Evaluation of Single Board Computers for the ATA Antenna Controller

G. R. Harp
Dec. 05, 2001

ABSTRACT
Eight off-the-shelf single board computers are considered for application as the ATA antenna controller. The evaluation process used the following procedure: we developed an equivalent small program on each computer. This program communicates over the network (Ethernet) to a remote host, and makes some simple tests of the network bandwidth. The boards are evaluated according to 1) the measured performance of the board and 2) the time it takes to develop the software. Based on these tests, we conclude that of all boards tested, one based on the Ajile aJ-100 processor is the best choice.

Introduction
The baseline design of the antenna network has a single board computer (controller) at every antenna and connected via Ethernet to a remote host computer network. One or more host computers issue commands to and receive data from the antenna controllers to control antenna pointing, antenna-based electronics, cryogenics, and various devices.

This document attempts to answer several questions related to these computers:

1. What is the quality of the development environment? How easy or hard are they to program or reprogram?

2. How mature is the underlying software (e.g. libraries or OS)? Is it bug-ridden?

3. What communications efficiency can be expected?

4. What processing efficiency can be expected?

Finally, there is another, overarching question: Is the baseline design feasible at all? At the outset the author and various other members of the ATA team believe it is. This document helps to answer this question by providing benchmarks (communications speed, processing power) from which conclusions can be drawn relating to baseline feasibility.

Choice of Boards for Study
Because of the short development timeline for the ATA, we sought an off-the-shelf solution for the antenna controller. A wide range of single board computers (SBC’s) were considered with prices varying from less than $100 to $1000, with a target price not exceeding ~$500. There are hundreds of SBC’s in this price range, so some other factors were considered. Firstly, the board must support TCP/IP over Ethernet, have a re writable
persistent memory (flash, no disk drive), and have a minimum of on-board I/O (e.g. serial ports, digital I/O, with bonus points for hardware timers, CANBUS interface, or other options we foresee using). We put a priority on ease of use, placing high value on systems that appear to provide mature development environments. We had a substantial bias toward SBC’s that support Java programming. Because Java is the native language of the ATA software, we believe that Java at the controller can save substantial development time on the part of the ATA software team.

A final, parenthetical criterion arises from ATA requirements for distributed control of the IF system. It would be convenient if the same SBC used at the antenna could also be applied for control of the IF. The processing requirements on the IF controllers are more stringent in the sense that the update rates for amplitude and phase control of signal pathways are roughly 100 times higher than the expected update rate of antenna pointing.

Even after applying these criteria, the final choice of candidate SBC’s was admittedly somewhat arbitrary.

Because only there was only a finite amount of time to be devoted to this study, it was necessary to stop evaluating boards after about 2 months of 50% effort on the part of the author. There may yet exist a better SBC, but we have identified one candidate board that meets our requirements.

Table 1 lists the SBC’s that were chosen along with their salient properties. For comparison, we also include the properties of a few desktop computers employed in the study.

