The TINI Platform

A.J. Wetherilt[†][*]

[†] *Artesis AS*
34950 Tuzla, Istanbul, Turkey

Lecture given at the:
Second Workshop on
Distributed Laboratory Instrumentation Systems
Trieste, 20 October — 14 November 2003

LNS

[*]anthony.james.wetherilt@arcelik.com

Abstract

The TINI hardware platform is described together with its operating system, Java functionality, and programming usage

Keywords: Linux TINI, Java, TINIOS, 1-wire
PACS numbers: 64.60.Ak, 64.60.Cn, 64.60.Ht, 05.40.+j

Contents

1 Introduction

The current workshop is devoted to topics involving the design, implementation and use of distributed control and information gathering systems, primarily in the laboratory but of relevance to many other areas and disciplines. As it's name suggests, the study of Distributed systems concerns the practice of using local devices for data acquisition and control of processes in the laboratory or elsewhere. They are distributed in the sense that they are local to the application and generally have the ability to communicate with other devices over a network of some description. Each device on the network performs a separate function, and the functionality of the network as a whole is dependent on the individual devices. These devices can have varying degrees of intelligence depending on the precise application, available hardware, and cost considerations. As hardware becomes less and less expensive (or more and more power becomes available for the same unit cost), it becomes much more efficient to have small, dedicated and cheap units performing data acquisition and control functions rather than a large and complex central system responsible for many devices. Such dedicated systems are not only relatively easy to develop but they also allow addition, removal or modification of individual devices in a much easier fashion than a monolithic system. Thus in the same way that distributed workstations have virtually replaced the previous generation of super computers, distributed process control systems are rapidly becoming the norm in industrial and research environments.

Experience has shown that development time and costs associated with it are often the most crucial elements in the development of any system with the actual cost of hardware being insignificant in comparison. The single most important contribution to the development costs, without exception is that of the process of software development and testing. It is therefore essential to choose a development system that ensures short programme development cycles in a number of ways. Firstly, the platform chosen for the development (not necessarily the final target platform) must have available good development tools (compilers, debuggers, simulators, profilers etc) and be fast enough (if only to prevent frustration in the programmer). Secondly, the programming language selected for development must allow the programmer the necessary functions to write code quickly, to reuse code, and most importantly, to prevent the programmer making the numerous mistakes that human beings are capable of and that are often very difficult to detect. The classic example here is of the so called 'Memory leak', where a block of memory is allocated dynamically and never released. By itself this may not be so serious, but when the code causing the memory leak is repeated, the amount of memory allocated grows steadily and can eventually cause device failure. C is particularly notorious for this behaviour as it has no built in methods for either detection or prevention. C++ is not much better in this respect and eradication of memory leaks can literally take weeks to months of the life cycle of a project. Thirdly, since the pace of hardware development is dramatic, the choice of hardware can be expected to be redundant after only a few years. This means that any code written must easily be portable to other processors without lengthy rewriting of drivers, or other sections of code.

With the above considerations in mind, Java is an excellent choice of programming language: It is highly portable between platforms, has excellent development environments, allows code classes to be reused and protects the programmer from making many common errors. Java has been declared the language of the Internet and indeed has found many applications in that field. It is also highly suited for the task of programming embedded or small micro-controller based systems. The Tiny Internet Interface or TINI is a complete Java based platform built for the Dallas Semiconductor DS80C390 micro-controller. The TINI evaluation kit is relatively cheap and comes complete with built in Ethernet, CAN and other serial communications interfaces and sufficient RAM and flash memory to hold a practically complete Java virtual machine and run-time environment. It is therefore ideal for the task in hand which is a platform for the development of networked, localised, intelligent systems for process control which communicate with each other over the Ethernet using TCP/IP, and will be used during the course of the practical sessions of this workshop. These notes then outline the features of the TINI platform, including its hardware, programming characteristics and operating system API, and give an overview in sufficient depth to facilitate its use in the practical sessions. For a more complete coverage, the reader is referred to the TINI Specification and User's Guide available in PDF form from the TINI website.

