Building a BeagleBone Black Super Cluster

Andreas Josef Reichel

Chapter No.1
"BeagleBone Black System Board"
In this package, you will find:
The author’s biography
A preview chapter from the book, Chapter no.1 "BeagleBone Black System Board"
A synopsis of the book’s content
Information on where to buy this book

About the Author

Andreas Josef Reichel was born in 1982 in Munich, Bavaria, to Josef and Ursula. He went to an elementary school from 1989 to 1993 and continued with lower secondary education for 4 years and started with middle school in 1996. In 1999, he finished school as the best graduate of the year. From 2000 to 2001, he went to Fachoberschule and got his subject-linked university entrance qualification, with which he began to study Physical Technology at the University of Applied Sciences in Munich. After two semesters, he got his preliminary diploma and began with general studies of Physics at the Ludwig Maximilian University of Munich in 2003. In 2011, he completed Dipl.-Phys. (Univ.) in experimental physics with the THz characterization of thin semiconductor films in photonics and optoelectronics. Now, he is working on his dissertation to Dr. rer. nat. on plasma etching processes for semiconductors at the Walter Schottky Institute of the Technische Universität München in Garching.

In his spare time, he has been learning programming languages such as BASIC, Pascal, C/C++, x86 and x64 Assembler, as well as HTML, PHP, JavaScript, and the database system MySQL and has been programming since he was 13 years old. Since 1995, he has been an active hobby musician in different accordion ensembles and orchestras. He also loves to learn about languages and drawing, and he began practicing Chinese martial arts

For More Information:
www.packtpub.com/networking-and-servers/building-beaglebone-black-super-cluster
in 2012. He invests most of his free time in hobby electronic projects and family
genealogical research.

He was the co-author of *Charge carrier relaxation and effective masses in silicon probed
by terahertz spectroscopy*, S. G. Engelbrecht, A. J. Reichel, and R. Kersting, *Journal of
Applied Physics*.

I would like to thank my friends Bruno Lorenz and Stefan Mayr for all the
great fun we had while programming. I'd also like to thank them for their
help when I shared my hardware experiments with them; they were always
open to discussions. I would like to dedicate this work to my daughter,
Maria Sofie.

For More Information:
www.packtpub.com/networking-and-servers/building-beaglebone-black-
super-cluster
Building a BeagleBone Black Super Cluster

In the beginning of the 20th century, a great mind began to think about his interests in technology and art. His name was Konrad Zuse. As a young man, he had been looking for fields where he could use his creativity, but he was somehow disappointed with fixed rules that gave him no space to apply any free thought. Having started with mechanical engineering, he soon switched over to architecture, where he was disappointed again because he could only draw predefined Doric and Ionic columns and could not create something he had in mind himself. So, he switched over to civil and construction engineering. During his studying years, he thought of automatization that could ease real-life tasks, such as automatically working cameras or programmable instruments that could simplify complicated and annoying calculation tasks. He also built the first working vending machine where one could select goods from a dial and retrieve them after inserting coins.

Konrad Zuse (1910-1995), the inventor of the freely programmable binary computer

Finally, he was possessed enough by the idea to invent a binary working computer, which he started to build in the living room of his parents' house, who were not quite amused about this. He was the first engineer who ever used the binary number system invented by Gottfried Wilhelm Leibniz (1646-1716). Based on this, he reinvented two-state logic without knowing that such a thing already existed. Using his ideas, he finally succeeded, after two years, in building the world's first working computer, the Z1, which he finished in 1937. It was a mechanical apparatus that used the motor of a vacuum cleaner, which seemed more practical to him compared to using electronic parts.

For More Information:  
www.packtpub.com/networking-and-servers/building-beaglebone-black-super-cluster
The Z1 was the first binary computer that had an input and output system, a memory, an arithmetical unit, and a program execution unit. Programs could be loaded from paper cards with hole patterns. However, the mechanical parts of the Z1 were not very reliable. Its successor, the Z2, worked with electronic relays and was able to perform floating point operations and possessed a 16-bit memory as well as a 10 Hz system clock.

