TINI Networked Microcontrollers

Networked microcontrollers include features for network enabling embedded systems, both in the classical sense of Ethernet and a wide variety of lower level networks.

The Tiny Internet Interface or TINI for short, is a microcontroller that includes the facilities necessary to connect to the Internet. The TINI platform is a microcontroller-based development platform that executes code for embedded web servers. The platform is a combination of broad-based I/O, a full TCP/IP stack, and an extensible Java runtime environment that simplifies development of network-connected equipment.

TINI was developed by Dallas Semiconductor, and therefore has an additional API to work with 1-Wire devices. This microcontroller has a lot of issues it has very powerful networking capabilities and has lots of ways to interact with it. Java enabled the easiness of programming the hardware with out the intervention with the low level languages. With its many ports you can control many signaling devices to enable external circuits and components.

Here I will brief you with the basic component of TINI and its hardware installation , software configuration and I will let you take a look of some of the application with specific details of each port used.

After you receive the TINI kit it should contain the following hardware pieces:

1 - The TINI Board Module (DSTINIm400)

2- The TINI Evaluation Board (DSTINIs400)

3 - A straight serial cable.

4- 5V DC power supply with 300 mA.

5 - Ethernet cable and access point of 10/100 Base-T Ethernet.

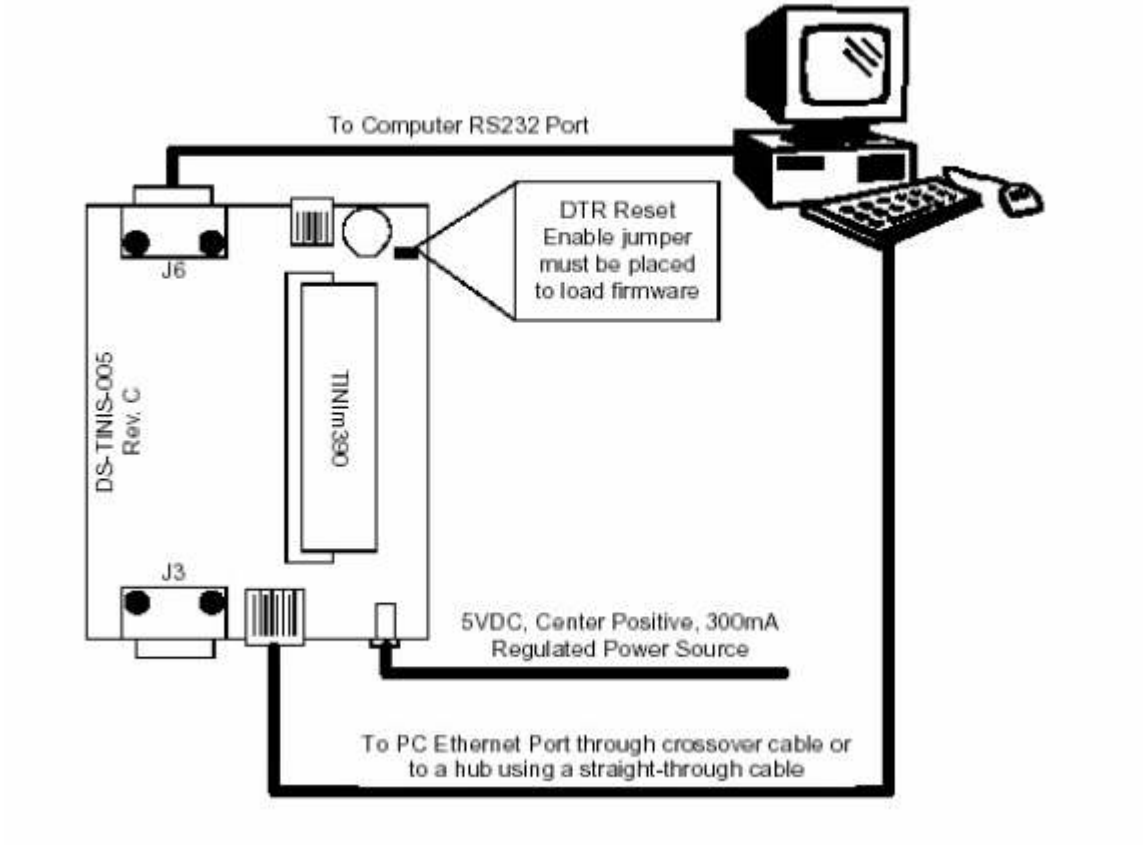
The **DSTINIm400** contains the core system including a DS80C400 Microcontroller, 1MB Flash Memory, 1MB SRAM, a real-time clock, 3 hardware serial ports, integrated 1-wire Network Master and a CAN2.0B port. The circular lithium battery that is mounted on the upper right-hand corner is to save the file system that is on the SRAM after lugging-out the power. The DSTINIm400 board can be interfaced using a 144-pin SODIMM connector. In order to fully evaluate the board, we need to have proper interfaces to its different components; this is offered by the DSTINIs400 Evaluation board that is discussed next. The datasheet for the DSTINIm400 can be found in
http://pdfserv.maxim-ic.com/en/ds/DSTINIM400.pdf

The DSTINIs400 is shown in Fig 1-2. The Board provides interfaces to different components of the DSTINIm400 module such as serial ports, Ethernet, power, 1-wire bus, CAN2.0B bus and different I/O pins. The DSTINIm400 can fit into the 144-pin SODIMM connector of the DSTINIs400 board. The board needs a 5V power input to operate. The datasheet for the DSTINIs400 can be found in
http://pdfserv.maximic.com/en/ds/DSTINIS-005-DSTINIS400.pdf

Hardware Installation

1. Slide the DSTINIm400 module into the 144-pin connector of the DSTINIs400 Evaluation Board.
2. Connect the DSTINIs400 to the serial COM port of the PC using the straight
serial cable. Use the female-connector of the DSTINIs400.
3. Plug the power adapter to the power jack of the DSTINIs400. Make sure that the power adapter is set to +5V.
4. Plug the network cable into the Ethernet interface of the TINI Evaluation Board.

This completes the hardware installation part is sown here:



For The Software setting up:

Caution: this is a very important step and the fate of your project will depend on how you abide by theses instruction ,hence follow it carefully.

To start working with TINI, you must have the following software pieces:

1- Java Development Environment (such as JCreator LE or any other JDE).

