

EMBEDDED TINI ROM FIRMWARE

As IP networks have become more pervasive, it has become more of a necessity to network enable embedded systems. However, network protocols tend to be complicated to code and require a lengthy test cycle. The TINI[®] (Tiny InterNet Interfaces) runtime environment provides the entire software infrastructure needed to write network aware applications for Dallas microcontrollers. The runtime environment provides a full TCP/IP v4/v6 protocol stack verified for compliance to Internet standards. The network stack is driven by a multi-tasking operating system (TINI-OS). Using the runtime environment and its built-in APIs, developers can quickly write embedded applications that are network aware.

The DS80C400 incorporates an embedded ROM specifically designed for hosting the TINI runtime environment. The ROM firmware implements three major components: a Full TCP/IP v4/v6 stack with industry standard Berkeley socket interface, a preemptive task scheduler, and NetBoot functionality. The NetBoot component utilizes the TCP/IP stack, socket layer, and task scheduler to provide automatic network boot capability. NetBoot allows an application to be downloaded from the network and executed by the microcontroller.

In order to utilize the ROM firmware, the system is required to have the following hardware components:

- 64kB SRAM memory mapped¹ at address locations 000000h-00FFFFh
- Memory (SRAM or flash) to store user application code
- DS2502U-E48 1-Wire chip (to hold physical MAC address)
- External crystal or oscillator²

¹ Merged program/data memory configuration is required

² NetBoot functionality requires external clock frequency be at least 7MHz

SELECTING TINI ROM CODE EXECUTION

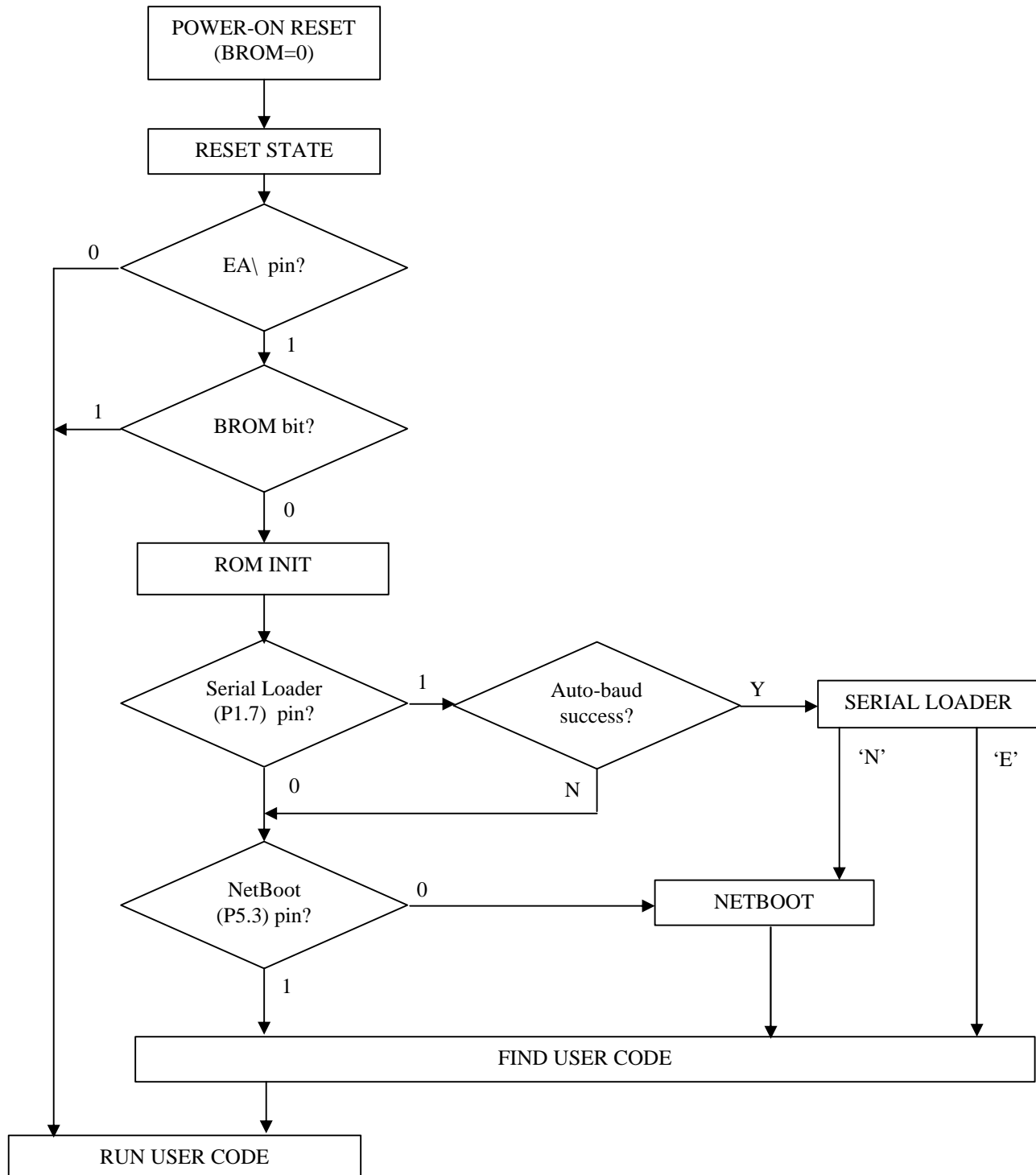
The DS80C400, following each reset, begins execution of program code at address location 000000h. Since the DS80C400 contains internal ROM and supports external program code, the user must select which of these two program memory spaces should be accessed for initial program fetching. There are two mechanisms that control selection of the internal TINI400 ROM code. These two controls are the \overline{EA} pin and the Bypass ROM (BROM) special function register bit. No matter the state of the BROM bit, if the \overline{EA} pin is held at a logic low level, the TINI400 ROM code will not be entered and will not be accessible to the user code. If the \overline{EA} pin is at a logic high level, the BROM bit is then examined to determine whether the internal TINI400 firmware should be executed or bypassed. If BROM=0, the TINI400 code will be executed. Otherwise (BROM=1), the TINI400 code will be bypassed and execution will be transferred to external user code at address 000000h. The BROM bit defaults to '0' on a power on reset, but is unaffected by other reset sources. This code selection process can be seen in Figure 13.