At this time, computers were huge and filled whole rooms. However, development from electronic relays to electronic tubes soon increased the possible computation speed. The most famous computer that used such tubes was the Electronic Numerical Integrator and Computer (ENIAC). Its power supply had to provide almost 200 kW, which is 500 times more than what is required for standard computers nowadays. Addition or subtraction of a simple number took 200 microseconds, and calculating the root of a number took around a third of second.

Further developments in electronics and the amazing discovery of semiconductors laid down the fundament for the invention of the transistor. The first working bipolar transistor was invented at Bell Laboratories and was presented on December 23, 1947. It was for the first time that it was possible to control electronic currents with voltages in a much more reliable way.

In 1949, Werner Jacobi invented a semiconductor amplifier that used five transistors on one semiconductor substrate. This development was not noticed in the beginning, but it provided the basis for further miniaturization. This approach found more and more popularity from 1958 onward. Robert Noyce invented a fully integrated semiconductor circuit that also included its wire interconnections, which already used photo lithographic processes and diffusion processes for fabrication.

After several improvements in transistor fabrication, such as self-aligned gate structures, the semiconductor company Intel® developed the world's first universally functioning microprocessor for the Japanese calculator company Busicom. This led to the development of the first Central Processing Unit (CPU), which is the Intel 4004, and it was universally applicable. It was first made available on November 15, 1971.

The Intel 4004 microprocessor in a self-built vintage computer

For More Information:
www.packtpub.com/networking-and-servers/building-beaglebone-black-super-cluster
The 4004 ran at a clock speed of maximum 740 kHz and consisted of around 2300 transistors. The execution of one command lasted eight cycles, which was one machine cycle. This led to a maximum throughput of 92500 instructions per second.

However, this was just the beginning. The number of bits the processors were able to work with at once was increased from four to eight, starting with the 8008 microprocessor. Also, speed and supported memory sizes increased dramatically over the first decade. Computers became affordable and are now an integral part of almost everybody’s home.

Modern processors consist of billions of transistors and run at clock speeds of around 4 GHz, which leads to command throughputs of up to 4 billion instructions per second. Memory sizes increased from a few kilobytes to terabytes over the last decades and transistor sizes have shrunken by a factor of thousand from 10 microns to 15 nanometers.

Additionally, miniaturization has led to the possibility of building a fully working computer the size of a credit card with a million times more memory and 50,000 times less power consumption, which is a million times faster than ENIAC.

Combining several of these small computers can further increase their unimaginable calculation capacity, leading to the exciting world of parallel computations and super clusters.

In this book, you will learn about a state-of-the-art model of these very tiny computer boards, that is BeagleBone Black, which resembles a full computer system. Its abilities are far beyond what anybody could dream of 50 years ago. You will learn how to integrate several of these small boards into a fully working super computer cluster, giving you the possibility to freely scale up its calculation power.

Practical examples will demonstrate what this immense power is really useful for nowadays.

**What This Book Covers**

*Chapter 1, BeagleBone Black System Board*, introduces you to the hardware and explains the handling and working principle of the BeagleBone Black system board. A very basic introduction to generic software programming is also provided.

*Chapter 2, Building a Beowulf Cluster*, shows you a step-by-step guide in order to build a super computer with only minimal hardware interconnecting several boards. The network interconnection and topology is explained in detail.

*Chapter 3, Operating System Setup and Configuration*, shows you exactly how to install the Ubuntu operating system used throughout this book. A step-by-step guide helps you configure the master and slave nodes, SSH, as well as basic developer tools. A simple method of data transfer between the cluster and an external computer is also shown.
Chapter 4, Parallel Computing with OpenMPI and ScaLAPACK, features the introduction and installation of the OpenMPI messaging system. This chapter provides the basis for scaling calculations of linear mathematical problems on the cluster nodes. This is shown with the popular library ScaLAPACK.

Chapter 5, Advanced Solving of General Equation Systems, enhances the previous chapter with advanced software libraries such as PETSc for linear and nonlinear problems as well as SLEPc for Eigenvalue problems. Example programs show you how to solve such equations on your cluster.

Chapter 6, Scientific and Technological Examples of Parallel Computing, shows you the final step in order to use deal.II, the highly sophisticated scientific finite element library for simulations and arbitrary dimensions. Example programs show you how to take your first steps in order to use this huge library to solve difficult equations with a state-of-the-art approach.