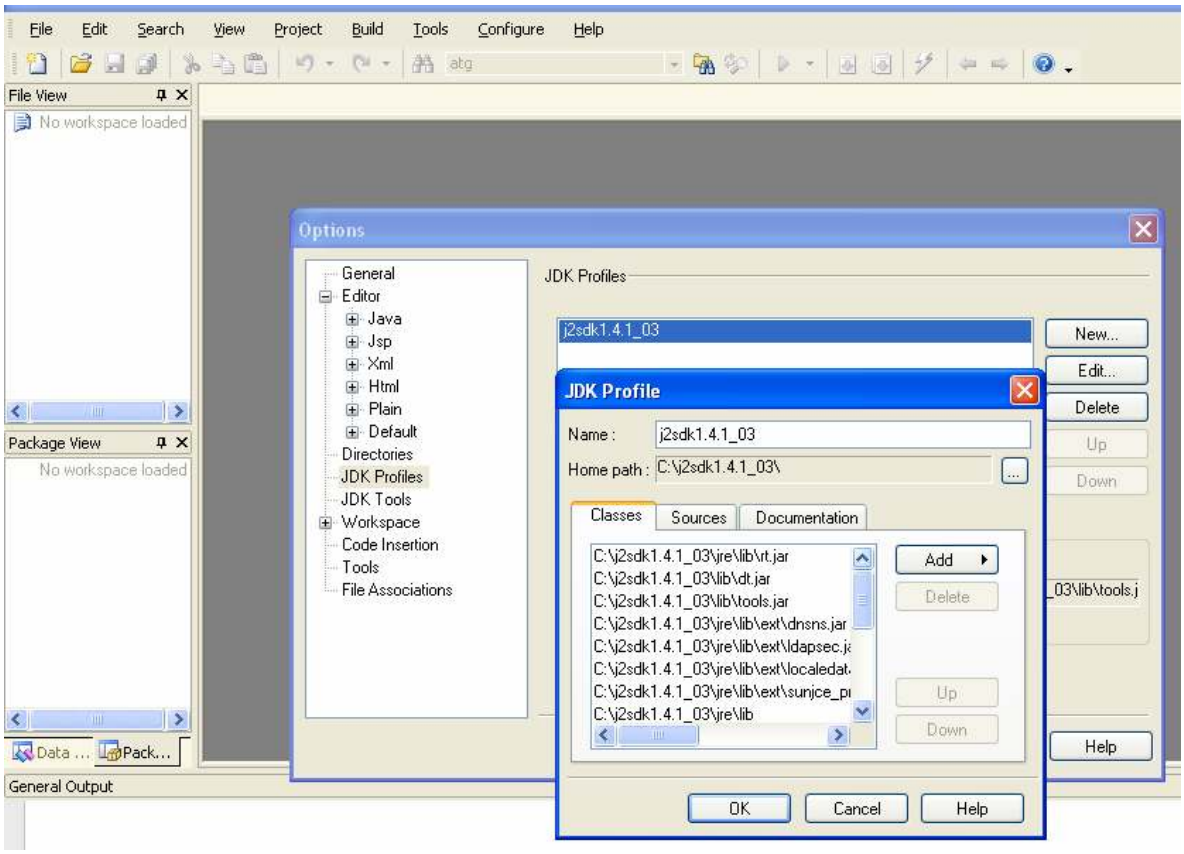
2- Java Development Kit (JDK) version 1.18 or newer (preferably V1.4.1)

The version used in our project is(V1.4.1_03) it is available in : \\ccse-software
You can access it from the academic buildings or from the student dorm with the following authentication:

USER:	**CCSE_NT_DOMAIN\st222222**
PASSWORD :	******

Whre st222222 is replaced with your ID #.
*And ****** is (your password) ccse password.*

Then go to the JDK folder under Java related subfolder \\ccse-software\UnIns_Apps\JavaRelated\JDKs\JDK1.4
Note that you can try other JDK version but it might not respond well, so it is recommended to install the mentioned version ,cause we faced problems with the V1.5.0 for the Tiniconvert commands. Bottom line, use jdk1.4.1_03.



3- Java Communications API.
4- TINI Software Development Kit (TINI SDK).

In addition to the basic JDK, you need two java packages to be able to develop Java programs that run on the TINI environment. The first is the Java Communication API commapi) that provides some API's to allow serial port operations. The package can be downloaded from http://java.sun.com/products/javacomm/downloads/index.html.

The other package is the TINI Software Development Kit that includes classes specific to TINI. The version available at the time of writing these lines is (tini V1.15). These classes allow you to control the TINI hardware from a high level using Java API's. An Example of such applications is reading from and writing to some I/O ports. Another example is controlling devices on the 1-Wire bus. The package can be downloaded from ftp://ftp.dalsemi.com/pub/tini/tini1_15.tgz.

Software Installation

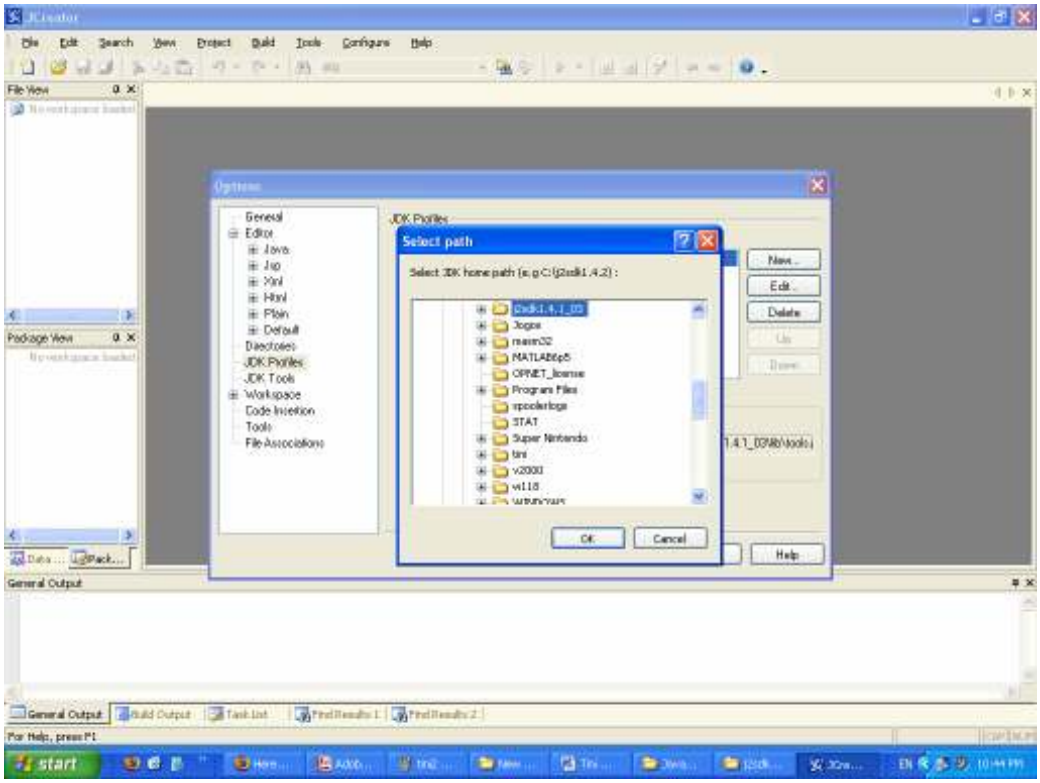
Note: The procedure given assumes that the Java JDK resides in the <JDK> directory which could be any directory on your PC.

In my case I have it in C:\j2sdk1.4.1_03\ directory.
So to easy the upcoming procedure it is preferable to do as such strictly.