<table>
<thead>
<tr>
<th>Name</th>
<th>CPU</th>
<th>CPU Speed / Bus (MHz / Bits)</th>
<th>RAM/Flash (MB)</th>
<th>Runtime Environ.</th>
<th>Ethernet Adapter Speed (Mbit/s)</th>
<th>Serial Ports</th>
<th>Digital I/O (Bits)</th>
<th>Price</th>
<th>Misc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows Laptop</td>
<td>Pentium III</td>
<td>600 / 32</td>
<td>256 / NA</td>
<td>Java 1.3</td>
<td>100</td>
<td>2</td>
<td>8+</td>
<td>$3000</td>
<td></td>
</tr>
<tr>
<td>Windows Desktop</td>
<td>Pentium III</td>
<td>450 / 32</td>
<td>128 / NA</td>
<td>Java 1.3</td>
<td>100</td>
<td>2</td>
<td>8+</td>
<td>$2000</td>
<td></td>
</tr>
<tr>
<td>Solaris Desktop</td>
<td>Ultra-Sparc</td>
<td>366 / 64</td>
<td>256 / NA</td>
<td>Java 1.3</td>
<td>100</td>
<td>2</td>
<td>8+</td>
<td>$8000</td>
<td></td>
</tr>
<tr>
<td>RCM2100</td>
<td>Rabbit 2000</td>
<td>22 / 8</td>
<td>0.5 / 0.5</td>
<td>Dynamic C</td>
<td>10</td>
<td>2</td>
<td>34</td>
<td>$280</td>
<td></td>
</tr>
<tr>
<td>BL2000</td>
<td>Rabbit 2000</td>
<td>22 / 8</td>
<td>0.125 / 0.25</td>
<td>Dynamic C</td>
<td>10</td>
<td>3</td>
<td>28</td>
<td>$200</td>
<td></td>
</tr>
<tr>
<td>TINI / Step</td>
<td>Dallas Semi. TINI</td>
<td>18 / 8</td>
<td>1 / NA</td>
<td>Interpreted JVM</td>
<td>10</td>
<td>2</td>
<td>16</td>
<td>$130</td>
<td>Has CAN 1-Button 1-Wire</td>
</tr>
<tr>
<td>JStamp Dev. Kit</td>
<td>Ajile aj-80</td>
<td>80 / 8</td>
<td>0.5 / 2</td>
<td>Native JVM</td>
<td>N/A</td>
<td>2</td>
<td>24</td>
<td>$300</td>
<td>No IP! 1-Wire</td>
</tr>
<tr>
<td>Arcom **</td>
<td>Intel 386</td>
<td>25 / 16</td>
<td>2 / 1</td>
<td>RTOS / C</td>
<td>10</td>
<td>3</td>
<td>NA</td>
<td>$300</td>
<td>No JVM</td>
</tr>
<tr>
<td>Model</td>
<td>Manufacturer</td>
<td>Memory /</td>
<td>Clock Rate /</td>
<td>Processor</td>
<td>JVM</td>
<td>Ports</td>
<td>Price</td>
<td>Features</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>--------------</td>
<td>----------</td>
<td>--------------</td>
<td>-----------</td>
<td>-----</td>
<td>-------</td>
<td>-------</td>
<td>-----------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>SaJe</td>
<td>Ajile</td>
<td>100 / 32</td>
<td>1 / 4</td>
<td>Native JVM</td>
<td>10</td>
<td>2</td>
<td>$500</td>
<td>1-Wire</td>
<td></td>
</tr>
<tr>
<td>AJ100-EVB</td>
<td>Ajile</td>
<td>100 / 32</td>
<td>1 / 4</td>
<td>Native JVM</td>
<td>10</td>
<td>2</td>
<td>$1000</td>
<td>LCD, Touch Screen, SPI port</td>
<td></td>
</tr>
<tr>
<td>Axis</td>
<td>Etrax</td>
<td>100 / 32</td>
<td>4 / 8</td>
<td>Java 1.3</td>
<td>10/100</td>
<td>2</td>
<td>$450</td>
<td>Quantity pricing $250.</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: List of computers included in this study.


Communications via Sockets

Except JStamp, the boards considered here are all “internet ready” in the sense that they provide an Ethernet connection and support TCP/IP. However, most of the boards under consideration support only one fairly primitive communications protocol (sockets). Only one of the boards (Axis) supports higher-level protocols such as Java RMI or CORBA. These protocols are usually available only at a higher price point.

Because of their primitive nature, sockets can be difficult to master. This is especially true on controllers that are programmed in C, since there is no standardization of how sockets are invoked or how data is transferred (e.g. byte order). Controllers programmed in a dialect of Java conform (more or less) to the Java standard, and our experience shows that sockets invocation and data transfers are more easily coded and ported between Java platforms than between C platforms.

Description of the Ethernet Tests

All the tests were based on a simple application pair named “client” and “server,” respectively. These applications are run on two computers: server on the SBC and client on a Windows laptop computer, and communicate over a controlled laboratory network. The server application opens a server socket and listens on a well-known port for incoming connections. When the client is invoked, it connects to this port and socket connections are established between them.

After this, a series of message transactions take place between the computers. A single transaction consists of the client sending some data (e.g. a request) followed by a

---

1 This point provides some justification for the recent adoption of JSDA (Java Simple Distributed Architecture) middleware by the ATA software team. Because JSDA runs directly on top of the Sockets layer, the antenna controller can plug in to the main software system in a natural way. By comparison, use of CORBA middleware in the ATA design would introduce an extra layer of software between the antenna controller and the rest of the system.

2 Tests were also run using the Solaris / UltraSparc desktop computer on the client side. The results reported here were not significantly impacted by the choice of client-side computer.
response sent back from the server. Various request / response pairs are implemented. For example, request packets include
  a) a single string of variable length,
  b) a string followed by a pair of floating point values (17 bytes), and
  c) a string followed by a Date object comprising several strings and integers (39 bytes).