Appendix, References, gives a list of important links for various software and examples, which provide additional reference.

While studying physics and natural science, I always felt that solving scientific or mathematical problems was the only way of really making use of a modern computer's speed and calculation power. Such problems appear in research, simulation, or visualization purposes, as well as in computer games. Playing around with computers and programming since the age of 13, I soon realized that linear problems such as equation systems can take a very long time to get solved on modern standard PCs, depending on the number of unknowns. Also, being interested in the miniaturization of hardware, I finally came across **BeagleBone Black (BBB)**, which is a credit-card-sized computer that can run Linux. I soon stumbled across some already realized supercomputer projects on the Web that utilize similar boards. Having read some articles about their scalability and capabilities, I decided to try and understand how to build and configure such a supercomputer with the more modern BBBs instead of Raspberry Pi. I succeeded in building my first self-built, low-cost supercomputer that is a **Beowulf** cluster, and now I will show you how to build your own. The common reason why people buy BBB is because of its fascinating hardware. There are a lot of other embedded systems such as Arduino Mega or Raspberry Pi, which were created in order to enable hobby programmers to start the development of their own hardware controls or other applications right away. Compared to Arduino Mega or other low-level products, BBB has 100 times or more computational power and a lot of other integrated features for the same price.

The technical specifications of BBB are introduced in this chapter. Alongside the basic hardware architecture and board features, you will get to know other useful information on the boot selection button and internal storage partitions. Coming to the operating system (OS) and software parts, the main focus will be on the OS used throughout this book and a basic understanding of programming languages and the development tools for later chapters.

For More Information:
The following topics will be covered:

- Explanation of the system board features
- Introduction to existing operating systems
- Understanding the partition structure with and without a microSD card
- Explanation of boot partition and boot failure recovery
- Introduction to the necessary programming environment for this book

The last point is especially written for hobbyists who might or might not have basic programming skills, as all the steps required to create your own software will be explained and kept at a basic level.

Introducing the hardware

The BBB board is a complete, low-cost, energy-efficient, multipurpose development system with onboard Ethernet, flash storage, video controller, and much more. Its primary goal is to offer a true open hardware and community-supported embedded computer for developers and hobbyists. Compared to low-level embedded systems such as Arduino, BBB is a fully functional standalone PC that has the size of a credit card.

Without any expansion cards, the power required by a BBB is around 2.5 watts, which is 5 percent of the power required by a typical light bulb. Compared to a common Pentium III computer with the same clock frequency of 1 GHz, it only needs about 2.5 percent of its power requirement, and even has a better graphics card onboard, but has slightly less memory.

The central processing unit

In general, the power of a computer is defined by the speed of its components. Most critical are the main memory, its bus interconnection, and the central processing unit (CPU). The slowest interconnecting path between faster components is called the bottleneck and defines the overall system's performance. On modern CPUs, the memory interface is on-die, which means that it is integrated into the silicon chip. This offers maximum performance and avoids any bottlenecks between the CPU and the main memory. This is also the case with BBB.

For More Information:
www.packtpub.com/networking-and-servers/building-beaglebone-black-super-cluster
BBB’s heart is the Sitara™ ARM® Cortex-A8 32-bit processor from Texas Instruments with a programmable clock driven at 1 GHz by factory settings. Interestingly, on some of my boards, the XAM3359AZCZ100 version of the CPU is used, which has a maximum frequency of 800 MHz according to TI. Despite this, it is driven at 1000 MHz on BBB without any problems. According to the manual, it was changed to XAM3358 in the board revision C. The architecture is RISC, which means that the CPU only needs one clock cycle to execute a command. This makes the Cortex-A8 faster than a typical Intel P3 with the same clock frequency if one focuses on standard instructions. Also, the CPU supports the NEON instruction set, which gives BBB a great advantage over Raspberry Pi.

Under full processor utilization, the board gets quite warm. This is not a problem if it is used as a standalone board. However, if there is more than one board in a small area or even if the boards are stacked on top of each other, they will need some cooling in order to avoid overheating. An example of a cheap cooling system is given in Chapter 2, Building a Beowulf Cluster.