For all the steps that follow just replace <JDK> with the directory you are using. This applies for other directories you will see like <comm> and <TINI>. The procedures given apply to Windows 2000 and Windows XP. There might be a need for some changes under other versions of Windows

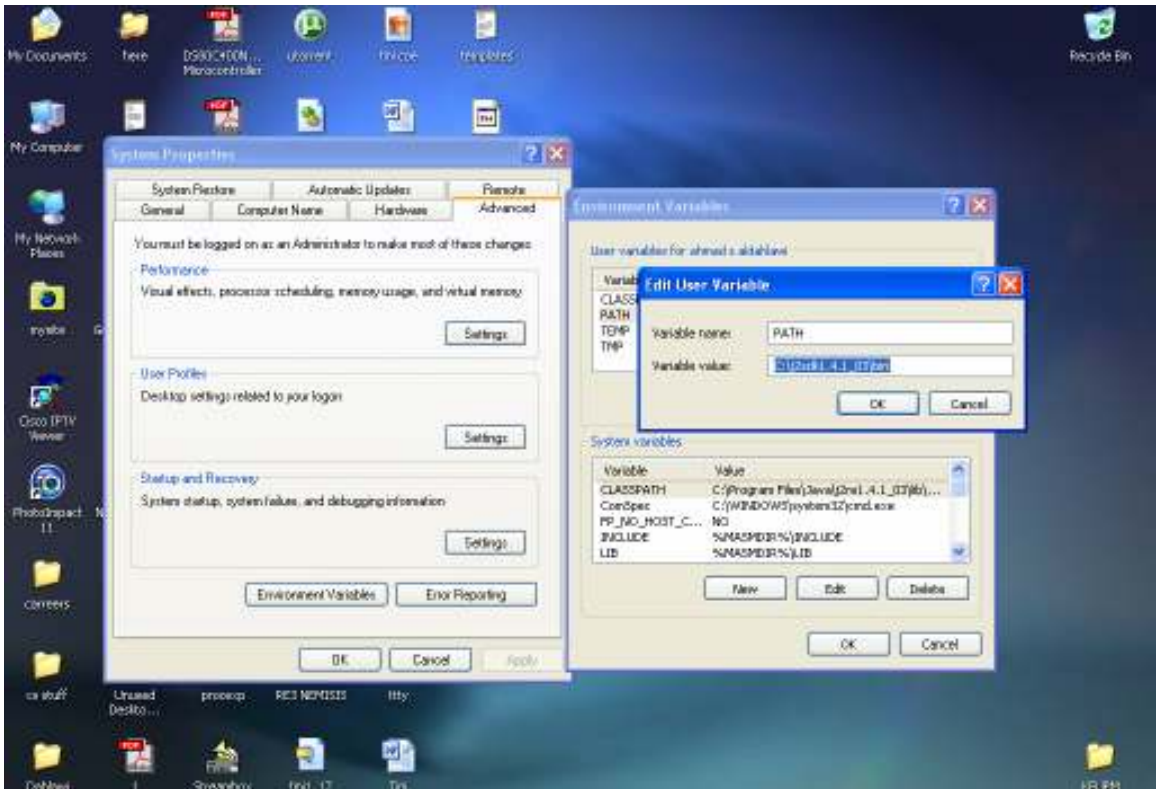
Configuring the Java Development Kit (JDK)

1. Run the JCreator.
2. Select Configure> Options.
3. Select New and traverse through the directories and select your JDK folder.
4. Press OK. The JDK path has been added to your JCreator.
5. Press OK to close the Options menu.



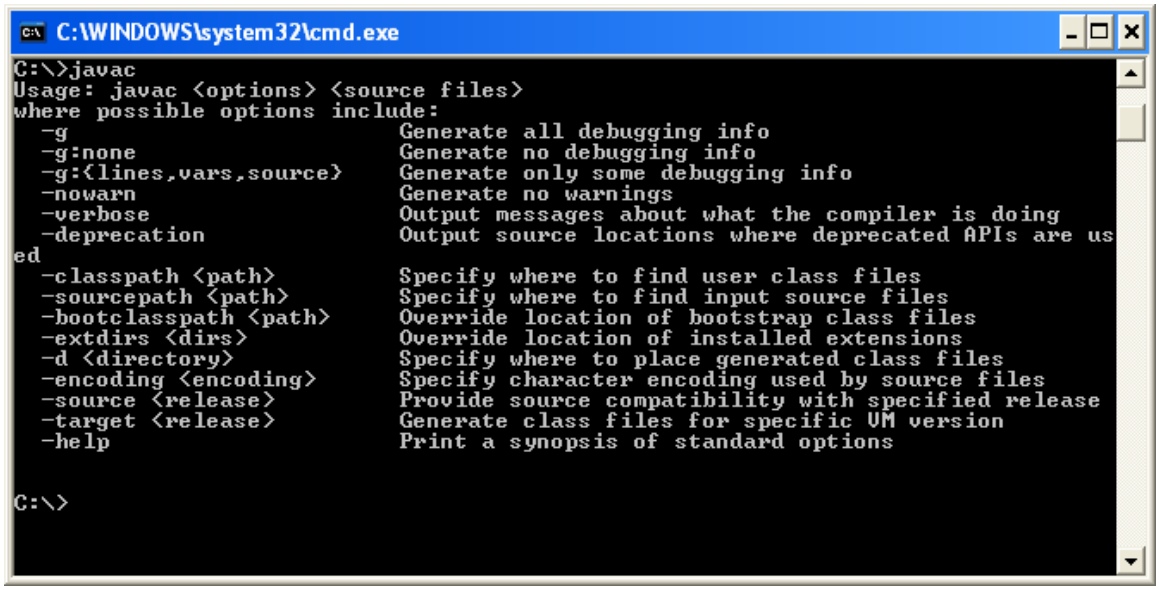
Now, you need to set the PATH variable to include the <JDK>\bin directory:

1. Right-click on **My Computer** and choose **Properties**.
2. Select the **Advanced** tap.
3. Click on **Environment Variables**.
4. Under the **User Variables** choose the PATH variable and select **Edit**.
5. Add the directory <JDK>\bin to the PATH. Use separate ";" to separate different values of the PATH.
6. If no PATH variable exists on your PC, create new one by clicking **New**.



Note: when you add the <JDK>\bin to your path, make sure that it precedes the %PATH% value that contains the system paths. The problem that may occur is that when you run **java.exe** command, the C:\WINDOWS\system32\java.exe file will run instead of the <JDK>\bin\java.exe and the installation of the *commapi* that comes later will not work. If you encounter any problem during the installation of the *commapi*, recheck this note.

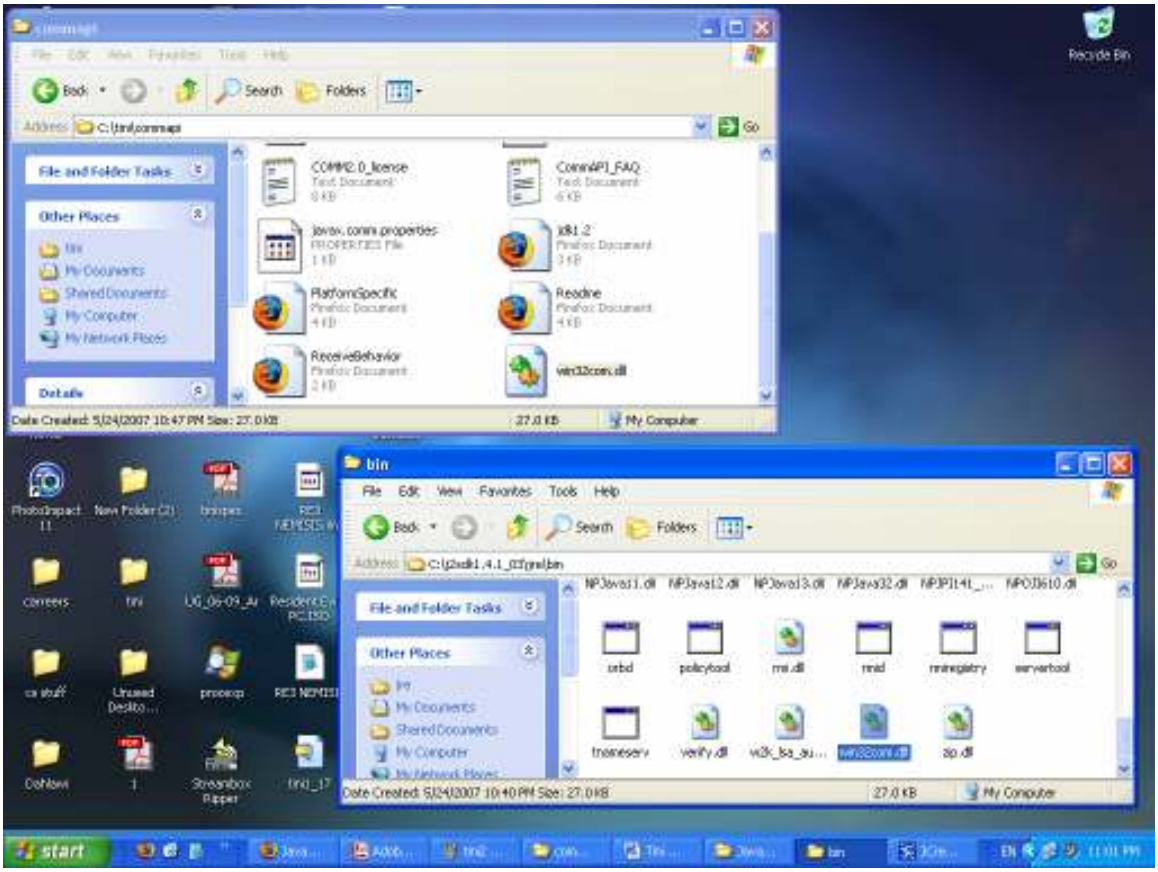
To make sure that the PATH variable is set correctly, open the Command-Prompt by clicking **Start>Run** and typing **"cmd "**and Press **Enter**. Type "**javac**" and press **Enter**. If the command "**javac**" is



