000000025
```

Table 2. Data Record

Data Length	Address	Record Type	Data	Checksum
20	0000	00	3D1830 . . . 00	25

3. End of file record

This is the last record in a hex file

```
:00000001FF
```

Table 3. End of File Record

Data Length	Address	Record Type	Checksum
00	0000	01	FF

References

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

See Technical Note 224 “Implementing a TCP-Based Download Manager” for a network-based solution.