2 TINI hardware

2.1 Overview

The TINI development board (TBM390) is a small board measuring 31.8 x 102.9mm that is designed fit into a 72 pin SIMM socket. The TBM390 carries the processor (DS80C390), with 1M of SRAM in two 512k chips (made non-volatile by the provision of a lithium battery), a flash RAM, an Ethernet controller chip and a host of interface drivers, and support chips listed in table 1. All connections to the outside world are made via the 144 pins of the SIMM connector along the lower edge of the board. In use, the TBM390 must be attached either to a bus system that provides the necessary power and signals or to a development socket such as the TINI Socket E manufactured by Dallas Semiconductors. This socket has the Eurocard form factor (100 x 160 mm) and includes the sockets listed in table 2. Space for the following extra functions is provided on the board:

- An extra 512k flash RAM
- 16 parallel input and 16 output lines
- Support for a 16550 compatible UART
- A CAN bus interface with optional end-line terminator

Other manufacturers who produce TINI compatible socket boards are listed in table 3 .

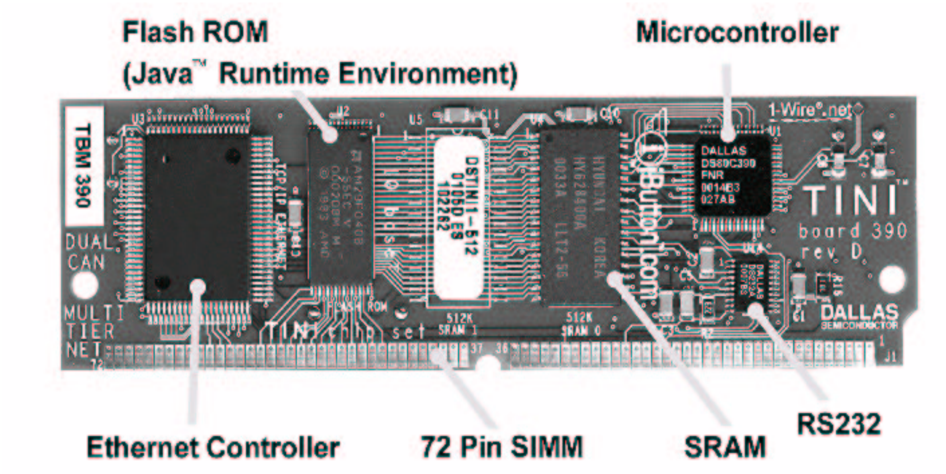

Figure 1: The TBM390 card (1)

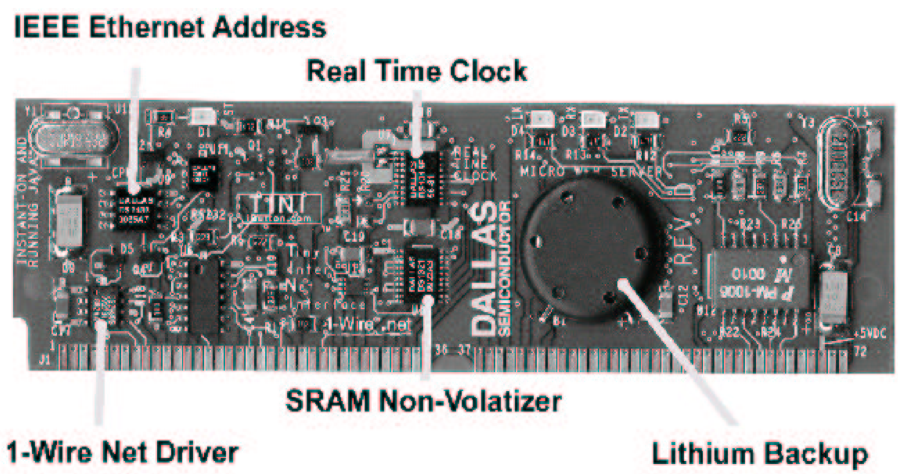

Figure 2: The TBM390 card (2)