The figure in the next section shows us the internal CPU architecture that also provides very sophisticated features such as a touchscreen controller and a 3D graphics acceleration on-die.

I/O interfaces and control buttons

The actual purpose of BBB was to provide a development platform that is easy to program and can be used to control or regulate other hardware. In other words, it must be able to measure and output signals that are easy to interface in the hardware and software. To provide this capability, two 46-pin headers are available for general-purpose I/O (GPIO). These pins use a 3.3 volt logic for I/O. A higher voltage level might result in damaging the CPU. The board also provides two programmable real-time units (PRUs) for time-critical applications. Please refer to the hardware manual for a detailed description. In this book, no I/O operation is described, as they are not used in the BBB cluster.

Besides general-purpose I/O, the board also provides the following:

- A RJ45 network jack
- A 5V power jack
- A USB client
- A USB host port
- A HDMI jack for video and audio support
- A serial port header that is useable for debugging

The system can be powered by either the 5V power jack or the USB client port using the preinstalled operating system. However, using the operating system as described in this book, we can see that only the 5V power jack will work. This means that the boards in the final cluster configuration cannot be powered up using a USB power supply. Please refer to Chapter 2, Building a Beowulf Cluster, for information on how to build a cheap and efficient power supply. This book only describes the usage of the RJ45 network interface for installation, configuration, and user control via SSH. Thus, it is not necessary to make use of the HDMI port in cluster applications. The onboard network controller works with 10/100 megabit/s.

The internal architecture of BBB's CPU

For More Information:
www.packtpub.com/networking-and-servers/building-beaglebone-black-super-cluster
The following image and list gives you an overview of the important onboard ports, plugs, and control buttons; a indicates a 5V power jack, b indicates an RJ45 Ethernet port, c indicates a general-purpose I/O with serial header on its right, d indicates a USB host port, e indicates status LEDs, f indicates the reset button, g indicates the power button, h indicates the CPU, i indicates the boot-selection button, j indicates a USB client port, k indicates the microSD slot, and l indicates the mini HDMI port:

While the power and reset buttons are used in the same manner as they are used on PCs, the boot-selection button is used to select the boot sector located in either the internal ROM or on the external microSD card. This function is very useful in order to recover a misconfigured or non-bootable system.

**The onboard memory and flash storage**

BBB is equipped with 512 MB of DDR3 RAM, which possesses a higher bandwidth compared to the competitor, which is Raspberry Pi, which uses DDR2 RAM. It is not possible to extend the physical RAM on a single BBB; however, this is not important when performing cloud computing because this makes use of distributed memory.
For nonvolatile storage, 2 or 4 GB of 8-bit eMMC flash memory—depending on your board revision—is available onboard. This is enough to install all the required software along with the operating system in order to perform the highly sophisticated computations described in this book. Any dynamic libraries will be installed in a common network shared folder. The storage capacity can also be extended using the free microSD card slot that provides more virtual memory or additional storage capacity. You should refer to the documentation of your specific board version to see supported microSD card sizes. In this book, an additional card with 16 GB size will be used for the master-node board, which will be explained in Chapter 3, Operating System Setup and Configuration.

The storage memory partition structure

Let’s first explain what partitions are and why they are used. Partitions are logical divisions of storage space divided into multiple logical units, providing a convenient way of storage management. Each partition can have its own filesystem, and thus, it can be formatted separately. Also, for virtual memory a specific type of partition, a so-called swap partition can be used. Every block device, which means every memory device with random access and consisting of discrete blocks such as sectors, can be partitioned.

Each operating system usually has a boot partition where important system files that are in charge of starting up the system are stored. A system can have more than one operating system where a boot menu can provide the possibility of selecting a specific boot partition on each system startup.

