

#### APPLICATION NOTE 3413

## Initializing High-Speed TINI Systems

*Before the DS80C400, TINI® systems were limited to running at a maximum of 40MHz, restricted by the maximum speed of the DS80C390 processor. With the promise of 75MHz on the DS80C400, support was added to the TINI firmware to allow systems to run with the fastest crystal multiplier allowed. To make the system affordable, however, the TINI reference design (TINI<sub>m</sub>400 and TINI<sub>s</sub>400) was not designed to run at these high speeds. Many developers tried to run their TINIs at the x4 crystal multiplier setting, and each finally noted that their TINIs simply stopped. The reason for this is not that the DS80C400 processor cannot support that speed, but that the flash cannot support that speed. Consequently, a high-speed TINI system requires a custom board with RAM for firmware storage and execution. This application note will discuss initializing such a high-speed TINI system.*

### Flash Limitations

Flash is generally the limiting factor in high-speed TINI designs. The best commercially available flashes have an access time of 55ns. The following chart shows how that relates to DS80C400 CPU speed and the AM29LV081 flash<sup>1</sup>:

Rated Flash Speed	Max CPU Speed
55ns	40MHz
70ns	33MHz
90ns	27.5MHz
120ns	21MHz
150ns	17MHz

The data presented here leave the developer with a choice—run out of the flash and be limited to 40MHz execution speed, or execute out of the RAM and run up to 75MHz. To achieve a high-speed design that uses the DS80C400 and the TINI OS, the normal TINI memory map (as on the TINI<sub>m</sub>400) must be altered.

### TINI OS Memory Requirements

To run the TINI Java™ Runtime (the TINI OS) on the DS80C400, there must be at least 512K RAM on CE0 (address 0) and another 512K of code space on CE2 (address 400000h). On the TINI<sub>m</sub>400, the code space is implemented as flash, but it could just as easily be implemented as RAM.

The problem with RAM for application code is that it is initially blank, and is erased if the power is removed, a battery dies, or some other external event wipes its contents. TINI applications will typically be used for remote monitoring and control, making field updates very inconvenient if a device loses its code. The solution is to copy an image of the application into the RAM from a flash that lies outside the normal TINI memory map.

### An Alternate Memory Map

Consider the following memory map for a high-speed TINI.