Part no.	**Description**	**Component**	**Data sheet**
U1	CPU	DS80C390	www.dalsemi.com/datasheets/pdfs/80c390.pdf
U2	Flash RAM	AM29f040B	www.amd.com
U3	Ethernet interface	LAN91C96	www.smsc.com
U4	SRAM	HY628400A	http://www.ineltek.ru/pdf/Hynix/sram/Low_Power/4M-bit/HY628400A.pdf
U5	Real-Time clock	DS1315E	http://www.chipdocs.com/pndecoder/datasheets/DALAS/DS1315E-5.html
U6	1-Wire interface	DS2480B	http://pdfserv.maxim-ic.com/en/ds/DS2480B.pdf
U7	SRAM non-volatiser	DS1321	http://pdfserv.maxim-ic.com/en/ds/DS1321.pdf
U8	Ethernet address	DS2502-UNW	pdfserv.maxim-ic.com/arpdf/DS2502-UNW-DS2506S-UNW.pdf
U9	Ethernet filter	PM-1006	www.premiermag.com
U10	RS232 interface	DS232A	pdfserv.maxim-ic.com/en/ds/DS232A.pdf

Table 1: Major components of the TBM390 board

Part no.	**Description**	**Component**
J1	DTR Reset Enable	2 pin header
J2	1-Wire net	RJ11
J3	RS232 DCE serial port	DB9 (male)
J6	RS232 DTE serial port	DB9 (female)
J7	Ethernet port	RJ45
J8	Coaxial power connector	
J12	TINI socket	SIMM72

Table 2: Connectors on the TINI E socket

Supplier	Email	Description
Systronix	sales@systronix.com	A wide range of socket boards
Vinculum	sales@vinculum.com	Development sockets and expansion boards

Table 3: Alternative TINI socket boards

2.2 The DS80C390 micro-controller

At the heart of the TINI hardware is the DS80C390 micro-controller. This device behaves like an extended 8051 operating at an equivalent clock speed of 120MHz using a 40MHz crystal. The processor is capable of addressing up to 4Mbytes of external memory using 22 bit addresses, in addition to 4kbytes of internal SRAM. High speed 32 bit arithmetic operations are performed in a dedicated maths accelerator with a 40 bit wide accumulator.

The DS80C390 has the following internal hardware ports:

- 3 timer/counters,
- 2 full duplex 80C52 compatible serial ports
- 4 8-bit I/O ports
- Programmable watchdog timer
- Programmable IrDA clock
- 2 full-function CAN ports

The chip is available in two packages: a 64 pin QFP and a 68 pin PLCC. The former is used on the TINI TBM390 board.

2.3 Memory map

The memory map for the TINI system is shown in figure 3 and has two components: the Address space and the Peripheral Chip Enable (PCE) space, each of 4Mbytes size. On the TBM390 board the PCE space is unused as access to devices in this space is slower and less efficient than to the Address space. The Address space is subdivided into three main areas known as the Code, Data and Peripheral segments mapped between the fixed addresses 0-0x0fffff, 0x100000 - 0x2fffff and 0x300000 - 0x3fffff respectively. The Code segment needs a minimum of 512kbytes and a maximum of 1Mbytes of flash RAM and contains the runtime environment consisting of the TINIOS and Java API. Similarly the Data segment requires a minimum of 512kbytes and a maximum of 2Mbytes.A part of this area is used by the system to store any data it needs to write, but the majority is set aside for use as a file system and heap.Finally, the TBM390 maps all peripheral devices into the Peripheral segment. The Ethernet controller is found at 0x300000 - 0x307fff and the RT Clock at 0x310000. If required, other peripherals can be added to socket boards at higher addresses up to the end of this segment, provided care is taken not to overload the driving capacity of the