The exercise of implementing 3 different value types was quite helpful for assessing the ease of use of the development environments.

Several communication tests constructed on this platform:

Test_A: The socket connection is opened and N message transactions are initiated in rapid succession. As soon as one is finished, the next one is begun. After all transactions are complete, the socket is closed. The time for N transactions is recorded and divided by N for our benchmark.

Test_B: A socket is opened, a single message transaction occurs, then the socket is closed. This cycle is repeated N times in rapid succession and timed as above. The benchmark is defined as the total time for N transactions divided by N.

Test_C: This is a variation on Test_A. Here a single string is sent to the server which echoes it back, repeated N = 100 times. The string length is varied over a wide range (4 – 8192 bytes). Using this test we can deduce the maximal data transfer rate. Here the benchmark is the one-way data transfer rate in Mbit/sec.

The client and server programs were initially developed on the Windows platform and then ported to the SBC’s. Because some of the SBC’s are programmed in Java while others are programmed in C, Windows servers were developed in both languages. (On the desktop, no significant performance difference between the Java and C server was observed.) Servers were then ported to each SBC. The client program was written in Java. As a very crude benchmark of usability, we estimate the time spent from the moment the SBC box was opened until all servers were developed on each SBC. This typically includes substantial time to become familiar with the development tools.

**Ethernet Performance**

As will be seen, the benchmarks from these tests show a great deal of scatter. The most easily interpreted results are those from Test_C. Figure 1 displays the average data transfer rate as a function of string length, L. Most curves are monotonic functions that saturate for large values of L. This is just what we expect, since the overhead associated with opening the socket and initiating each transaction should be fairly constant, and the contribution of this overhead to the transfer rate should decrease with increasing L. Although more boards were tested, we did not find significant differences between boards employing the same chipset, so Figure 1 displays only five representative curves.
The implementation of this benchmark uses the DataInputStream class, which performs at least two copy operations on the string as it is read from the socket. Even with a fast processor (Win-Intel curve) the net transfer rate is not limited by the underlying hardware. Yet this is a meaningful benchmark as it exemplifies a careful but straightforward implementation.

Two of the SBC’s show transfer rates about 10 times slower than that of a Windows desktop (aJ100-EVB and RCM2100). This is reasonable considering the lower processor speeds and network connections.

Two of these curves saturate at surprisingly low values (TINI and Axis) – 100 times slower than the other SBC’s. We have traced this slowdown to the difference between the server running as native code versus interpreted code. The desktop computers employ a Just In Time (JIT) native compiler built in to the JVM. The RCM2100 is programmed in C, and programs are compiled to native in the usual way. The aJ100-EVB uses the aJ-100 processor, for which Java bytecodes are the native instruction set. Comparatively, on both TINI and Axis the Java bytecode is interpreted at runtime (JIT not yet available).³

³ We contacted Axis regarding this interpretation and they confirmed it, pointing out that developers may deploy C programs on this card and achieve significantly better performance.
It is easily shown that the transfer rate is not hardware limited by considering that ftp transfer rates may exceed 16 Mbit/s on Axis.\(^4\) Similarly, on the TINI card the maximum ftp transfer rate is 0.1 Mbit/s, which is still an order of magnitude faster than our benchmark.\(^5\)

For the SBC’s showing higher transfer rates, their associated curves are not smooth. In fact, both curves show a large change in “overhead” for strings exceeding 256 Bytes (RCM2100) or 512 Bytes (aj100-EVB). We speculate that the cause of these overhead changes is associated with the details of the Ethernet driver implementation (specifically Nagle’s algorithm).\(^6\)\(^7\)

The results of Test_A and Test_B are presented in Table 2. Beginning with Test_A, the data show a lot of scatter, and reinforcing our observation that the transfer rate is not easily predicted. The short messages used here give us some idea of the minimum response time for communicating with these boards. With proper tuning, any of the boards can be coaxed into acknowledging a request within 10-56 ms.

<table>
<thead>
<tr>
<th>SBC</th>
<th>Test_A Date (ms)</th>
<th>Test_A Floats (ms)</th>
<th>Test_B Date (ms)</th>
<th>Test_B Floats (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows Desktop</td>
<td>7.3</td>
<td>2.7</td>
<td>14</td>
<td>5</td>
</tr>
<tr>
<td>RCM2100</td>
<td>56</td>
<td>56</td>
<td>3620</td>
<td>3620</td>
</tr>
<tr>
<td>TINI</td>
<td>531</td>
<td>48</td>
<td>8140</td>
<td>940</td>
</tr>
<tr>
<td>aJ100-EVB</td>
<td>370</td>
<td>10</td>
<td>3800</td>
<td>1520</td>
</tr>
<tr>
<td>Axis</td>
<td>102</td>
<td>28</td>
<td>183</td>
<td>62</td>
</tr>
</tbody>
</table>

Table 2: Results of Tests A and B, using either date objects or float objects. Lower scores are better.