recognized, then OK. Otherwise, you need to go back and check the previous steps you followed.

Configuring the Java Communications API (commapi):

Note: This procedure uses commapi version 2.0, for more info about other versions, read the ReadMe.html file available with the package.
1. Unzip the file javacomm20-win32.zip into <comm> directory. This will produce a hierarchy with a top level directory being commapi. Preferably unzip it in C:\tini\commapi directory.
2. Copy win32com.dll to your <JDK>\jre\bin directory.
.(i.e from C:\tini\commapi to C:\j2sdk1.4.1_03\jre\bin)



3. Copy comm.jar to your <JDK>\jre\lib\ext directory.
4. Copy javax.comm.properties to your <JDK>\JRE\lib directory. Also:

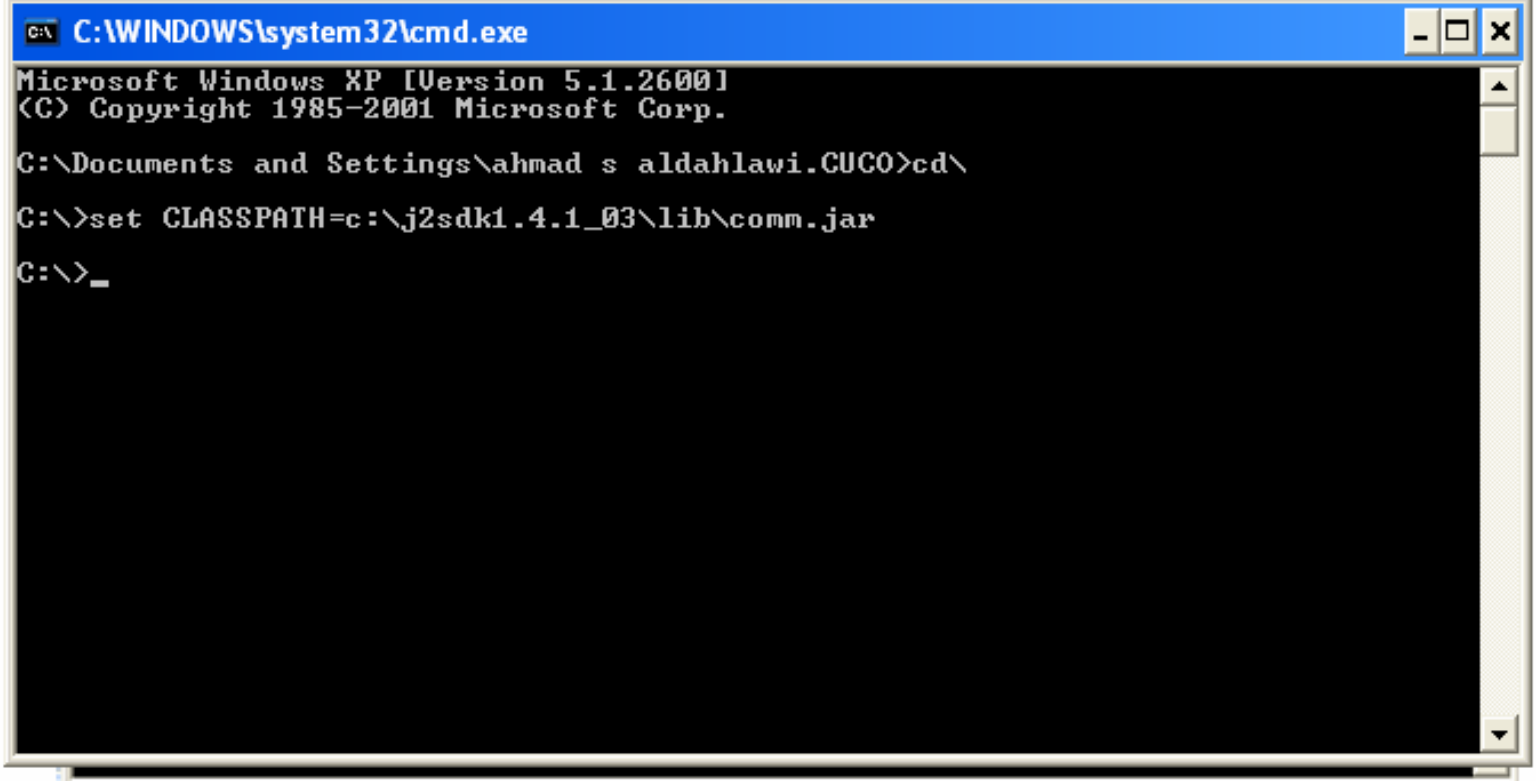
    1. Copy win32com.dll to your <JDK>\bin directory.

    2. Copy comm.jar to your <JDK>\lib directory.

    3. Copy javax.comm.properties to your <JDK>\lib directory.

    The javax.comm.properties file must be installed. If it is not, no ports will be found by the system.

    4. Add comm.jar to your classpath (do not do this step for a JRE installation).
        - If you don't have a classpath defined:
          in the command prompt do this



- C:\>set CLASSPATH=c:\ j2sdk1.4.1_03\lib\comm.jar

5. If you already have a classpath defined:

   C:\>set CLASSPATH=c:\ j2sdk1.4.1_03\lib\comm.jar;%classpath%

6. Several serial port sample applications are provided with this release. One of them is BlackBox. To run BlackBox, first add BlackBox.jar to your classpath:

   C:\>set CLASSPATH=c:\tini\commapi\samples\Blackbox\BlackBox.jar;%CLASSPATH%

   Now you can run BlackBox:

   C:\>java BlackBox

Running with the JRE

Use the -cp option to the JRE to specify the path to the comm.jar file and to your application.

```
C:>\JRE\bin\jre -cp c:\JRE\jre\lib\comm.jar;c:\commapi\samples\Blackbox\BlackBox.jar BlackBox
```

The javax.comm.properties file must be correctly installed (in the <jdk>\lib directory, see above for more details) or no ports will be found.