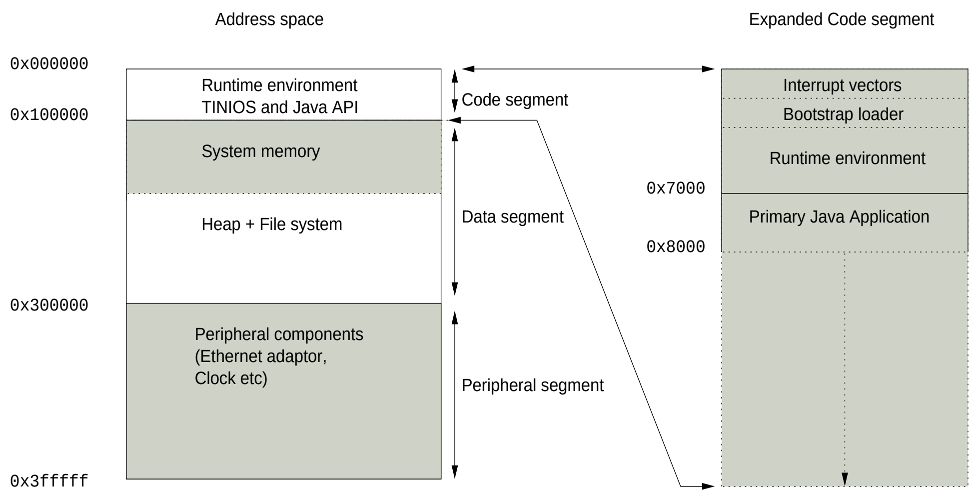

Figure 3: The TINI memory map

bus. The devices internal to the processor are not mapped to either the Address or PCE spaces but use special, internal registers and memory.

3 Tools and utility software

3.1 The TINI Runtime Environment

The feature that makes the TINI hardware really useful as a platform is perhaps the firmware provided by the manufacturer. The firmware comes as a set of software tools and utilities that allow applications written in Java to be compiled, downloaded and run on TINI. The TINI Runtime Environment provides a flash resident environment where previously compiled Java binary code can execute. At the top of the Runtime Environment, a Java Virtual Machine (JVM) executes the Java code which makes calls to the Java Applications Programming Interface (API) as shown in figure 4. The API includes reasonably complete (within the limitations of the platform) implementations for most classes in the most commonly used Java core packages given in the following table 4. Differences in the functionality of the classes implemented in these packages and the JDK1.1.8 API do exist and can be found in the file “APIDiffs.txt” included with the TINI SDK documentation. Although every effort is being made to standardise the packages, it is neither expected nor is it desirable to include the entire functionality of the official Java SDK in the same way that a small embedded controller with limited resources will never be able to perform in the same manner as a fully fledged workstation. However, most of the functionality that is relevant to the platform is expected to be included.

Package	Contents
java.lang	Classes that are fundamental to the Java language
java.io	Provides for system input and output through data streams, serialization and the file system
java.net	Provides classes for implementing networking applications.
java.util	Collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-numbergenerator, and a bit array).

Table 4: Core Java packages implemented in the TINI API

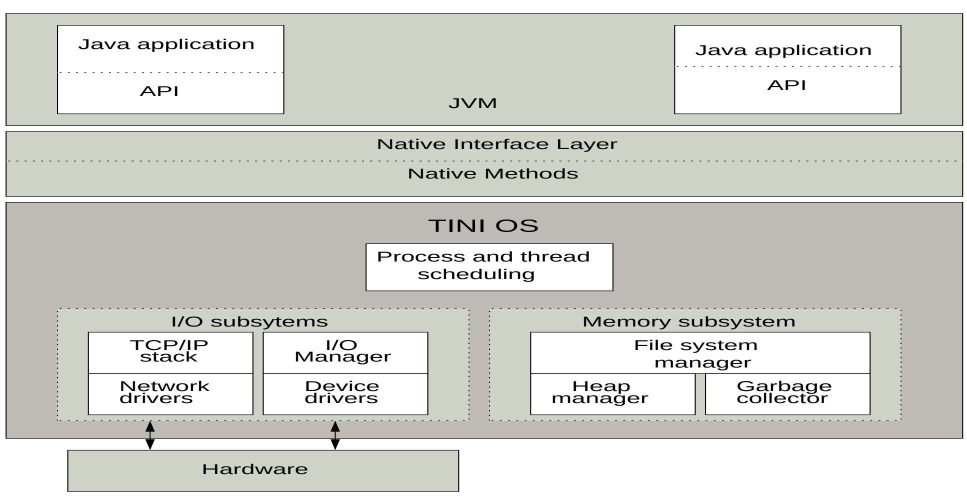

Figure 4: The TINI memory map

3.2 The TINI Operating System

The bottom layer of the Runtime Environment is the TINIOS or TINI Operating System. This is a structured set of software routines that performs the many and varied functions that are needed for managing hardware input and output, memory management and process scheduling.