Test_B is more of a test of the time it takes to initiate a socket connection to the SBC. We see that socket initiation is expensive on most SBC’s, though the Axis card wins by a large margin. This has implications for program implementation: when communicating with most SBC’s, one should keep the socket open for as long as there is data to transmit – possibly forever.

\(^4\) Ftp is a file transfer program written in C and compiled to native on the Axis and TINI card s.

\(^5\) Because neither RCM2100 nor aj100-EVB has operating systems, comparable ftp transfer rates are not available.

\(^6\) The RCM2100 data may be explained by supposing the driver delays transmission of small packets less than 256 Bytes in length in hopes that more small packets will arrive during the delay (this is one form of Nagle’s algorithm). The results from aj100-EVB might be explained by another aspect of Nagle’s algorithm, where after sending one packet, subsequent packets are delayed unless the recipient returns an acknowledgement packet. If packets have length 512 Bytes, then strings larger than this are broken up, introducing added delays. In any event, packet fragmentation will introduce some overhead.

\(^7\) Note that Java 2 provides socket level control of whether or not Nagle’s algorithm is applied through the Socket.setTcpNoDelay(boolean) method. Throughout this paper, on all Java platforms that support this method, we turned Nagle’s algorithm off for the tests (boolean = true).
Usability

This section presents some highly subjective observations of the author. People with different backgrounds might, for example, find development in C more natural than development in Java and come to different conclusions. With that caveat, here are some observations.

Notice that the Arcom board was not mentioned in the previous section. We were unable to develop using this card due to confusion and bad timing. The board was purchased several months before the beginning of this study, and we believed our purchase included a complete kit (specifically, it included a software package called the hardware development kit). However, when the box was opened, we learned that we were missing a critical piece of software: the software development kit. Contacting Arcom, we learned that the SDK was no longer available for that board, though a new one was due for release later in the year. We did not pursue this vendor beyond this point.

Apart from that, Table 3 presents impressions of usability for four representative SBC’s. Tool Quality is an evaluation of the development tools (compiler, linker, loader, debugger) that may be purchased for the board, merging observations about learning curve and professional quality. Stability indicates impressions about whether the runtime environment is “finished” or still immature (i.e. buggy). Support rates the responsiveness of technical support. Finally, Time to Port is the approximate total work time elapsed between opening the box and completing the server applications.

<table>
<thead>
<tr>
<th>SBC</th>
<th>Tool Quality</th>
<th>Stability</th>
<th>Support</th>
<th>Time to Port (days)</th>
<th>Overall Ease of Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCM2100</td>
<td>B+</td>
<td>A</td>
<td>C</td>
<td>5</td>
<td>C</td>
</tr>
<tr>
<td>TINI</td>
<td>C</td>
<td>C</td>
<td>A</td>
<td>3</td>
<td>C</td>
</tr>
<tr>
<td>aJ100-EVB</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>Axis</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>&lt;1</td>
<td>A</td>
</tr>
</tbody>
</table>

Table 3: Ratings of usability for representative SBC’s. The grading system varies from high (A) to low (F), except Time to Port for which lower scores are better.

Details of Usability

RCM2100

This board has no operating system per se. C programs are developed in a proprietary environment (Dynamic C) that supports language extensions (co-statements, co-functions) for multitasking. All development is performed with a single IDE acting as editor, compiler, linker, and symbolic debugger. The IDE has excellent quality and the runtime system seems quite stable.

The learning curve for the proprietary extensions is steep. This is a paradigm with which we’re unfamiliar and we anticipated that application development would be hindered by its adoption. We also stumbled over byte-ordering issues during implementation of integer and floating-point communication. (Because Java explicitly specifies numeric
representation, this was never an issue on the other SBC’s.) Finally, because of the proprietary nature of the environment, developing on this platform seems risky. If some future requirement forces us to switch to another SBC, porting our code would be difficult.

There is no user group for this device, and while struggling with server development I emailed technical support a couple of times. I finally received a (useless) response more than one week after submitting my request, at which time I had discovered my own solution to my problem.