TINI400 ROM CODE EXECUTION FLOW

Once the internal TINI400 ROM code has been selected ($\overline{EA} = 1$, BROM = 0), it must first execute some basic configuration code (ROM Init) to provide functionality to subsequent ROM operations. Next, the ROM code reads the state of port pin P1.7. The ROM associates the logic state of P1.7 with the user desire to invoke the serial loader function. If the Serial Loader pin (P1.7) is a logic '1', the ROM monitors for activity on serial port 0 and tries to respond to the external host with its own serial banner. Once serial communication has been established at a supported baud rate, signified by correct reception of the DS80C400 loader banner and prompt, commands may be issued by the user. The Serial Loader commands are described later in the datasheet. If the Serial Loader pin is pulled to a logic '0', the ROM

reads the state of port pin P5.3. Much like the association made between P1.7 and invocation of the Serial Loader, the ROM links the logic state of P5.3 with the user desire to begin the NetBoot process. If the NetBoot pin (P5.3) is asserted (logic '0'), the ROM initiates the NetBoot process. If the NetBoot pin is not asserted (logic '1'), the ROM executes the Find User Code routine to identify executable user code. Figure 13 illustrates the ROM decisions described above.

ROM CODE BOOT SEQUENCE Figure 13

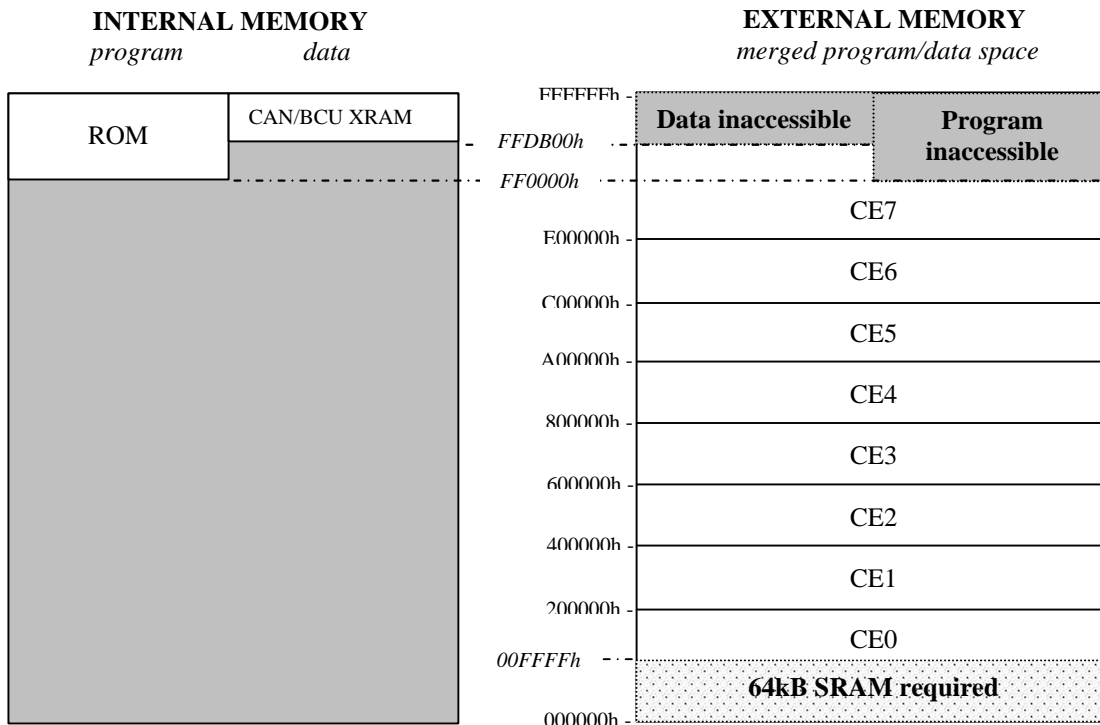


TINI400 ROM INITIALIZATION CODE

The TINI400 firmware automatically executes Initialization Code (ROM_Init) to generate the memory map as shown in Figure 14 and configure the DS80C400 hardware as follows:

Enables 24-bit contiguous address mode	(ACON.1:0 = 11b)
Logically relocates ROM to addresses FF0000h – FF7FFFh	(ACON.5 = 1)
Enables CE0-3, 2MB / chip enable	(P4CNT = 2Fh)
Enables PCE0-3	(P5CNT = 07h)
Enables CE4-7, 1M / peripheral chip enable	(P6CNT = 27h)
Merged program/data CE0-3, relocate internal XRAM	(MCON = AFh)
Enables extended 1kB stack option	(ACON.2 = 1)
Configure to maximum MOVX Stretch Value	(CKCON.2:0=111b)
Configure UARTs for Mode 1 serial operation	

MEMORY MAP FOLLOWING EXECUTION OF ROM_INIT Figure 14



SERIAL LOADER