TINIOS is a multitasking system, and as such needs special functionality to create and destroy processes. A process is a unit of code that and can generally be thought of as a running programme. If a system can run but a single process, all the operating system needs do is to start running the process and tidy up after the process terminates. When a system can multi-task by running two or more such processes, some mechanism must be provided to switch between or *schedule* the processes. Many different types of scheduling exist depending on how they function. One of the simplest, known as *round-robin scheduling* is to put all processes in a circular buffer and let them run for a set period of time known as the *time slice*. Other methods allocate each process a priority, and allow only the highest priority process to run until it either completes or blocks waiting for some system resource to become available. This would be the situation in a Real-Time system where system response time is critical. Variations on round-robin scheduling on the other hand are generally used in non-time critical systems such as Linux and Windows. An essential item of both systems is the system clock for different reasons. Under TINI, processes are scheduled using standard round-robin scheduling with a time slice of 8ms and a system clock that generates an interruption once every millisecond. Within a process, separate *threads* or *lightweight processes* can run in a similar manner. Threads have the advantage that since they share the same code and variable set, they are relatively straightforward and inexpensive in terms of time and code to schedule in comparison with processes. TINIOS also uses round-robin scheduling for threads with a time slice of 2ms. The number of processes is limited to 16 by TINIOS which in practice is a sufficient quantity for most device applications.

At the lowest level of TINIOS, the hardware devices (both internal and external to the DS80C390 micro-controller) are interfaced to a set of device drivers which control the flow of data and information into the TINI system. As usual with device drivers, their function is to manage bidirectional data into a format that can be handled at a higher level in a *device independent* manner in the I/O manager layer, and in the lower level hardware. The drivers support the standard serial ports as well as devices as diverse as the CAN and 1-wire systems. There are no built in drivers to handle any devices that may be attached to the expansion bus as the devices can be arbitrary.

Handled separately from the other I/O devices is the the TCP/IP stack as this imposes rather specialised requirements on the operating system. Indeed, the TCP/IP stack is reported to be the one of the largest blocks of code in the whole runtime environment. Multiple network interfaces such as Ethernet and Point to Point Protocol (PPP) are supported by the stack, using the Ethernet and serial port device drivers respectively.

The Memory management subsystem is responsible for the two main tasks in the Data segment: managing the file system; and memory allocation and deallocation. Java applications can access files through various classes in the java.io package. TINIOS therefore provides a file system to allow the storage and retrieval of data through the familiar concept of a file system, in which files are manipulated within a directory structure. On those systems equipped with RAM non-volatiser circuitry, the file system, much as on a hard disk, is not lost when power is cut, and data is retained across booting sessions. It should be remembered that when there is limited memory available, a large file system inevitably has consequences on the size of applications that can be run and vice versa.

When Java allocates an object using the *new* operator, the memory needed for that object has to be supplied from the heap. As in any other walk of life, improperly managed resources can lead to shortages and delays in issuing those resources. In an environment where objects are continually being created (and destroyed) proper memory management is essential to the continuing operation of the system. Under Java unlike C and C++, memory is not implicitly reclaimed and an object known as a *garbage collector* is employed to examine blocks of memory and determine whether or not they are still being used. If not, the block of memory are released back to the heap for reuse by a future object. Under TINIOS, the garbage collector is implemented as a process that is created during the boot process, and runs under certain well defined conditions:

- A *new* operation causes the available memory to fall below a threshold of 64kbytes
- The garbage collector is called directly by use of the gc process in `java.lang.System`
- A Java process exits and implicitly invokes the garbage collector

When the garbage collector runs through invocation by one of the above, it examines the memory allocation of only the process that called it either directly or indirectly, and frees any allocated memory no longer in use. It does not free memory belonging to any other process.