The examples in this document assume that you have unzipped the javacomm20-win32.zip file in your C:\tini partition and your JDK installation is in C:\ j2sdk1.4.1_03. If you have installed JDK in an other location or unzipped javacomm20-win32.zip in an other location modify the example commands appropriately.

If you are installing the Java communications API into a JRE (Java runtime environment) follow the same instructions as for the JDK except where noted. See the additional instructions for running using the JRE.

To test your installation, do the following:
1. Set the ClassPath variable to include to the <comm>\samples\BlackBox\BlackBox.jar file, if no ClassPath is available
on your system, create new one.
2. Run the Command-Prompt and type “C:/> java BlackBox”.
3. If the program runs, then OK. Otherwise, you need to go back and check the
previous steps you followed.

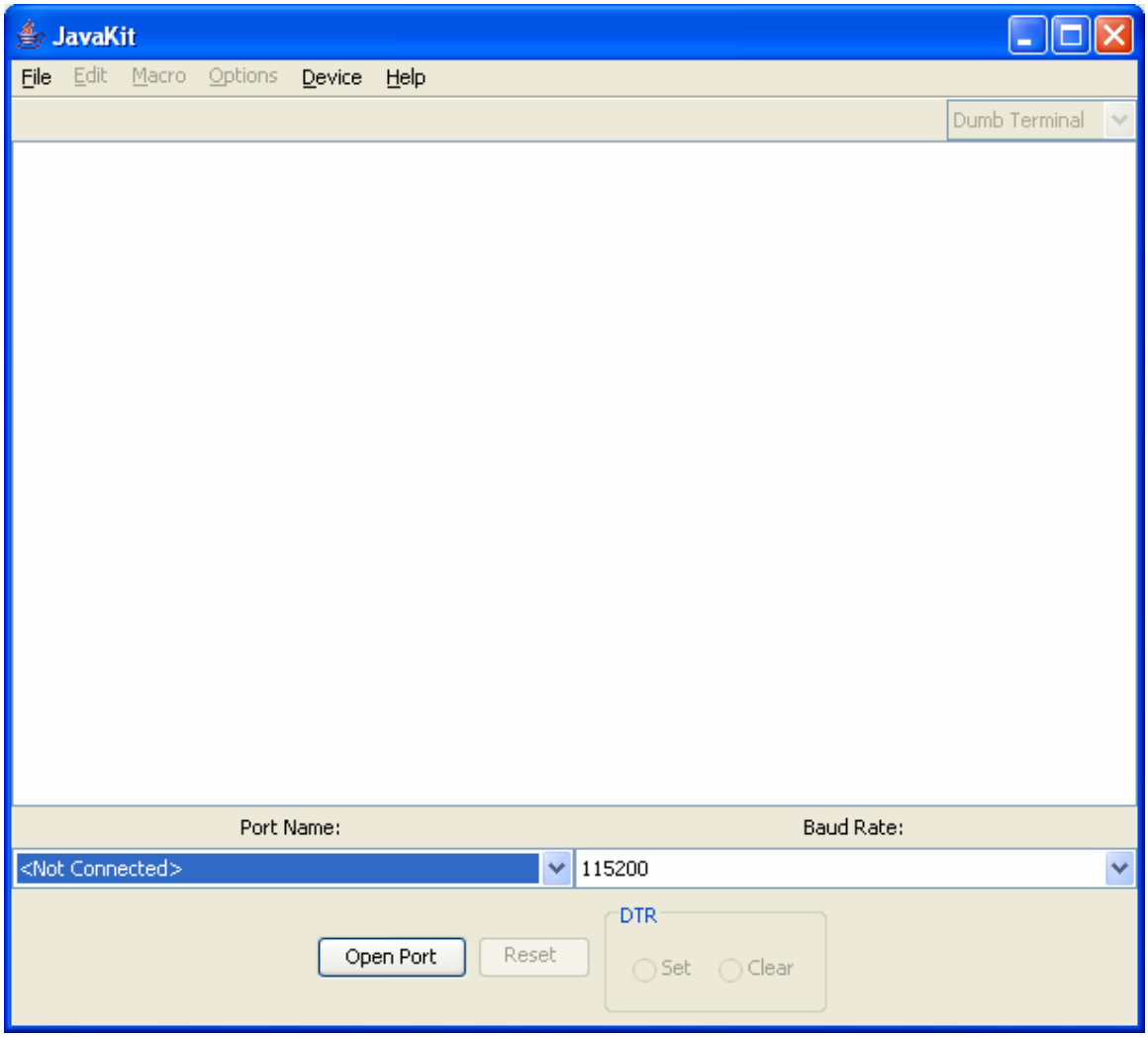
By now you have finished the installation of the JDK and the communication API
Next we will introduce you to the tini Kit installing methodology to communicate directly thru the serial cable.

Configuring the TINI SDK
1. Unzip the tini1_15.tgz into a <TINI> directory.
(i.e in C:\tini)
2. Copy the tini.jar and the tiniclasses.jar from <TINI>\bin into <JDK>\jre\lib\ext.

Starting the JavaKit

The JavaKit is a Java application that enables you to work with TINI in a friendly manner. The application interacts with the TINI board through the serial port. To start the JavaKit:

1. Run the Command-Prompt.



2. Type Java JavaKit

you can also launch JavaKit using the detailed command line

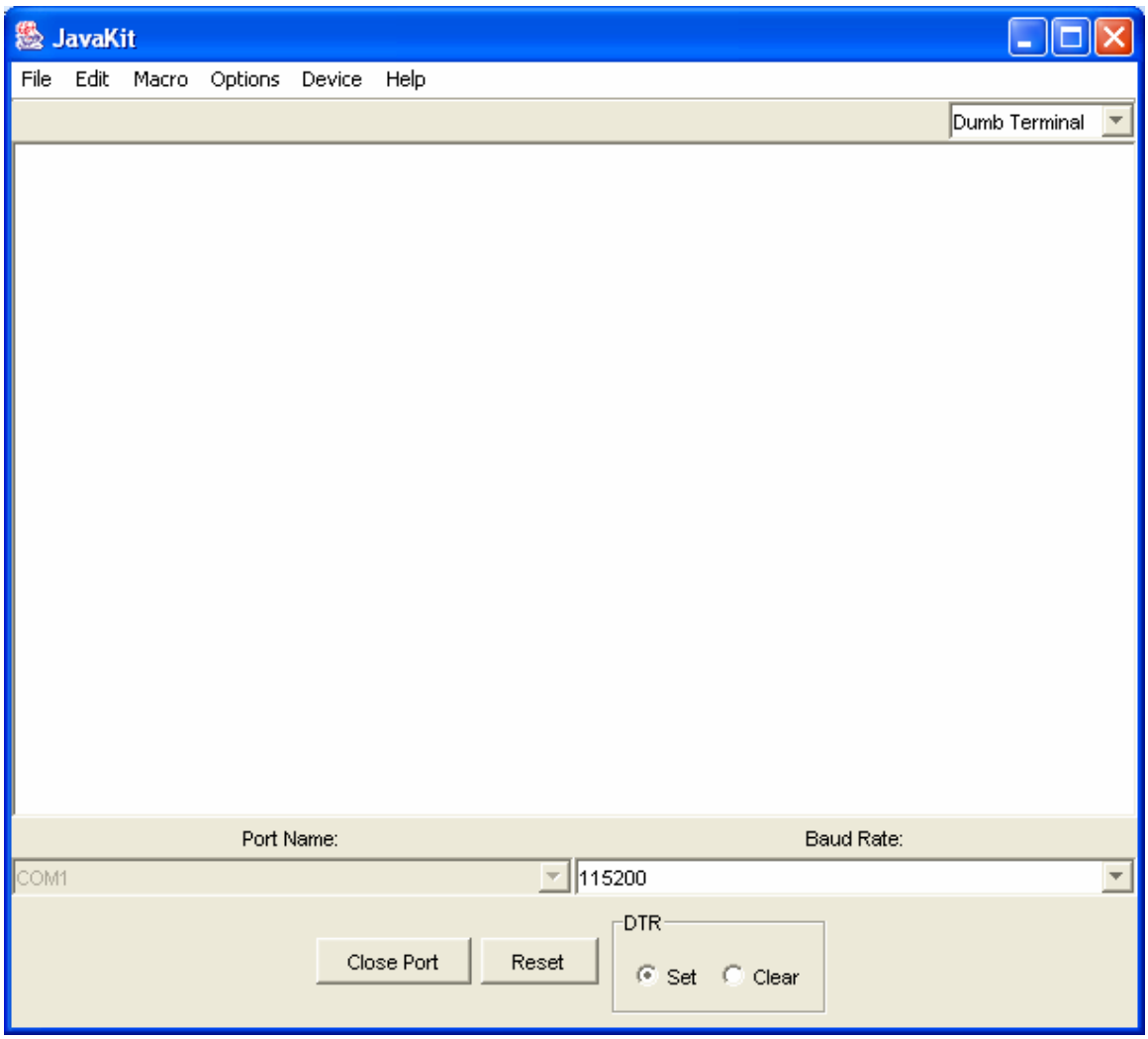
c:\j2sdk1.4.1_03\bin\javaw -classpath C:\tini\tini1.17\bin\tini.jar JavaKit