It is very important to understand the partition structure of the BBB storage memory in order to know how to install alternative operating systems. By default, there is no extension microSD card installed, and the internal flash memory is divided into two partitions. If you boot up the preinstalled operating system from the internal memory, there will be two partitions for the internal and two partitions for the optional uninstalled microSD card. Furthermore, there will be two virtual partitions that represent the boot loader for the internal and external memory. The latter two are permanent and cannot be accidentally overwritten. The kernel of the preinstalled Linux version, like any other version, will map the storage partitions in its local filesystem to the /dev directory.
The following table shows the existing partitions if they are booted from the internal flash memory:

<table>
<thead>
<tr>
<th>Partition</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/mmcblk0p1</td>
<td>The first block device and the first partition</td>
</tr>
<tr>
<td>/dev/mmcblk0p2</td>
<td>The first block device and the second partition</td>
</tr>
<tr>
<td>/dev/mmcblk1p1</td>
<td>The second block device and the first partition</td>
</tr>
<tr>
<td>/dev/mmcblk1p2</td>
<td>The second block device and second partition</td>
</tr>
<tr>
<td>/dev/mmcblk0boot0</td>
<td>The boot code partition 1</td>
</tr>
<tr>
<td>/dev/mmcblk0boot1</td>
<td>The boot code partition 2</td>
</tr>
</tbody>
</table>

The mmcblk0p1 and mmcblk0p2 partitions, respectively, relate to the first and second partitions of the first block device (the number zero), whereas the mmcblk1p1 and mmcblk1p2 partitions relate to the second block device (the number one). The order of the block devices changes if an external microSD card is installed and booted.

If there is no external microSD card installed, the internal memory is /dev/mmcblk0; if it is installed, the internal memory is /dev/mmcblk1 instead.

**The boot-selection button**

The boot-selection button enables the user to select the boot code. If it is pressed during the system startup, the mmcblkXboot1 partition is selected, which will boot up the first partition of the microSD card. If it is not pressed, the system will load the internal boot partition code using mmcblkXboot0. The letter X means that boot1 and boot0 can be mapped onto either the first or second block device, which is irrelevant. The preinstalled boot sector will try to load the operating system on any inserted microSD card on its own. This will only work if the kernel located there is compatible. If it is not, the system will not boot up and will hang. This means that if you insert a microSD card with an alternative operating system for the first time, you have to push the boot-selection button until you see the system starting up (indicated by the status LEDs flickering). This only works if you cycle the board power, which means that you need a cold start and should not use the reset button.

This will be explained in Chapter 3, *Operating System Setup and Configuration*, which talks about how to install another Linux operating system and work with system partitions and filesystems in detail.
Recovering a boot failure

Whenever you encounter a boot problem and cannot start up your BBB with the operating system on the internal eMMC flash, you can use a failsafe microSD card that you set up earlier in order to boot up the system. For this purpose, you just have to insert the microSD card into the slot and push the boot-selection button before you power on the system and hold it until it boots. It is always a good idea to keep a working system image on a microSD card for such purposes.

Operating systems

BBB up to version A5C and built before May 2014 comes with a preinstalled operating system, namely the **Angstrom** Linux distribution. Later versions come with the Debian Linux distribution. Texas Instruments provides two releases: an Android and a Linux operating system. All the examples in this book are based on the Ubuntu Linux distribution; however, it is a question of personal taste which Linux distribution you want to use. If you decide to use another distribution, you might have to figure out how to compile your software and install the required packages on your own. There is, however, a chance that packages used throughout this book for Ubuntu might also work with other Linux distributions. Android does not support general Linux software and cannot be used to build a cluster with the means described in this book.

ARMhf images

Linux operating systems are available for a vast variety of different computer platforms such as PowerPC, x86, IA-64, ARM, and many more. An interesting feature of the CPU used on BBBS is the implementation of floating point instructions. This means that mathematical operations based on non-integer values can be executed by hardware rather than software, and thus they are carried out much faster. **ARMhf** stands for **ARM hard float** architecture. To make use of this advantage, a special operating system image is used throughout this book, namely the **Ubuntu-12.04-armhf** image from John Clark.
The Ubuntu 12.04 ARMhf Linux system

The Ubuntu 12.04 Linux distribution, compiled for the ARMhf platform, can be obtained from the home page of John Clark's website at http://www.armhf.com/download/. Please keep in mind that Linux systems and software are updated a lot. If you are not able to get exactly the same software as that used in this book, you can try a newer one or download the version from the download section accompanying this book. You will find more details on how to download and install the operating system in Chapter 2, Building a Beowulf Cluster.