4 The System boot sequence

When power is applied to a system, its micro-controller undergoes a series of actions that load a portion of code found in non-volatile memory. This code can be the main application that is to be run or more often, especially in more complicated systems, it is a small portion of code that helps initialise and load larger programmes which in turn load other programmes and so on. This process is called bootstrapping. The DS80C390 used in TINI boots using the code found at the very beginning of the flash RAM in the Bootstrap Loader following the interrupt vectors. (figure 3). On booting, the processor starts by loading the Reset Vector at address 0x0000 containing the starting address of the bootstrap loader. This action causes control to jump to that location and execute the code

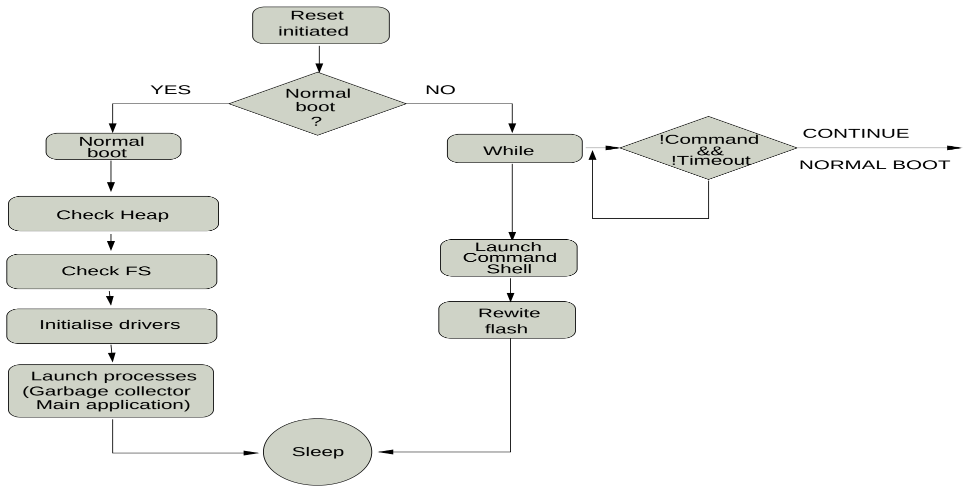

Figure 5: The bootstrap sequence

found there. The bootstrap loader can perform two separate actions depending on whether the reset was triggered by power being applied or whether an external reset was issued by bringing the RST low (pin 2 on the QFP version used with TINI). It distinguishes between these actions (and indeed whether the system was reset by a watchdog timeout) by which bits are set in one of the internal special function registers (SFRs). If a power on reset has occurred after the power has been applied and reached a minimum stable level, the boot-loader transfers control the the runtime environment. Alternatively, if the reset was caused by an external reset, the bootstrap loader initialises the Serial 0 port and waits for a pre-set sequence of bits to arrive. On receipt of this signal, a small command interpreter shell is run that allows a user to reload the flash RAM by downloading from a host machine over the serial port. In this way, the contents of the flash can be changed when for example a new application needs to be installed or the run time environment needs to be updated. If the correct bit sequence does not arrive within 3 seconds, control reverts back to the normal boot procedure. The bootstrap sequence is summarised in figure (5)

4.1 Initialisation of the Runtime Environment

The normal initialisation sequence consists of four steps:

- First the heap is checked to determine whether there are any inconsistencies. Since the TBM390 includes an on board battery for the RAM, the heap will remain in the same state as it was when last running. Certain actions such as unexpected power interruption can leave the heap in a state of disarray, and it is important to remove these inconsistencies before any process tries to use any objects saved on the heap. If the heap is found to be damaged, it is simply erased and all information lost. In any case, the heap can be expected to contain unused and unwanted objects following termination of applications by loss of power. These objects are removed and their memory reclaimed.