this will work by specifying the correct directory and the classpath, in case it is not already fixed.

3. From the **Port Name** drop menu select the COM port you attached your device to.

4. From the **Device** menu, choose **TINIm400**.

5. Press the **Open Port** button. If the JavaKit is able to open the selected serial port, the **Open Port** button will be changed to **Close Port**. If the



Open Port doesn't change to Close Port, then you are not connected to the board and you need to go back and check your previous steps.

For The First Time Only

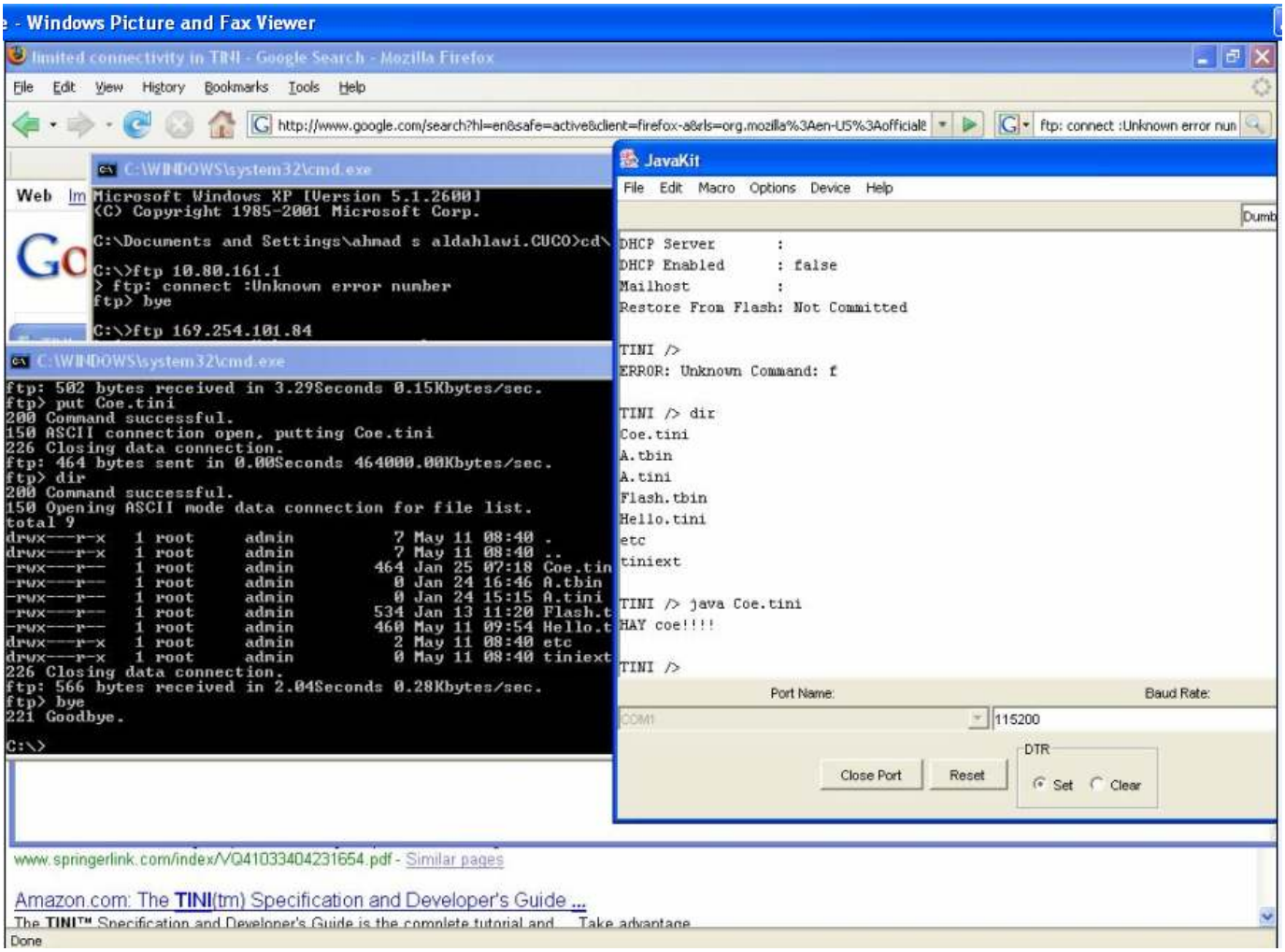
Now, you are connected to the board, we need to load the TINI's Runtime Environment (tini_400.tbin) and the Slush Shell (Slush_400.tbin).

1. From the **File** menu select **Load File**.
2. Traverse the directories and go to <TINI>\bin and select the tini_400.tbin and the Slush_400.tbin files.
3. Press **Open**, the two files will be loaded. This may take some time due to the large size of the files.
4. Type "Exit" in the command prompt and press **Enter**. The Slush_400.tbin is a shell that runs on the microcontroller and executes the commands; you can look at it as a micro-operating system. The tini_400.tbin is a the TINI Interpreter that translates your *.tbin and *.tini

files into a instructions understandable by the hardware. You can think of the latter as TINI Runtime Environment.

Note: Loading the system files (tini_400.tbin and Slush_400.tbin) should be done only the first time you use the board. Loading the files will result in creating a new file system and all the old system configurations will be lost. To keep you file system and configuration skip the load step by typing “exit”.

Press Enter in the JavaKit program and wait till you get a prompt.
Now, to login to the TINI environment use the following user name and password;
Username: root
Password: tini
Now, you are logged in and can try to execute some commands. TINI uses UNIX-like
commands. Type “help” to get a list of the commands available.