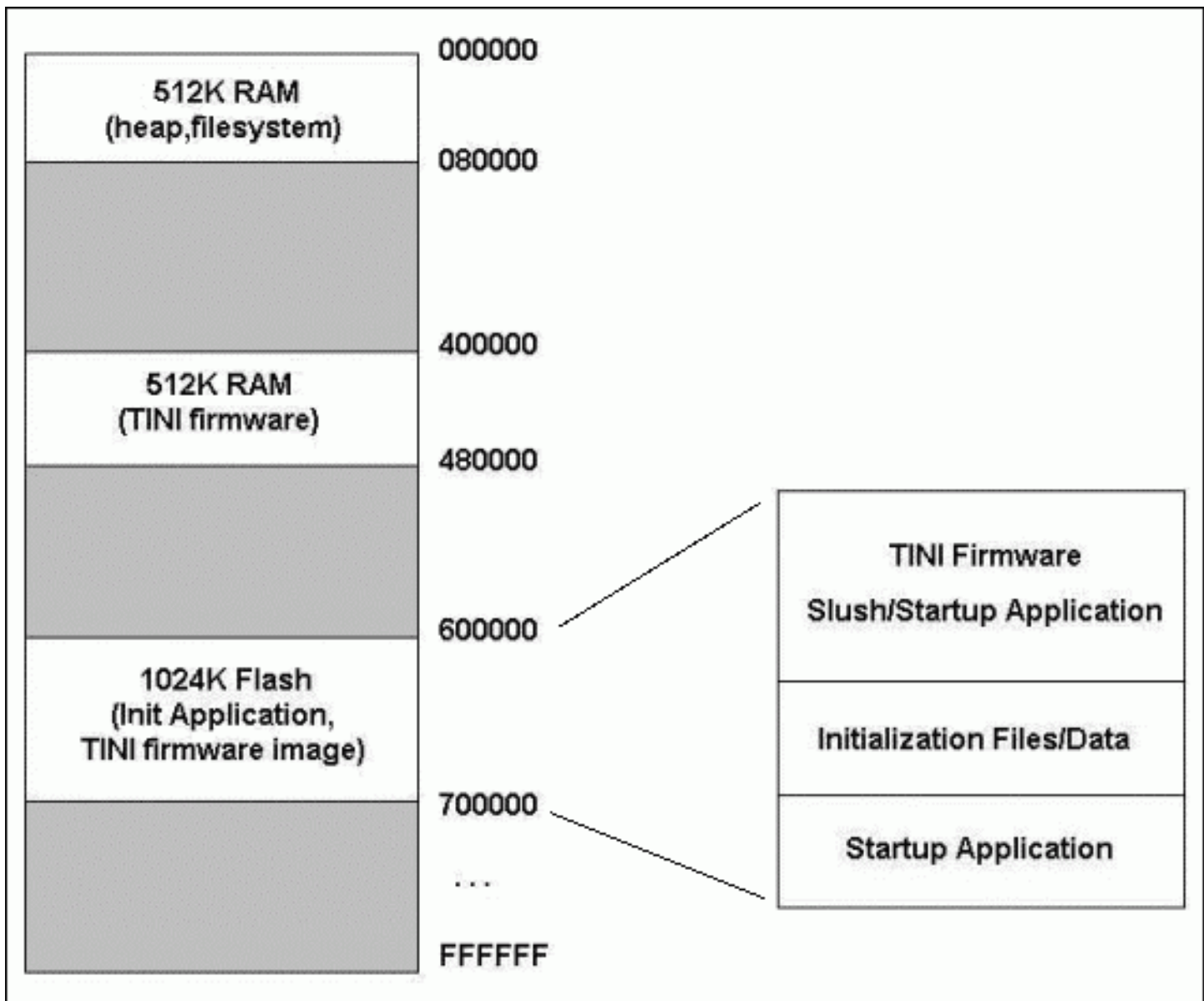


Figure 1. Possible memory map for a high-speed TINI system.

This memory map shows two 512K RAMs at the locations required by the TINI OS (one at address 0 and one at address 400000h). It also contains an additional flash at address 600000h. This flash contains an initialization application that is responsible for:

- Initializing the heap
- Copying the firmware and application image to address 400000h
- Transferring control to the TINIOS

When the DS80C400 boot loader comes up, it starts from the top of memory and searches for an executable to run. When it finds a valid TINI application signature, it transfers control to that address. We will place our startup code at address 6F0000h, ensuring that it is the first application code found. A valid TINI signature consists of the following structure at address 0 of a 64K memory bank:

SJMP statement	String 'TINI'	Bank Number (or 0)
2 bytes	4 bytes	1 byte

See the section "Find-User Code" in the DS80C400 data sheet<sup>2</sup> for more information on this process.

## Making the Firmware Run Faster

Now that we have our custom TINI hardware designed, we need to alter the firmware to run at a faster speed.

The TINI firmware shipped with the TINI Software Development Kit is pre-programmed to run at a crystal multiplier of 2 on a 14MHz crystal. High-speed TINI systems typically run at a multiplier of 2 or 4 on a 14MHz or 18MHz crystal. The TINI firmware can be altered for different crystal speeds and multipliers using the [TBINFixer tool](#). Run this application with no arguments to see the command line options. Typically, only the crystal multiplier (-m option) and the crystal value (-c option) will need to be changed. Use the file `tini400.tbin` from the TINI SDK as the input file for this application.

## Startup Application

With the firmware altered to run at a faster crystal multiplier, we can now turn our attention to the program of initializing our TINI system to execute out of the RAM. A typical startup application presented here is written in assembly and built with free tools (macro and a390) that come with the [TINI Software Development Kit](#).

For this startup application, we chose to store our TINI firmware and Slush application as TBIN<sup>3</sup> files in the flash. As a result, our startup application will need to handle the parsing of the TBIN file format. The benefit of this is that the TBIN format stores all the address information for its data, making the startup application somewhat more general purpose.

The file `init.a514` shows an initialization application that reads some TBIN files stored in the flash and copies them to the RAM, then transfers execution to the RAM. The code is general-purpose, although three locations might require some adjustment for different memory configurations:

- `TBIN_GLOB_ADDRESS equ 600000h`  
This is the address where the TBIN files are stored. The image that resides here should be built using the BuildTBIN application (see the next section).
- `EXECUTE_ADDRESS equ 400000h`  
For the TINI Java Runtime, this address should stay 400000h (it can change for applications written in C). After the initialization application is complete, an LJMP to this location will be executed to transfer application control.
- `org 6F0000h`  
The org statement for the application specifies the location of the code.

With the default configuration of the initialization application, the TBIN files stored at address 600000h will be copied into the RAM (at the addresses specified in the TBIN files themselves). After the copy, program execution will transfer to address 400000h.