- Next, the file system is checked for internal consistency and any irregularities dealt with.
- The device drivers for the main hardware (Serial, Ethernet ports etc). Other devices such as the CAN and 1-wire are not initialised at this stage, but on an as needed basis by a running application.
- Finally, the processes to be run by the system are initialised and launched. In most cases, first the garbage collector process and then the main Java application are created at this time. The garbage collector has nothing to do at this stage as no other processes are running and the heap is clean, so it immediately sleeps and waits to be woken when one of the conditions outlined previously are met. Thus the main application, once started is the only active process and has full use of the available processor time. It must first initialise the Java Virtual Machine and the various classes of the TINI API. Once these actions have been performed, the primordial thread is started with execution being transferred to the main() function of the application. It is worth noting at this stage, that unlike more conventional systems where an application is typically run with limited privileges (i.e. it cannot access ports directly but must make use of services provided by the operating system), TINI gives an application full control of all resources. Indeed, in order to use some of the hardware resources, an application must first configure and initialise them. It is the responsibility of the main application to launch any other processes required to provide the desired functionality. These may be processes developed specially for certain tasks or general in nature. A specific example to be covered in greater detail later, is the launching of **slush** a small command interpreter provided as a utility function with the TINI SDK. This can be a very useful tool when developing software.

5 Using TINI

When developing applications for TINI, a host system must be attached to the TINI board to display both user input and responses from TINI. The host can be any system that runs the Windows, Linux or Solaris operating systems (and possibly other systems that supports Java and have ports for the software items listed later in this section). The host must provide an Ethernet adapter for communications with the board, and if the runtime environment is to be installed or changed, a standard RS232 port is needed. These conditions are met by virtually any contemporary PC or workstation. For our purposes, we will assume the host to be a PC running the Linux operating system. The host must also have the following items of software installed and accessible:

- Java Development Environment
- Java Communications API

- TINI Software Development Kit

Java development environments are available from a variety of both free and commercial sources. The Java Communications API is supplied by Sun Microsystems. For Linux (and a rapidly increasing selection of other platforms) an Open Source version is available from RXTX ([4]).

The latest version of the TINI SDK is available by downloading from from Dallas Semiconductor ([3]). It contains a number of files essential to the developer together with full documentation and copious programming examples. Among other tools it contains:

- tini.jar: Includes the JavaKit and TINIConverter programmes.
- tiniclasses.jar: The class files for the TINI API.
- tini.tbin: The binary image of the TINI Runtime Environment.
- slush.tbin: The binary image of the slush command shell.

Once the above packages are installed JavaKit should be run to kick start the TINI board into action. A straight RS232 cable should be first be attached between the PC and the board and JavaKit used to load both the command shell slush and the runtime environment using the JavaKit's LoadFile command from the File menu. Once loaded, it is necessary to load the first 64kbytes of heap with zeroes. This is done by typing the commands:

- BANK 18
- FILL 0

and finally

- EXIT

This last command quits the serial loader and launches first TINOS and then slush. To log in to as root use *root* and *tini* as the username and password respectively. Slush is a shell with a Unix like command set and feel and anyone familiar with Linux should have little difficulty navigating around. Help is available by typing *help* which prints a list of the available commands. Typing *help* followed by a command gives brief information as to how the command is used.

5.1 Setting up the Ethernet connection

Whilst the serial link is a useful way of booting the system for the first time, the real power of TINI comes from its networking abilities. However, before leaping into setting up TINI as a web server, the Ethernet adapter must first be configured. Before starting this an IP address should be obtained from the system administrator if a Dynamic Host Configuration Protocol (DHCP) server is not available. The command *ipconfig* can be used to set the network properties. Just typing this command will give a list of the options available under TINIOS. If a DHCP server is not present, typing *ipconfig -a xx.xx.xx.xx -m yy.yy.yy.yy* where xx.xx.xx.xx and yy.yy.yy.yy are the domain and subnet mask respectively will be sufficient as a minimum configuration. TINIOS will issue the prompt:

- OK to proceed? (Y/N)

and after entering *y* the on board ftp server will start (assuming an Ethernet cable has been attached). The TINI Board should now be able to communicate with other hosts using the standard set of network utilities (ping, telnet, ftp etc).