You can try some commands like dir and ls for listing the files in the TINI. Java File.tini will run the java program and give the output immediately.

Configuring TINI Network Interface

In order to be able to use the Networking capability of the TINI board, you should configure Network interface with a proper IP address.For this example we will use X.X.X.X as the IP address and Y.Y.Y.Y as the network mask.

To configure the IP, just type the following line

ipconfig –a X.X.X.X –m Y.Y.Y.Y

eg. **>ipconfig –a 192.168.1.1 –m 255.255.255.252**

and press **Enter**.

To check your IP specifications of your board type

ipconfig

and press **Enter**. You can see the configuration you made. Alternatively, you can configure TINI to use DHCP to get an IP address, but I did not cover this here.

After configuring the Ethernet you no longer have to use the serial port or JavaKet you can use Telnet from command prompt from the PC to access the TINI and in my opinion it is easier than using JavaKet.

<telnet xxx.xxx.xxx.xxx>

Converting Java Classes into TINI *.tini files

In order to be able to run you Java programs on the TINI microcontroller, you need to convert them to a format that TINI runtime environment can understand. For this we use a special class that comes with the TINI SDK called TINIConvertor.jar. to start the procedure, let us first build and compile a simple Java application. Make a new source file with the JCreator and name it HelloTINI.java. Copy the following java program into the source file and compile it. You should have a class called

HelloTINI.class.

```
import java.util.*;
public class HelloTINI{
public static void main (String[] coe)
{
System.out.println("Hello, TINI is finally responding");
}
}
```

To convert the file, you need one more file that is found in the TINI package. The file is named tini.db and its in <TINI>\bin. The tini.db must be in the same directory as the
HelloTINI.class.
Open the command prompt and traverse through the directories looking for the directory in which you have HelloTINI.class. Type the following command
java TINIConvertor –f HelloTINI.class –o HelloTINI.tini –d tini.db
and Press Enter. You should have a new file called HelloTINI.tini that you are going to
upload to the TINI board after a while.
You can always specify the detailed command if the previous one is not working
java -classpath C:\tini\tini1.17\bin\tini.jar TINIConvertor -f **HelloTINI.class** -d C:\tini\tini1.17\bin\tini.db -o **HelloTINI.tini**
note that you only have to use one of these commands only.

Uploading a Program into the TINI board
The only way to upload a program into the board is by using the File Transfer Protocol.
From your PC, ftp into the TINI board by typing
ftp X.X.X.X
we used ip : 192.168.1.1 for the tini and 192.168.1.2 for the PC connecting to it.

in the command-prompt and press **Enter**.
Now, login to the board using the username: *root* and the password: *tini*.
To switch into *binary*
file transfer mode, type
>bin
and press **Enter**.

To upload the file type:
>put c:\TINI\HelloTINI.tini
where the file name is the complete path of the file. Press Enter and the transfer will take
place.
To see the file, type
>ls
And press Enter in the JavaKit prompt.
To run the program type
Java HelloTINI.tini
And press **Enter**. You should see the "Hello TINI" message coming up.
By this we complete all the steps you need to start your TINI journey.

TINI Homepage
Application notes, datasheets and latest released software can be found in Maxim and
IButton web sites.
http://www.maxim-ic.com/TINIplatform.cfm
http://www.ibutton.com/TINI/book.html

Exercise1-1: My First TINI Run.
The following program flashes one of the (Light Emitting Diodes) LEDs on the TINI
board. The LED is connected to P3.5. Copy this program to a Flash.java Java source file and
complie it. Go through the steps mentioned earlier to convert the file into a *.tini file and
upload it to the TINI board and run it.
NOTE: The program contains an infinite loop. To be able to kill the program after you
run it, execute the program in the background. To do this in TINI (as well as UNIX),
type
java Flash.tini &
If you get any compilation errors, then revise the steps of configuring the TINI SDK
mentioned earlier.

```
// This Java program flashes one green LED on the TINI board at a frequency of 2Hz
import com.dalsemi.system.BitPort;
import com.dalsemi.system.TINIOS;
class Flash {
```

```
public static void main(String[] args) {
BitPort bp;
if (TINIOS.getCPU() == TINIOS.DS80C390)
bp = new BitPort(BitPort.Port3Bit5);
else
bp = new BitPort(BitPort.Port5Bit2);
for (;;) {
// Turn on LED
bp.clear();
// Leave it on for 1/4 second
try {
Thread.sleep(250);
} catch (InterruptedException ie) {}
// Turn off LED
bp.set();
// Leave it off for 1/4 second
try {
Thread.sleep(250);
} catch (InterruptedException ie) {}
}
}
}
```

TINI I/O Ports:
The TINI microcontroller has eight I/O ports numbered P0-P7. Five ports out of those are consumed by the data bus; address bus and chip enable signals, namely P0, P2, P4, P6 and P7. This leaves P1, P3 and P5 available for other usages. Each port consists of eight bits.
The convention when referring to a certain bit of a port is to write the port number and the bit number separated by a ".". For example, to refer to bit 3 of port 5 we write P5.3.

Pin Number	Corresponding DSTINIs400 Signal	Location	Dual Function
P1.0	T2	J27	I²C Bus, Time2 increment
P1.1	T2EX	J27	I²C Bus, Timer2 reload
P1.2	RXD1	---	Serial Port 1 Receive
P1.3	TXD1	---	Serial Port 1 Transmit
P1.4	INT2	---	External Interrupt 2
P1.5	$\overline{\text{INT3}}$	---	External Interrupt 3
P1.6	INT4	---	External Interrupt 4
P1.7	$\overline{\text{INT5}}$	J5	External Interrupt 5
P3.0	RXD0	---	Serial Port 0 Receive
P3.1	TXD0	---	Serial Port 0 Transmit
P3.2	INT0	J4	External Interrupt 0
P3.3	INT1	---	External Interrupt 1
P3.4	$\overline{DRST}$	---	Counter External Input
P3.5	INTOW	---	Status LED
P5.0	C0TX	J20	Transmit data for CAN0, clock signal for 2-wire synchronous serial I/O
P5.1	C0RX	J20-	Receive data for CAN0, receive signal for 2-wire synchronous serial I/O
P5.2	T3	J20	Transmit data for CAN1, receive data for serial 1
P5.3	----	J3	----
P5.4	$\overline{PCE0}$	J21	SPI Bus (chip enable 0)
P5.5	$\overline{PCE1}$	J21	SPI Bus (chip enable 1)
P5.6	$\overline{PCE2}$	J21	SPI Bus (chip enable 2)
P5.7	$\overline{PCE3}$	J21	SPI Bus (chip enable 3)

Table 2-1 : Some pins of the DTINIm400 and their usage.

The BitPort class
The port classes can be instantiated to refer to certain pin on the microcontroller. Using
some methods provided by the class, a programmer can play with the pin. The class can be found in the com.dalsemi.system.BitPort package which must be imported in order to use the class.

The constructor of the BitPort class takes the identity of the pin as a public static constant of type **byte** belonging to the BitPort class. For example, to instantiate an instance of the BitPort referring to pin P3.2, you write:

BitPort bp = new BitPort(BitPort.Port3Bit2);

The BitPort class provides several methods helpful to manipulate the single bits easily. This table shows some methods available in the BitPort class.