Developers should consider a few issues in the current init code:

- **Serial Debug:** The initialization application configures timer 2 to generate baud rates for serial port 0. (The application contains some equates for tuning the baud rate.) It outputs information on each TBIN record that is being copied to the RAM. All the serial routines can be removed without affecting the functionality of the init code.
- **Unrolled copy:** The function Copy256 contains a partially unrolled 256-byte copy (unrolled as 16 loops of 16 single-byte copies). This is good for performance, but increases code size. If code size is more important than startup delay, this is a good place to look first.
- **POR Checking:** The init code does not check for a POR before it copies the TINI firmware and application. However, this may be desirable if the TINI application intentionally resets without cycling power. In this case, the application could skip the copy and go straight to code execution. Note that it is advisable to distinguish between intentional and unintentional watchdog resets. An unintentional watchdog reset may mean an improperly functioning TINI, in which case reloading the application code worth considering.
- **RAM Erase:** The startup code also performs an erase of the first 64 kilobytes of RAM to make sure the system has a clean start. This action may be removed if the RAM at CEO is battery-backed. This startup code is

included because on power-up many TINI systems assume that the file system needs to be reinitialized.

The macro (macro preprocessor) and a390 (assembler) were used to build the startup code, using the following commands:

```
macro init.a51
a390 -l -Ftbin -d -p 400 init.mpp
```

The resulting file is called `init.tbin` and should be loaded onto a TINI system along with the output of the BuildTBIN application, detailed in the next section.

## The BuildTBIN Application

The BuildTBIN application is a Java application that takes one or more TBIN files and formats them into one file, usable by the startup application presented in the prior section. As our high-speed application will copy Slush and the TINI firmware from address 600000h to its proper location, the command line for BuildTBIN looks like this:

```
java BuildTBIN 600000h file_to_load.tbin slush400.tbin tini400.tbin
```

In this case, the files `slush400.tbin` and `tini400.tbin` will be combined to make the file `file_to_load.tbin`, which will be targeted for address 600000h. The file `file_to_load.tbin` should be loaded onto the TINI system, along with the file `init.tbin`.

The BuildTBIN application takes the input TBIN images and treats them as binary data, rewrapping it into another TBIN file targeted for a different address. There is a complication: the startup application needs to know how many TBIN records are included. This is not a problem with loader programs such as JavaKit and MTK, which look for the EOF (end-of-file) to determine when there are no more TBIN records. To solve this problem, the output TBIN file includes as its first byte a count of the number of TBIN records in the file. Because a TBIN file can include more than one TBIN record, the BuildTBIN application must parse each input TBIN file to count how many records it includes.