5.2 Programming TINI

Writing programmes for TINI involves creating the source file on the host machine, compiling, converting the class file to a loadable image, downloading the image to TINI and finally running the code on TINI under slush. Writing the source file can be done using any text editor and saving the resulting file with a *.java* extension. Compiling rather depends on the particular environment chosen. Using the Java SDK, type:

- javac Progname.java

and the programme should compile and, if there are no error, produce the output file: *progname.class*. The next step is to convert the class file into a format acceptable to TINI. This is accomplished using the TINIConverter application from Tini.jar. This application needs the following arguments:

- -f Input file
- -d API Database
- -o Output file

Thus converting the file Progname.class to Progname.tini would need:

- java -classpath tini.jar TINIConverter -f Progname.class -d tini.db -o Proganame.tini

Here it is assumed that the directory paths for Java, TINIConverter and tini.db have been included in the environment PATH variable, enabling a much simplified command line.

The final step is to transfer the resulting binary file to TINI over the Ethernet connection. This is done by starting a ftp session on the host and connecting to TINI. As before login using *root* and *tini* and issue the commands

- bin
- put Progname.tini
- bye

At this point, the programme is loaded in TINI but has yet to be executed. This can be done by opening a telnet session and executing:

- Java Progname.tini

This will cause the programme to execute.

Package	Description
com.dalsemi.comm	Low level classes for CAN the controllers and serial port configuration
com.dalsemi.fs	Extensions to the standard java.io.File class
com.dalsemi.io	A few classes to aid conversion from specific encoding schemes
com.dalsemi.onewire	Classes for accessing Dallas Semiconductors onewire communications protocols
com.dalsemi.protocol	Classes used by various networking protocols
com.dalsemi.shell	Implementations of command shells for ftp and telnet servers
com.dalsemi.system	Classes for accessing the native hardware of the DS80C390
com.dalsemi.tininet	Classes for both configuring and manipulating networking hardware, and providing support for network protocols

Table 5: com.dalsemi packages

5.3 Using TINI's hardware capabilities

Whilst a discussion on the nuts and bolts of programming every facet of the the standard TINI platform is subject matter of a complete book, an overview is not out of place.

The drivers for each item of standard hardware (i.e.the inherent DS80C390 peripherals and the device appearing on the TBM390 board) are wrapped by Java classes. In some cases, where the devices fulfill functions common to most Java implementations such as RS232 communications, the classes are included in one or other of the standard packages. However, in the majority of cases, the hardware is non-standard and customised classes have been provided. These can be found in a group of packages known collectively as the **com.dalsemi** packages, and which are included with the TINI SDK. Also included is full html documentation for the entire set of packages.

There are seven packages included in com.dalsemi outlined in table 5

6 Summary

The information presented in this chapter, has of necessity been a brief introduction to the hardware, operating system and use of the TINI platform. For

Site	Web page
The TINI Interest Group	www.maxim-ic.com/TINIdocs/interest.cfm
Unofficial TINI Information site	www.smartsc.com/tini
Java Guru	www.jguru.com/faq/TINI
TINI Resources	www.apms.com.au/tini
Brent's TINI page	www.nordist.net/TINI
TINI Board Resource Center	www.junun.org/TINI
TINI binding for JXTA	www.tini.jxta.org
TINI Board WebRing	www.vinculum.com/webring
TINI development in C	www.vigor.nu/tini.html

Table 6: com.dalsemi packages

practical applications the reader is always referred to the documentation accompanying the TINI SDK (ref).

The platform is a remarkably compact and versatile system with a rapidly growing and enthusiastic user base. There exist a number of web sites devoted to its promotion and use, which are more than worth a casual browse (table 6.

References

[1] Don Loomis (2001). *The TINI Specification and Developer's Guide*. Addison-Wesley

[2] *www.java.sun.com*

[3] *www.ibutton.com/TINI* 5

[4] *www.rxtx.org* 5