The Serial Loader function implemented by the firmware can be invoked by leaving the Serial Loader Pin (P1.7) at a logic '1' during the boot sequence. When this condition is found, the ROM monitors the RXD0 pin for reception of the <CR> character (0Dh) at a supported baud rate. The serial loader function uses hardware serial port 0 in mode 1 (asynchronous, one start bit, eight data bits, no parity, and one stop bit in full duplex). The Serial Loader can automatically detect certain baud rates and configure itself to that speed. The equation below is used to calculate the nearest integer reload value for Timer 2 (used for serial port 0 baud rate generation) based upon the external clock frequency and desired baud rate. The calculated (nearest integer) RCAP2H,RCAP2L reload value may not result in an exact baud rate match. The calculated reload value and clock frequency can be used in the equation to solve for the baud rate configurable by the DS80C400. It is advised that the baud rate mismatch be no greater than +/-2.5% to maintain reliable communication. The functionality was designed to work for clock rates from 3.680 MHz to 75.000 MHz and baud rates from 2400 to 115200.

$$RCAP2H, RCAP2L = 65536 - \frac{\text{Oscillator Clock Frequency}}{32 * \text{Baud Rate}}$$

For example, suppose an 18MHz crystal is being used and a 19200 baud rate is desired. The above equation yields a nearest integer reload value of FFE3h. This reload value results in a true baud rate of 19396.6 (+1% error).

Once a supported baud rate has been detected, the DS80C400 will transmit an ascii text banner containing copyright information and prompt for command entry. At this point, the user may issue any of the supported Serial Loader commands. A summary of the supported Serial Loader commands can be seen in the table below. A detailed description of each command and further information pertaining to the Serial Loader can be found in the DS80C400 User Guide Supplement.

SERIAL LOADER COMMAND SUMMARY Table 18

Command	Function
B	Bank Select
C	CRC-16 of memory range
D	Dump Intel hex data from selected bank
E	Exit the loader and try to execute code
F	Fill selected bank memory with hex data
G	Go - Start executing code at offset 0 in the current bank
H, ?	Help - Display ROM version and current bank
L	Load Intel hex into memory
N	NetBoot
V	Verify memory against incoming hex
X	Execute code at a given offset in the current bank
Z	Zap - Erase/clear the current bank.

