TECHNICAL NOTE

TN220

Implementing a Serial Download Manager for Two 256K Byte Flash Memories

Disclaimer

The programs described in this note 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 note describes a method of implementing a serial Download Manager (DLM) and downloaded program (DLP) as a pair of coresident programs on a Rabbit-based board having two 256K byte flash chips. The DLM resides in the primary flash hooked to CS0, and the DLP resides in the other flash. This sample code was introduced with Dynamic C 7.06.

Rabbit Semiconductor manufacture many board types that have two 256K flash chips. Some examples are:

- TCP/IP DevKit Board
- RCM2100
- SmartStar

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's bootstrap mode. That solution may be too expensive for some applications where low cost and small size are critical.

Internet Protocol Alternatives

The techniques used in this serial method could be easily adapted to an Internet protocol based DLM/DLP system. It is not necessary to transmit the DLP in HEX format with checksums if a reliable transport protocol is used; a BIN image file suffices. This would remove much of the complexity in the current sample, since the details of reliable transmission are hidden in the provided TCP/IP libraries. Such a DLM is provided starting with Dynamic C 7.32. See Technical Note 224, "Implementing a TCP-Based Download Manager," for details.

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 for details.) Both the DLM and DLP monitor the channel for non-ASCII characters (bytes with the high bit set), to detect a pre-defined/user-definable command string of user-definable length which is understood to be the restart command. When the restart command is detected, the DLM starts.

A menu is displayed on the terminal emulator screen when the DLM starts, giving the user a choice of actions. The first action either must be to enter a password, or restart the DLP. Restarting the DLP or entering a password are the only actions allowed 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 emulator program with the ability to send a raw ASCII or binary file and use software flow control (XON/XOFF) is needed. A good terminal program called Tera Term was used for 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 when a flash sector is written.

Running the DLM and DLP

The following files are used for this system:

- `Samples/Down_load/DLM_SERIAL_2_FLASH.C` - The primary program (the DLM).
- `Samples/Down_load/DLP_SERIAL_2_FLASH.C`[1] - A simple secondary program (the DLP).
- `Samples/Down_load/RESTART.BIN` - contains the DLM restart command which when sent to the target will signal either the DLM or DLP to restart the DLM.

1. DLP_SERIAL_2_FLASH was named DLP_2_FLASH prior to Dynamic C 7.30.

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 COM 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 code to select the port.

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

2. Running the Terminal Emulator

Set your terminal emulator for 57600 baud, one stop bit, no parity, ANSI terminal emulation, XON/XOFF flow control, transmit delay of 1 ms and the appropriate COM port. Slower boards may require a slower baud rate. The baud rate is changed with a macro named SERIAL_BAUDRATE. It is in DLP_SERIAL_2_FLASH.C and DLM_SERIAL_2_FLASH.C.

```
#define SERIAL_BAUDRATE 57600ul
```

3. Compiling DLP_SERIAL_2_Flash.C

Open the Options | Compiler dialog box from the main menu. 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
```

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.

If you are using Dynamic C 7.30 (or later) and don't want to split the RAM between the DLM and the DLP, you must also add the macro

```
DONT_SPLIT_RAM
```

to the macros listed in the Defines text box. 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 DLP, it must be defined in the DLM and vice-versa.

Use Dynamic C's File menu to open DLP_SERIAL_2_FLASH.C. Select the correct Board ID with the Options | Define target configuration dialog box. (e.g., RCM2100).

Now compile DLP_SERIAL_2_FLASH.C by selecting Compile | Compile to a .bin file. Choose the option, Compile with defined target configuration. When compilation finishes there should be a file called DLP_SERIAL_2_FLASH.HEX in the same directory as DLP_SERIAL_2_FLASH.C. This is the secondary program which will be coresident with the DLM.

4. Compiling DLM_Serial_2_Flash.C

To compile the DLM, open DLM_SERIAL_2_FLASH.C. Use the Options | Compiler dialog box and the Defines button to change the macros to this:

```
COMPILE_PRIMARY_PROG; COMPILE_SECONDARY_PROGx;
```

If you are using Dynamic C 7.30 (or later) and don't want to split the RAM between the DLM and the DLP, you must also add the macro

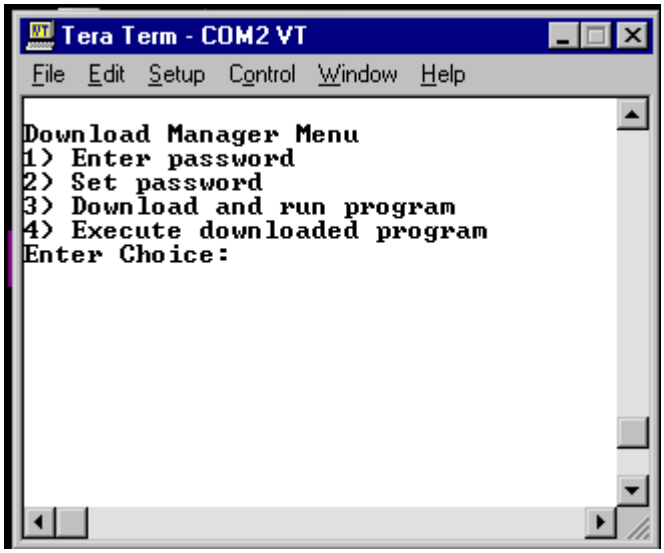
```
DONT_SPLIT_RAM
```

to the macros listed in the Defines text box. 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 using <Ctrl-Y> so that it can incorporate the newly defined macros. Now compile DLM_SERIAL_2_FLASH.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 then 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_2_Flash.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 should appear refreshed on the terminal after sending this file. If you do nothing for 10 seconds, the DLP will start again. If you type "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:

```
int PutChar(char ch)
int GetChar(char ch)
int InitComm()
int CloseComm()
int UserTask()
int PauseComm()
int ResumeComm()
```

Functional Details

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

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 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.

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.

BIOS Changes

The following conditional macro redefinitions are in the file RABBITBIOS.C, in Dynamic C 7.32.

```
#ifdef  COMPILE_SECONDARY_PROG
   #if RAM_COMPILE==1
      #error "For DOWN_LOAD sample only. Not compatible with RAM_COMPILE."
   #endif
   #ifdef  INVERT_A18_ON_PRIMARY_FLASH
      /* Single flash, split in half */
      ...
   #else                                  //  Two flash memory chips; one here, one there
      #undef  CS_FLASH
      #define CS_FLASH   0x02          //  This is the chip select (CS) for 2 flash boards
      #undef  CS_FLASH2
      #define CS_FLASH2  0x00
   #endif
   #ifndef DONT_SPLIT_RAM
      #if (_RAM_SIZE_==0x80)           //  These directives split RAM and make
         #undef _RAM_SIZE_             //     the start of RAM for the DLP begin after
         #define _RAM_SIZE_ 0x40       //     the RAM used for 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
      /* Locate secondary'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 here 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 rest can be viewed in the BIOS source code. For the DLM, the change that the directives cause from a normal compilation is that the available RAM is cut in half to accommodate the DLP if the macro DONT_SPLIT_RAM is undefined.

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 it's 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 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");
}
```

This file is provided with Dynamic C in Samples/DOWN_LOAD/RESTART.BIN.

The following functions are part of 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. ProcessRestartCommand() 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 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 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 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. But in version 7.05, no such redundancy exists. So if the end user entered a new password, and a power failure occurred while the new password was being written, it is possible the DLM would become unreachable. (Run-time password changing can be disabled.)

> **NOTE:** The redundant copies in the User block are only used if the board has a System ID block version 3, or higher. A System ID block version 3 (or higher) is highly recommended. The only difference between a version 3 block and a version 2 block is the version number itself, which can be examined in the structure member `SysIDBlockType.tableVersion`. It is possible to update the System ID block using `Write_idblock.c`. This utility is available for download at:
>
> `http://www.rabbit.com/support/downloads/downloads_feat.shtml`

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 is a reset occurs just as a remote user starts to reprogram, causing the need for a retry to establish the connection.

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 compliment 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 218A “Implementing a Serial Download Manager for a 256K Byte Flash” for a one flash chip solution.

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