Software programming

The most important part of a good computer is good software. Without good software, specifically optimized for its hardware, the full computational power cannot be utilized. In this book, I will show you how to build a supercomputer cluster that gains its high-speed computational power from distributing certain tasks to other its via networking. For this purpose, special software is required and has to be compiled from the source code. How this works and what nodes are will be explained in Chapter 2, Building a Beowulf Cluster, and Chapter 3, Operating System Setup and Configuration.

The open source philosophy

Although there are a lot of already existing helpful software packages, it is very important to understand that Linux is an open source operating system written for an open source community. Usually, Windows users are frustrated when they gain first contact with open source software, because they are used to having already working and easy-to-install software. A huge disadvantage is that these software packets are compiled for a standard platform and might not be optimized to a specific computer that they are installed on. Another problem is that if software components of the operating system are updated but older versions are required by the user software, instabilities might arise or completely different interfaces might disrupt the software functionality completely.
Software modularity and dependencies

Linux is a highly modular operating system. The whole system is built on the philosophy of open software, which means that every part of the operating system can be compiled from available open source code. This source code is then compiled by standard programming languages such as C, C++, FORTRAN, Assembler, and others in order to build binary code specifically optimized for certain hardware. The technique by which software is built does not differ much from Windows or Linux operating systems. For the beginner, it might be hard to produce a working compiled program starting from source code because usually, there are a lot of software dependencies such as missing software libraries or other programs that code is based upon. In this case, it might be hard to find all the required libraries, especially when newer versions that have changed in interfaces such as function definitions are available. On the other hand, a dependency can soon lead to several others so that the search for all the required libraries grows exponentially and takes a lot of time.

Also, on certain hardware, some well-established compiling parameters do not work and have to be modified or bug fixes have to be found. This can make the simple task of "just compiling software" an unsolvable problem for beginners. Thanks to the rising community of hobby programmers and Linux enthusiasts, there are a lot of forums online that can be searched for such problems. Often, solutions are present, and if not, there can be hints that point us in the right direction, at least.

The following sections will explain the basics of creating software on Linux operating systems with standard programming environments. It is written for hobby enthusiasts who might or might not have already tried and programmed their own software. Although existing knowledge is very helpful, it is not required in order to understand the following explanations.

The source code and programming languages

Each computer program consists of binary code, which means a sequence of two states usually described as zero and one. A specific state is called a bit. Four of these bits make up a so-called nibble and eight make up a byte. Several bytes can be described as a word, a double word, or a quad word. The following table gives you a small summary of the most important data sizes:

<table>
<thead>
<tr>
<th>Amount of bits</th>
<th>Amount of bytes</th>
<th>Special name</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1/2</td>
<td>nibble</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>byte</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>word</td>
</tr>
<tr>
<td>32</td>
<td>4</td>
<td>double word</td>
</tr>
<tr>
<td>64</td>
<td>8</td>
<td>quad word</td>
</tr>
</tbody>
</table>

For More Information:
www.packtpub.com/networking-and-servers/building-beaglebone-black-super-cluster
A central processor does nothing else except interpreting bit sequences as commands. These commands are called instructions and tell the CPU what to do. This is the lowest level of programming, which is the so-called machine language. Machine language is, except to certain freaky people, not human-readable. For example, it is not obvious that the binary code 1011 0100 0100 1100 1100 1101 0010 0001 is the end of an MS-DOS program.

Low-level programming

Low-level programming means the direct programming of machine language. Of course, this has to happen in a human-readable way. One possibility to simplify 1011 0100 0100 1100 is given by using another number system, such as the hexadecimal system, resulting in 0xB4 0x4C. This is better, but it's still not readable by humans. The final simplification is the invention of so-called mnemonics. For example, on Intel x86-platforms, 0xB4 0x4C would mean \texttt{mov ah, 0x4C} in this mnemonics language. Now, one can understand that this code sets the CPU register named \texttt{ah} to the value of \texttt{0x4C}. This language as well as the software that translates this back into bits is called Assembler.

Assembler has its advantages and disadvantages. One big advantage is that the resulting software does exactly what you programmed. This means that there is no optimization that modifies your code and you can program very effectively in size and speed. One big disadvantage, however, is the problem that each CPU has its own instruction set. This means that our Sitara CPU will not understand the preceding example, because it has no \texttt{ah} register. For any real problems we want to solve using computers, we are primarily interested in the nature of the problem and not the nature of the CPU used in the computer. To make programming independent of the used computer platform, there exist so-called high-level programming languages.