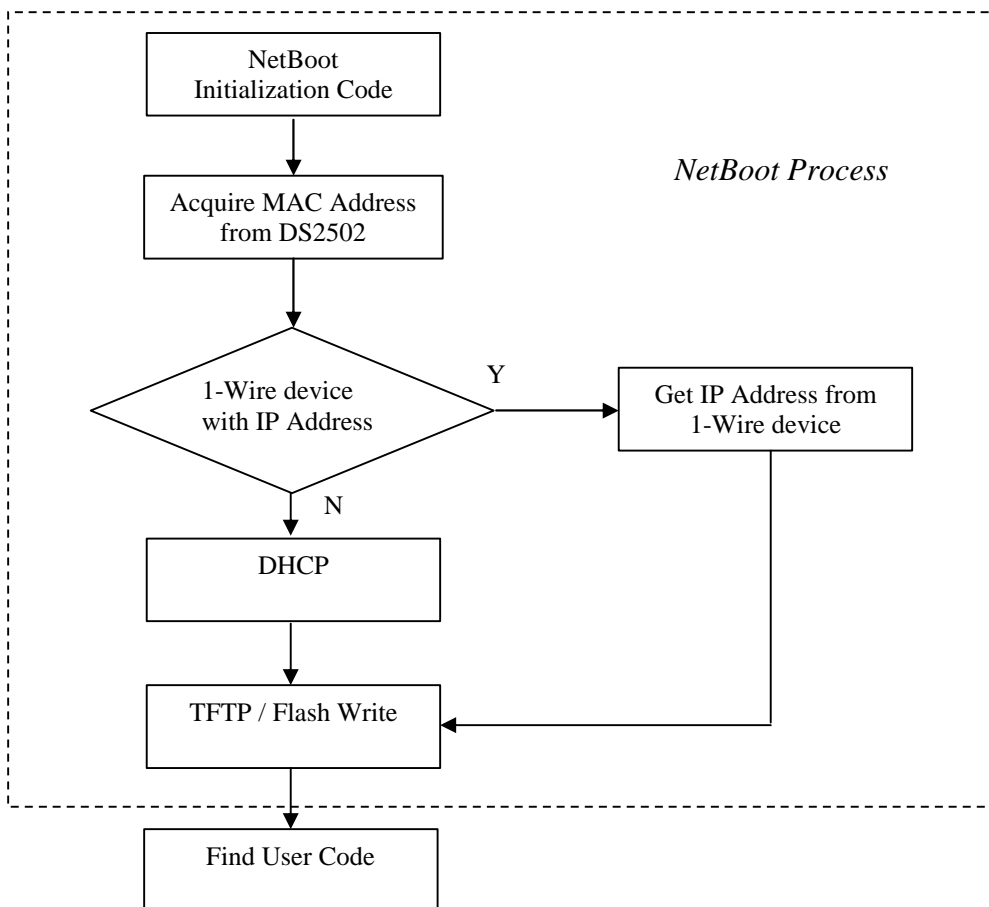
NETBOOT

The NetBoot process affords the user flexibility to download or update code remotely over the network. This capability is quite powerful. Not only does it make firmware revisions trivial, but also makes remote diagnostics very practical. Also, since NetBoot can automatically reload the latest version of the user application code, the system designer now has the option to select volatile SRAM for code storage.

In order for the NetBoot function to work, the TINI400 ROM firmware must initialize certain hardware components and create the environment needed to support the process. The NetBoot initialization code implements a primitive memory manager, kicks off the task scheduler, and initializes the 1-Wire hardware, Ethernet driver, TCP/IP stack, and socket layer.