Method	Function
public void set()	Sets the corresponding bit value to high (logic 1)
public void clear()	Sets the corresponding bit value to low (logic 0)
public int read()	Returns the current value of the corresponding bit (0 or 1)
public int readLatch()	Returns the value of the last write operation to the corresponding bit (0 or 1)

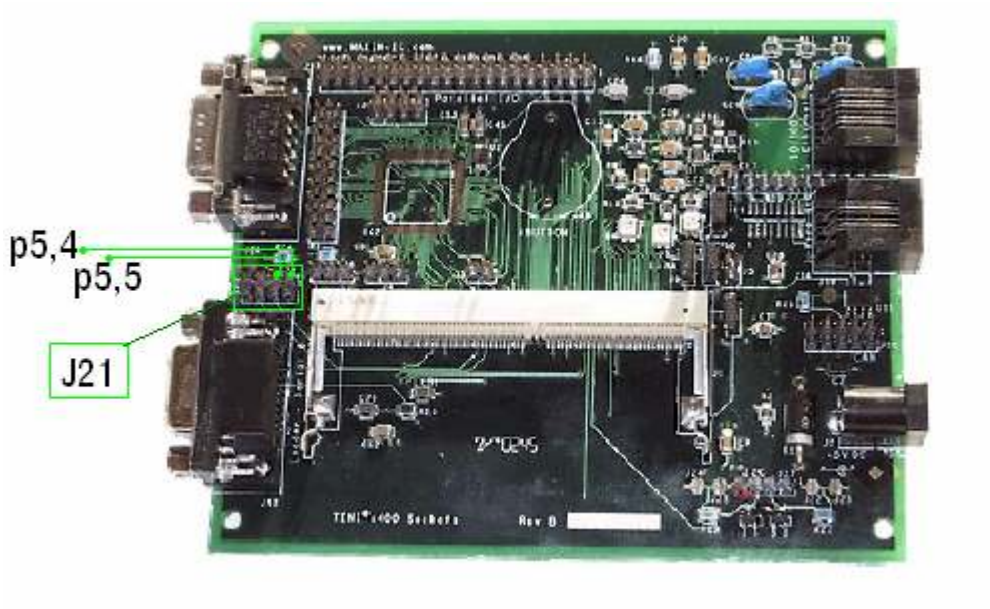
Our Project (Child Home Safety) Implementation:

The program installed in the TINI microcontroller is a TCP Server that opens a socket in the between the application and the transport layer with a specific port number (6789), so that the client connecting in the PC can communicated thru this socket and he must specify the destination IP address along with the port number.
After opening a welcoming socket in the server
ServerSocket welcome_socket = new ServerSocket(6789);

The two pin in the J21, of port 5 shown in the figure are going to be used, the initialization of bit 4 and 5 in that port are done as follows:

bp1 = new BitPort(BitPort.Port5Bit4);
bp2 = new BitPort(BitPort.Port5Bit5);

now the server will be running consciously(Listening) in the TINI until a message is sent to it. If the message indicate a child has been detected. The port bits bp1 and bp2 are going to be set and cleared respectively.
This will result in cutting the solid state relay with is connected n one of its ends to P5.4. Hence the Kitchen appliances such as the Oven and the light will be disconnected from the electricity.
While the other pin P5.5 will enable the LED that resembles the magnetic lock that will lock the drawers which might contain sharp and hazardous material.

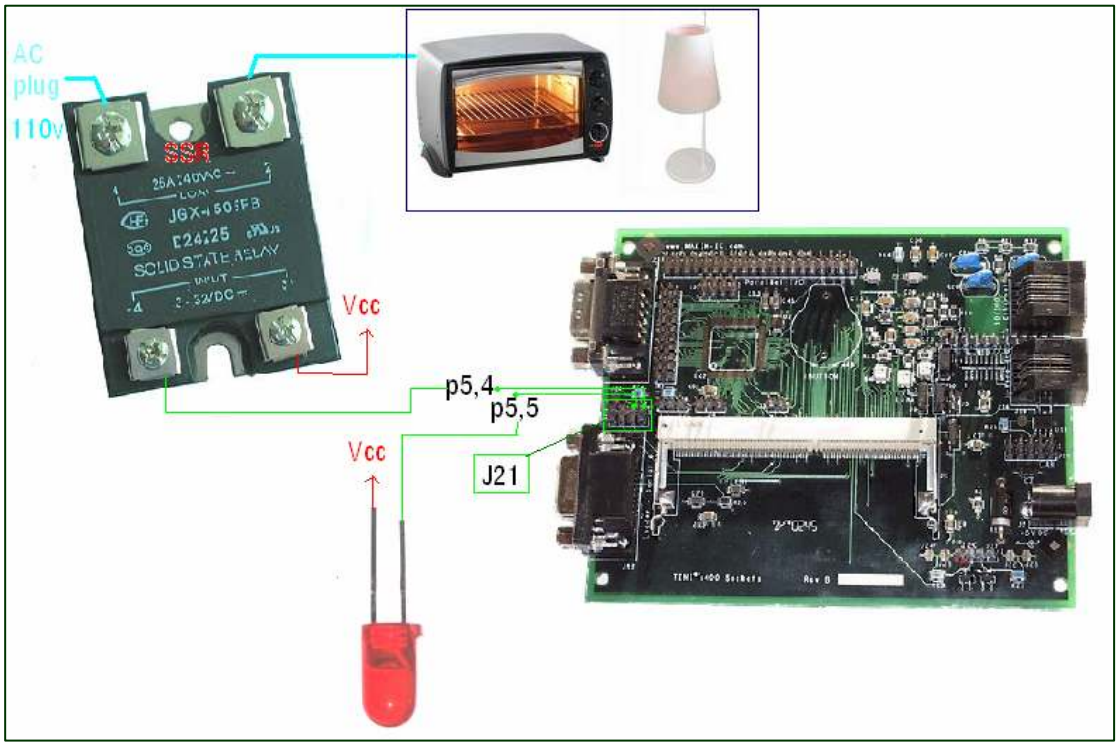


The Hardware Connections :

The reality behind the hype is the physical connection with an innovative simple way, not including any extra resistors or capacitors. Let me talk about my unique experience with the TINI Microcontroller. The first thing that came to my mind is to connect an LED for testing. From an electrical point of view the LED has a tolerance voltage and it would burn if an excess current flow in to its sides. So testing the flash.tini program in an external LED has the following results. At the start the LED seem to be on, when connecting the positive leg to the port5 bit4 with a pull up resistor ,and the negative leg to the ground. After running the program the LED would be turned off and never turn on again . The reason is that when the pin the microcontroller is reset to zero voltage, and then set high again, it will output a low voltage, barely enough to run the LED. One classic solution is to use an amplifier wither a transistor or and op amp.

However, the question of "why not" to try a new solution to keep the circuit neat with no intruders. The new approach was carried out by reversing the LED's conections. That is, the positive leg is connected to the VCC and the signaling pin P5.4 is connecting to the negative leg of the LED,(Common Vcc instead of Common GND). Here when the Pin P5.4 is reset (low) the LED will be ON , and when P5.4 is set (high) the deference in voltage will be insignificant making the LED or any circuit to be clogged off.

The details of the final connection is here:



It worth knowing that to launch the program in the TINI microcontroller from the Client ,Telnet is used to connect to the server and run the program using

>Telnet 192.168.1.1

>java TcpServer2.tini

Now the program is running and waiting for any change to be executed by detecting the RFID tags appended to the child.

The Sound Alarm Making:

The group leader Motasim has recorded his voice using a PC microphone Then the merging of alarm sounds and voice was done using MS moviemaker with .wma (windows media audio) format but because Java does not support wma , the sound file were transformed to audio waves .WAV using fairStar Audio converter application.