High-level programming

High-level programming languages consist of keywords, syntax, and grammar, as with every spoken language. The keywords define the vocabulary that can be used, whereas syntax and grammar define the exact utilization and order of these keywords. To understand this, we should have a look at how a simple loop will look in a low-level language compared to a high-level language.

For More Information:  
www.packtpub.com/networking-and-servers/building-beaglebone-black-super-cluster
The following example shows us that a simple loop already needs four different instructions in Assembler, while it can be realized by a relatively simple for keyword in the high-level language C++:

<table>
<thead>
<tr>
<th>Low level (Assembler)</th>
<th>High level (C++)</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov cx, 0</td>
<td>for (int j = 0; j &lt; 345; j++) {</td>
</tr>
<tr>
<td>@mark1:</td>
<td>other code...</td>
</tr>
<tr>
<td>other code..</td>
<td>}</td>
</tr>
<tr>
<td>inc cx</td>
<td></td>
</tr>
<tr>
<td>cmp cx, 345</td>
<td></td>
</tr>
<tr>
<td>jne @mark1</td>
<td></td>
</tr>
</tbody>
</table>

A low-level language compared to a high-level language

While C++ is a general-purpose high-level programming language, there are also languages that are more specifically optimized. One example is FORTRAN, which is mainly used for mathematical problems due to its ability to define matrices and other mathematical structures very easily.

**The compiler toolchain**

Once the required code has been written and is ready to be translated from its human-readable form into machine language, there is a certain sequence of tools that have to be used. This toolset is called the compiler toolchain:

1. Firstly, the code is treated by the compiler itself. The compiler translates the high-level language to a low-level language, mostly Assembler.
2. It is then translated to object files. Usually, these two processes are performed by only one compiler internally.
3. The object files then have the binary format and can be executed theoretically. However, the OS must know a few things in order to execute programs correctly. It must be told where in the main memory the program has to be loaded, how much memory it uses, which libraries it needs, and so on. To fulfill these requirements, we need to link the program.

For More Information:
www.packtpub.com/networking-and-servers/building-beaglebone-black-super-cluster
4. The so-called linker combines one or more object files and adds information that's specific to the OS in use.

5. The final program then consists of a special **executable format** generated by the linker and incorporating the object files produced by the compiler. In Linux, this executable format is called **Executable and Linking Format (ELF)**.

The whole process is depicted in the following diagram:

![Software compilation and simple toolchain](image)

Another important feature of the linker is its capability to embed the required libraries into the executable file. This is called **static linking**. The program can then be used on other computers that do not have that specific library installed. The opposite of static linking is **dynamic linking**. In this case, only a stub of the library is linked into the program that tells the OS which library to provide. Dynamically-linked programs are smaller in size but always need their libraries.

In all examples, the main focus will be on C++. Some of the modules are only available as FORTRAN code; however, once compiled, their functions can also be accessed from C++ programs.
Summary

In this chapter, BeagleBone Black was introduced, starting with its hardware and key features. It was compared with other platforms regarding its power and energy requirements. Special attention was paid to the internal architecture of the central processing unit, the I/O-interfaces, and control buttons, as well as the embedded memory features.

For the following system setup, the internal flash partition system was described and special caution was taken on the variable partition mapping regarding an optional microSD card expansion.

The default boot mechanism and how it can be altered using the onboard boot selection button was explained. It was also mentioned how to unbrick the system if it fails to boot.

A short description of available operating systems was given as well as an explanation of the importance of native floating point support. The chapter ended with a short introduction to the basics of software programming, keeping an eye on the basic functionality of compilers and linkers. The difference between static and dynamic linking was explained, which will be of further importance in the next chapter. Missing elements such as data types will be explained where necessary.

In the next chapter, we will cover practical things. I will show you how to connect the boards, build a simple frame, and how to get a cheap power supply. You will also be introduced to the basic network structure and the general idea of a Beowulf cluster’s topology.
Where to buy this book


Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.