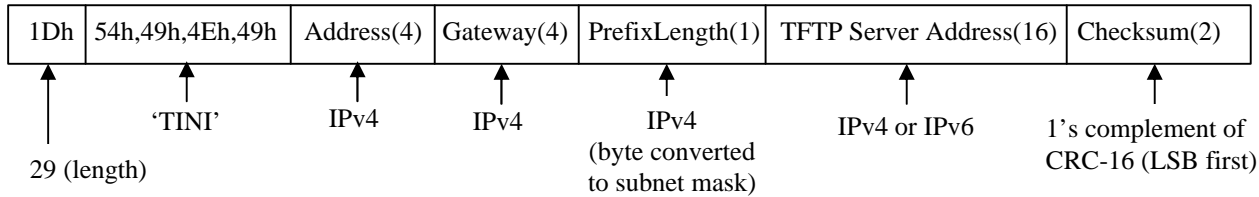
Once the NetBoot initialization code has completed, the true network boot process can begin. The DS80C400 Ethernet Media Access Controller (MAC) first must be assigned a physical address. Within the NetBoot process, the physical MAC address can only be acquired via an external DS2502U-E48 1-Wire chip. Hence, this 1-Wire chip, containing the MAC address, is required for successful NetBoot operation.

NETBOOT CODE FLOW CHART Figure 15



Next, the TINI400 ROM searches the 1-Wire bus for an external device (separate from the device containing the MAC address) that contains an IP address and TFTP server IP address. In order to correctly acquire the IP and TFTP server addresses from an external 1-Wire device, the data read from the device must conform to a specific format. This format is shown in Figure 16.

1-WIRE IP AND TFTP SERVER IP ADDRESS FORMAT Figure 16



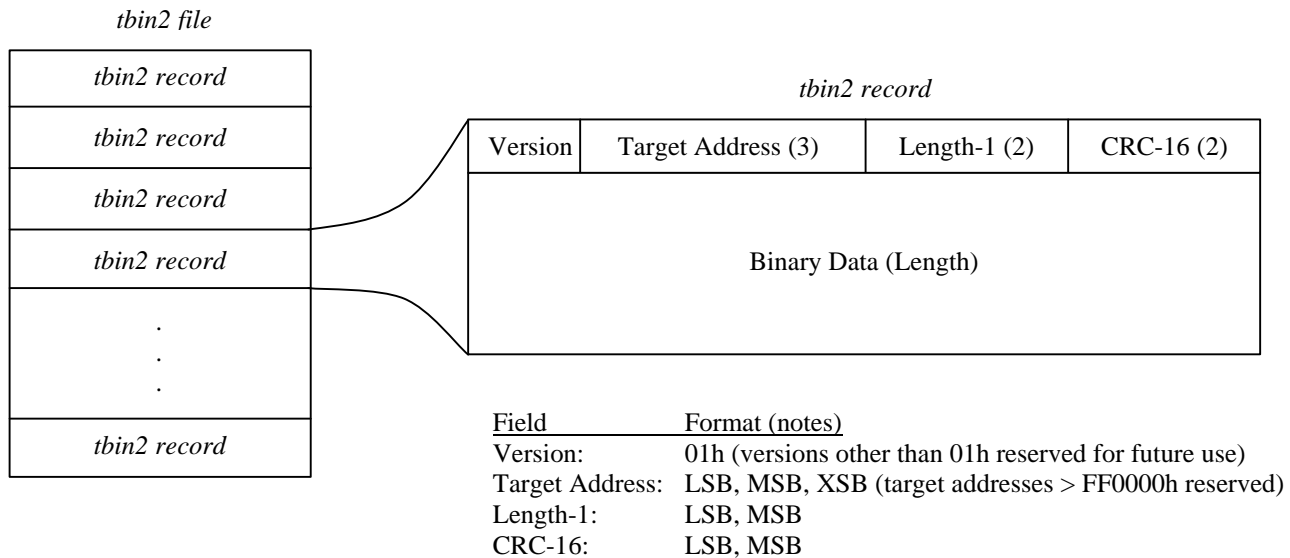
If the IP and TFTP server addresses cannot be acquired from a 1-Wire device, the NetBoot process will use DHCP to get this information. The DS80C400 will broadcast its MAC address in a DHCP Discover packet. A DHCP server, if available, should then respond with an IP address offering. The DS80C400 will subsequently request the IP address, to which the DHCP server must acknowledge. In the DHCP acknowledge packet, the TFTP server IP address is then read from the “next server IP” field. Since some DHCP servers do not allow configuration of the “next server IP” field, the DS80C400 recognizes the site-specific option 150 (also used on Cisco IP phones to get TFTP server IP addresses). When option 150 is present in the acknowledge packet, it will take precedence over the “next server IP” field.