**TINI**

On the TINI board, one develops Java programs with any editor and uses the Sun compiler in conjunction with a proprietary class library. A command-line program performs static linking. A limited multitasking Unix-like OS runs on the board, and Java programs are executed (interpreted) on a Java Virtual Machine (JVM).

It should be stressed that the TINI platform is by far the least expensive of those considered here. So it is perhaps pardonable that the tools and runtime have an amateurish feel and do not inspire confidence. The technical support is highly responsive, and there is a very active user group. Unfortunately, much support is needed as the runtime system and development environment are still in flux. The learning curve for development is relatively high since there are many, many steps to be performed before one is ready to run the program (reset, load the boot loader, load the OS, boot the OS, compile / link / load the program, …).

**Arcom**

See paragraph under Usability.

**aJ100-EVB**

This board supports no OS. Instead, Java programs enjoy native execution on the CPU. This is unique – the CPU is designed such that the native instruction set is the set of Java bytecodes. Native support for Java real time extensions (www.rtj.org) and hardware implementation of the Java threading model makes it an attractive platform. Multitasking is provided via Java threads and the capability for running multiple JVM’s. Currently, the board supports the Java CLDC API, a limited subset of the Java standard edition. A planned future release will support the CDC API that is more complete. (Proprietary implementations of some classes not present in CLCD but present in CDC are provided.)

Programs are written with any editor and compiled (Sun compiler) against Sun’s CLDC classes and proprietary classes supporting hardware and CDC extensions. Two

---

8 A proprietary subset of the Java standard API is supported.
9 Ordinarily, Java programs are not linked but class resolution occurs at run time. However, run time class resolution incurs a performance penalty at class-load time, and generally requires more memory space for the program. Static linking allows classes to be loaded more efficiently, and also allows optimization – classes and functions that are never called may be elided from the code. Both TINI and aJ100-EVB employ optimizing static linkers.
professional-quality GUI-based programs perform static linking and debugging / loading. These programs and the runtime system give an impression of stability.

After becoming familiar with the tools, program development was straightforward. Technical support is available from the board manufacturer (Ajile), the Ajile user group, another board manufacturer (Systronix) who employs the same chip, and the Systronix user group. All these groups are responsive.

Program development was slowed down by limitations of the CLDC API. For example, socket invocation in CLCD is different from that in the Java standard edition, though many methods in these classes are equivalent. By comparison with porting between dialects of C (RCM2100), porting the servers was a great deal easier in Java.

**Axis**
Axis employs an embedded Linux OS, which comes pre-loaded in ROM. Installation is therefore quite easy. Java programs developed and compiled under Windows execute *without change* on this board. You simply ftp the class files to the board and then run from a terminal window (terminal comes in on Ethernet port).

Regarding ease of use, the Axis board is the clear winner. The box was opened at 8:00 am, and all the servers were running by 2:30 pm. The complete Java standard edition is supported, and development uses any tools with which the developer is familiar. The Axis sales staff was responsive to questions. Axis has a user’s group, but we did not need it.

**Numerical Performance**
The Axis card is very attractive for its ease of use yet its Ethernet performance under Java is disappointing. Supposing that the transfer rate issue might be overcome with added development work (e.g. calling into native sockets routines using the Java Native Interface), we constructed another performance test.

One task our SBC will commonly perform is to interpolate a function evaluated at a series of discrete points. For example, the azimuth and elevation of an astronomical source might be transmitted to the board on a 1 second time interval while the board interpolates to a irregular time interval. We implemented a floating-point cubic spline interpolator in Java and ported this to both the Axis and aJ100-EVB cards. We measured the time required for N interpolations and record the total time divided by N in Table 4.

<table>
<thead>
<tr>
<th>Name</th>
<th>Time for 1 Interpolation (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows Laptop</td>
<td>20</td>
</tr>
<tr>
<td>aJ100-EVB</td>
<td>83</td>
</tr>
<tr>
<td>Axis</td>
<td>4661</td>
</tr>
</tbody>
</table>

*Table 4: Benchmark of numerical processing speed.*
Here we again observe the vast difference between native and interpreted Java execution. To be fair, we emphasize that the Axis card is expected to run at a speed comparable to the aJ100-EVB if the interpolator were rewritten in C.

**Other Comments, Choice of Board**

Beginning with the RCM2100, we note that it is explicitly single-tasking, though it supports a kind of multi-threading.\(^1\) This makes it difficult to employ in our application since we wish to download software updates on the fly. Furthermore, software must be loaded through a proprietary programming port (an Ethernet adapter is available for this port, but requires a second Ethernet connection). Because of this point and concerns cited above (portability, usability) we rejected this SBC for our application.