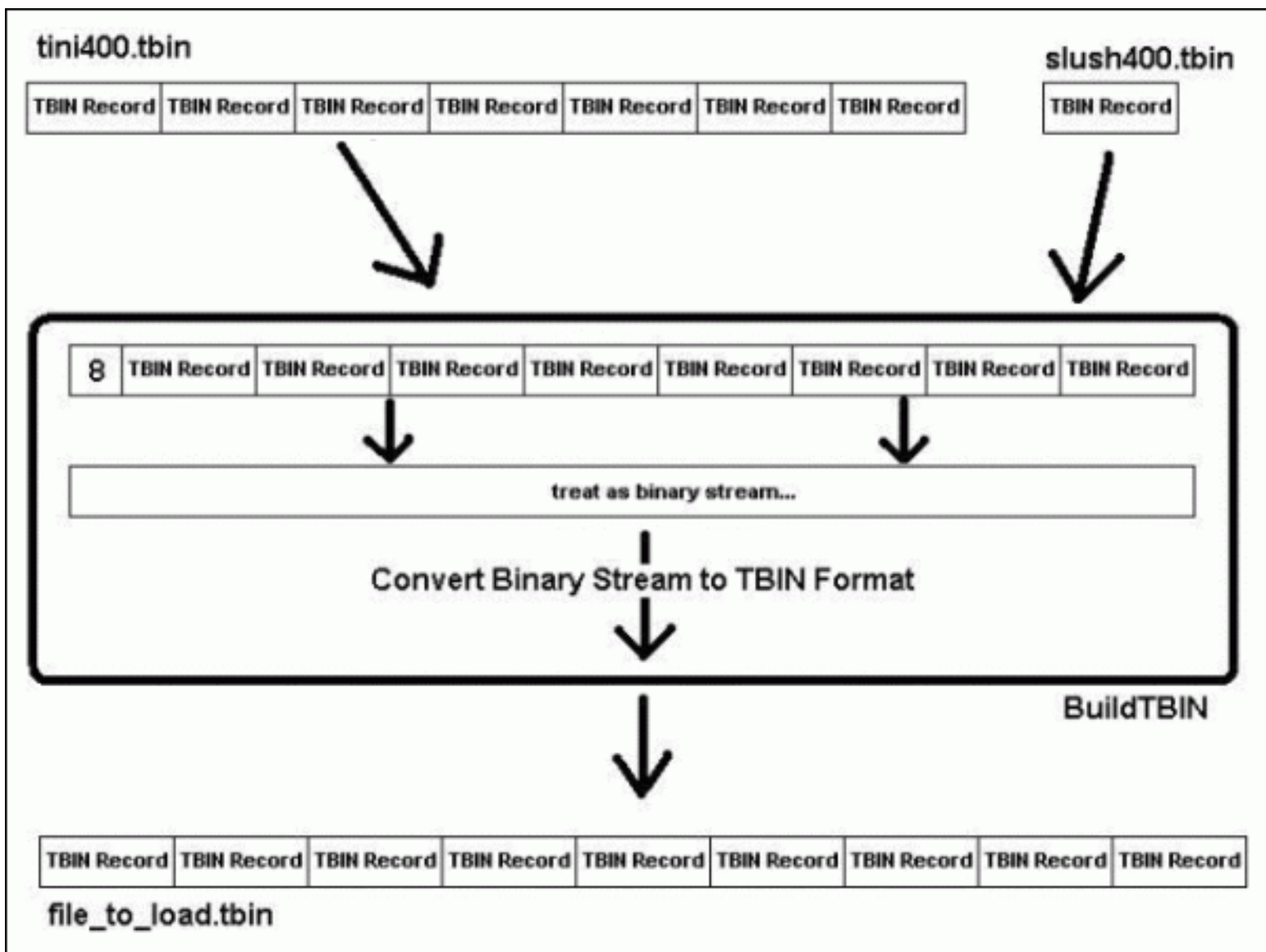


Figure 2. The BuildTBIN program takes input TBIN files, adds a 'count' byte, treats the whole thing as a binary stream, and outputs a new TBIN formatted file. As each TBIN record contains some overhead, note that the output file may contain up to 9 TBIN records, although the input files contained only 8.

The TBIN file format is described at: [http://files.dalsemi.com/tini/ds80c400/c\\_libraries/tbinformat.html](http://files.dalsemi.com/tini/ds80c400/c_libraries/tbinformat.html). The source for the BuildTBIN tool is available at <http://files.dalsemi.com/tini/ds80c400/tools/buildtbin>.

If you would also like to include non-TBIN data in your final TBIN image, you can translate any binary data to the TBIN format using the tool Bin2TBIN. The source for this tool is available at: <http://files.dalsemi.com/tini/ds80c400/tools/bin2tbin>.

## Conclusion

The approach presented here for initializing high-speed TINI systems is one of many possible solutions. This article and the code accompanying it are meant both as a solution and as an education on solving the problem. By adding a little more RAM and flash, a stable, high-speed TINI can be realized to take full advantage of the abilities of the DS80C400 processor.

## Relevant Links

- [TINI Software Development Kit](#)

- High Speed User's Guide Supplement: <http://files.dalsemi.com/tini/ds80c400/tools/DS80C400>
- [DS80C400 Data Sheet](#)
- [Discussion Forum](#)
- Source code for the application note: [http://files.dalsemi.com/tini/appnotes/highspeed/highspeed\\_initcode.zip](http://files.dalsemi.com/tini/appnotes/highspeed/highspeed_initcode.zip)

### Notes

<sup>1</sup> An application note titled "Flash Memory Selection" is currently in development for information on compatible flash devices

<sup>2</sup>The [DS80C400 Data Sheet](#)

<sup>3</sup>Information on the TBIN format can be found at [http://files.dalsemi.com/tini/ds80c400/c\\_libraries/tbinformat.html](http://files.dalsemi.com/tini/ds80c400/c_libraries/tbinformat.html)

<sup>4</sup>The source code for this application note is available at [http://files.dalsemi.com/tini/appnotes/highspeed/highspeed\\_initcode.zip](http://files.dalsemi.com/tini/appnotes/highspeed/highspeed_initcode.zip)

---

Application Note 3413: <http://www.maxim-ic.com/an3413>

### More Information

For technical questions and support: <http://www.maxim-ic.com/support>

For samples: <http://www.maxim-ic.com/samples>

Other questions and comments: <http://www.maxim-ic.com/contact>

### Related Parts

DS80C400: [QuickView](#) -- [Full \(PDF\) Data Sheet](#) -- [Free Samples](#)

AN3413, AN 3413, APP3413, Appnote3413, Appnote 3413

Copyright © 2005 by Maxim Integrated Products

Additional legal notices: <http://www.maxim-ic.com/legal>