Now armed with an IP address and TFTP server IP address, the DS80C400 tries to find code to be loaded into external program memory. The TINI400 ROM will first request to read the file from the TFTP server coinciding with its unique physical MAC address (e.g. 006035AB9811). If the request is denied, it will issue a second, less specific, request to read the filename associated with the TINI400 ROM revision (e.g. TINI400-1.0.1). If this request is denied, then it will lastly attempt to read from the TFTP server the file ‘TINI400’. Using this strategy, the TFTP server operator can distinguish between different devices and/or different releases of the TINI400 ROM firmware.

After successfully locating the desired file on the TFTP server, the DS80C400 must transfer and program the file into external memory. Currently, the DS80C400 only offers programming support for SRAM and AMD compatible flash memory devices. The NetBoot code expects the transferred file to be in the Dallas tbin2 format. The tbin2 format consists of one or more records, allowing binary concatenation of multiple images into one file. Figure 17 below illustrates the tbin2 file format.



DALLAS tbin2 RECORD AND FILE FORMAT Figure 17



For each 64kB bank to be programmed, the TINI400 ROM first performs a CRC-16 of the current memory bank contents. If the CRC-16 of the current memory matches the data to be programmed, the bank is left alone. If the CRC-16 differs, it performs a couple of write/read-back operations to assess whether the bank is flash or SRAM and then executes the erasure (if flash) and programming.

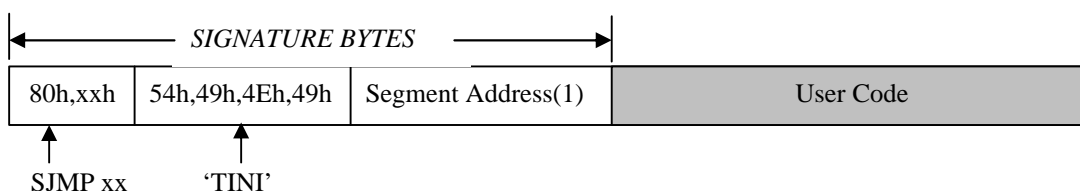
After completion of the TFTP server file transfer and programming of external memory, the NetBoot process concludes by updating a 'previous TFTP success' flag and executing the ROM Find User Code routine.

If either DHCP or the TFTP transfer fail, the NetBoot code checks whether the TFTP transfer has been successful in a previous attempt. If so, the TINI400 ROM exits NetBoot and transfers execution to the Find User Code routine. If the TFTP transfer has not been successful in the past, the TINI400 ROM allows the watchdog timer to reset the DS80C400.

FIND USER CODE

The TINI400 ROM firmware attempts to find valid user code by searching for specific signature bytes at the beginning of each 64kB block of memory. The search begins at address location C00000h and continues downward through memory in decrements of 64kB until executable code is located or failure occurs (search terminates at 000000h). In order for the Find User Code routine to judge a block of memory as valid executable code, it must be tagged with the signature bytes shown in the figure below.

USER CODE SIGNATURE (REQUIRED BY FIND USER CODE) Figure 18



Once a valid signature is found, the signature byte at offset 6, referred to in the above figure as the Segment Address, is examined to determine whether execution control should be transferred immediately or whether the search should continue. If the Segment Address byte equals 00h or matches the most significant address byte for the 64kB block being examined, execution is transferred to the user code. If the Segment Address byte does not match, that Segment Address byte is used to determine the next memory block examined for a valid signature.

EXPORTED ROM FUNCTIONS

The TINI400 ROM firmware implements many functions that are made accessible to the user application code. In order for user application code to call a specific function, the location of that function must be known. The absolute address location of each TINI400 ROM function must be read from an export table (also found in the ROM). To allow flexibility for future ROM firmware structural changes and improvements, the export table itself is not tied to a specific address range, but instead, a three byte pointer to the start of the export table is fixed at addresses FF0002h (XSB), FF0003h (MSB), and FF0004h (LSB). The first three bytes of the export table contain the quantity of function entries in the export table. In three byte increments, following the first three bytes, the rest of the table contains absolute address locations for the exported ROM functions. Thus, once the export table location has been discovered, the index for a given function/structure (found in the list above) can be used to find its absolute address (Function address = ExportTable[Index*3]) The figure below illustrates the method for locating the export table and a specific ROM function. The table below shows the contents of the ROM export table. Brief descriptions of the functionality provided by the TCP/IP stack, socket layer, and task manager are included after the table, while the full details for these and other exported ROM functions will be covered in the DS80C400 User Guide Supplement.

FINDING THE LOCATION OF AN EXPORTED ROM FUNCTION Figure 19