The TINI card is likewise rejected. Major concerns are the platform stability and the board performance. Specifically, the Ethernet performance is so low as to render the board useless for our application. Note that there is no C compiler for TINI, so using JNI to work around this problem is not an option.

When running Java, the usability of the Axis card makes it very attractive. However, the Java performance is problematic. Both the Ethernet and numerical performance fall below the minimum level required for our application. Moreover, no matter how simple your program, the JVM takes 10-20 seconds to start up! In principle, many performance limits can be overcome using native method calls (or simply writing the application in C or C++) but this complicates development and reduces Axis’ attractiveness. Finally, we rejected this board as well.

Ultimately, we chose the aJ-100 processor chip as the fundamental basis for the antenna controller board. Candidate SBC’s utilizing this chip include the AJ100-EVB and SaJe, with another, JStik, available soon from Systronix. Development for any of these cards is practically identical, so the final choice need not be made immediately. The aJ-100 boards support two concurrent JVM’s. This feature can be used for on the fly downloads of program updates. One JVM negotiates the download and serves as watchdog timer for the system. The second JVM is started from the first and performs most of the work.

aJ-100 computers support two frequency-controllable hardware timers. This provides a convenient way of controlling the azimuth and elevation axis motors at the antenna. These are stepper motors, and require TTL pulses with slowly varying frequency (0-10,000 Hz) as a function of time. Rather than setting up a tight loop to manually control an I/O pin, these timers provide a simple way of driving the motors at constant velocity, allowing leisurely updates of timer frequency in the application code. This feature is not available on any of the other cards discussed here.

aJ-100 computers support the Java threading model at the native level. As a result, thread switching is extremely fast, only ~1 µs. This is fast even compared to Java or even compiled C++ programs on Windows desktops and Sun workstations, where thread

---

\(^1\) Note that RCM2100 supports a hardware watchdog timer providing one aspect of a multi-tasking environment.
switching may take 1 ms or more. Additionally, these boards support the interim Java Real Time specification and promise to update when the final JRT is released. None of the other options considered here do this, giving us added flexibility to employ hard real time processing should that ever become necessary (potentially useful if these boards were used in the IF system).

Finally, we note that porting code from aJ-100 computers to the Java 2 API is quite easy. Indeed, the interpolator was developed firstly on the aJ100-EVB and then ported back to the Windows and Axis platforms. Porting of this program required only changing a few import statements at the tops of the class files and otherwise ran without change. Thus, if future requirements demand switching to a different SBC, difficulties in porting the code will be limited to areas depending on the specifics of board hardware.

**Conclusions**

Our main conclusion is that before committing to a particular SBC, it is very important to prototype your application on real hardware. There were many surprises along the way during this study. For example, we have gotten used to Java performance on desktop machines where JIT runtimes are available and supposed that performance on SBC’s might be predicted simply by scaling the processor size. This calculation turned out to be wrong by two orders of magnitude! In another case, we assumed that C development would be equally difficult on all platforms but learned that some SBC’s (RCM2100, BL2000) implement certain OS functions via proprietary C extensions, greatly complicating program development.

Other surprises included hardware limitations. The JStamp development kit sports an RJ-45 connector and we carelessly concluded that this was an Ethernet connection (not true). On the RCM2100, programs must be loaded through a proprietary hardware port, making software updates via Ethernet impossible.

Regarding the use of Java in an embedded system, the state of the art is changing rapidly. In a year or two, embedded Java will be more widespread and many more options will be available. An important example missing from this study is an SBC based on an Intel 486 or Pentium chip. Our search for SBC’s did not turn up such an option (supporting a flash disk) for less than $1000. Yet an Intel-based system running embedded Linux would automatically support a JIT runtime and / or the GNU gcc Java native compiler. Either of these options would dramatically improve the Java Ethernet and numerical performance benchmarks considered here. Thus an Intel solution might be attractive, though we would not commit to one without testing.

The aJ-100 solution is expected to remain attractive over time. Future support of the Java CDC API will simplify coding and porting, and Java real time extensions may well be crucial if these boards are used in other parts of the ATA system. This is desirable as it simplifies system design. Therefore, choice of aJ-100 boards as the ATA antenna controller is one we are not likely to regret.

---

11 Of course, task switching on these platforms can take advantage of hardware implementation and occur more quickly.