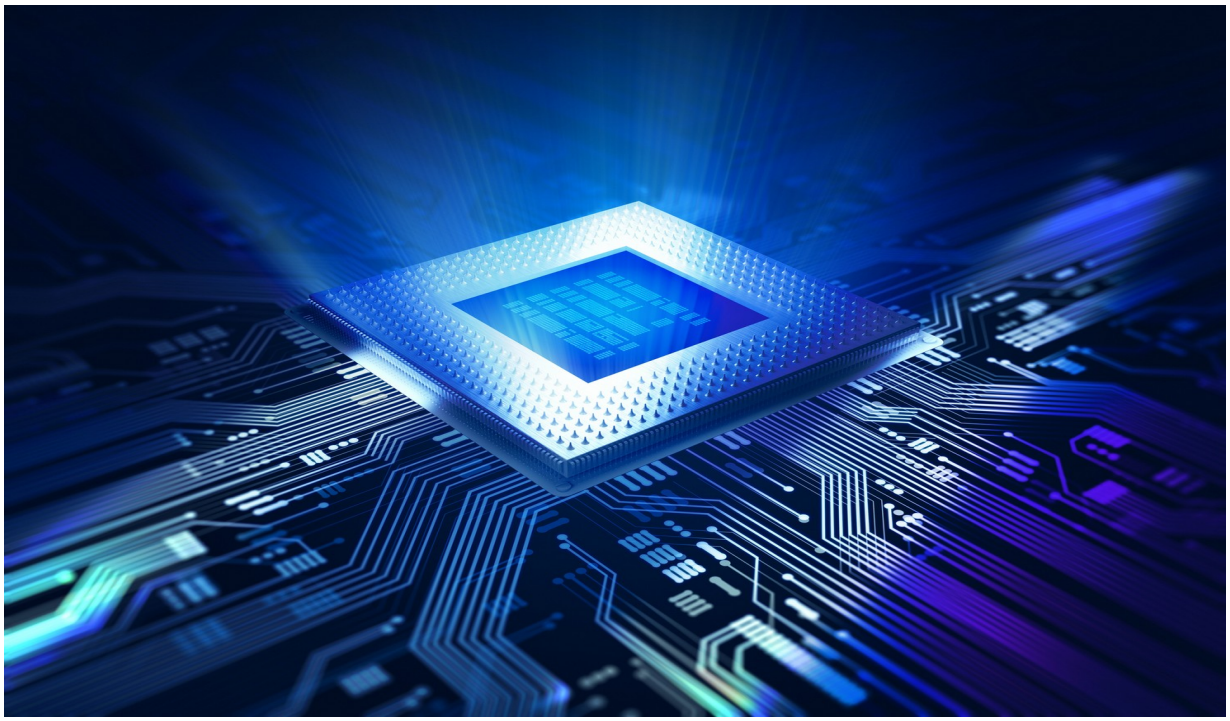
AimValley

White Papers
AimOS Generic



AimValley
Utrechtseweg 38
1213 TV Hilversum
The Netherlands

phone +31 35 689 1900
sales@aimvalley.com
aimvalley.com/aimos

AimValley

White Papers
AimOS Generic

Table of Contents

AimValley

Utrechtseweg 38
1213 TV Hilversum
The Netherlands

phone +31 35 689 1900
sales@aimvalley.com
aimvalley.com/aimos

1. Introduction

AimValley is a company that has its roots in the telecommunication industry where reliability and availability of service is a major requirement. The systems built are required to provide an uninterrupted service for 15 years or more, including maintenance and upgrade windows with zero down time.
Zero down time requires reliable software that does not break. However, robustness in software alone does not bring zero down time, it requires a system architecture that supports that goal.

AimValley OS includes support for the following concepts which are often used in high availability system architectures:

- Dual image support that allows for a secure upgrade of the software.
  - A secure upgrade entails that after a software upgrade the system is never isolated, bricked or otherwise unmanageable. A secure upgrade requires hardware facilities and software that support active, standby, soak and auto fallback procedures during the installation of new software.
  - Dual image support and a solid update process, solves only half of the problem.
    The configuration database needs to be run-time upgraded to be compatible with the new functionalities in the next release. Besides this, if the system falls-back to the old release (manually forced or automatically), the old database needs to be restored without any service interruption when the old software load is started.
- The above described dual image support, does not work unless the system is capable of rebooting the software without service interruption. This entails that the software needs to inspect the current hardware and ASICs' settings cautiously, before writing to any registers that will cause a service interruption. It also means that relevant protocol states need to be saved before any spontaneous or intended reboot of the system, to prevent any unwanted network reconfiguration that can cause a service interruption.
- Independent control and protocol plane protection.
  - Every control and data plane task is independently, redundantly executed in an active and protective state. The protecting task is in the same state and has the same configuration as the active task, so that it can presume control without a glitch. Note that a fully protected system requires hardware that supports communication paths to both controllers from the controlled units or systems.
- Dynamic hardware configuration management.
  - Configuration of the hardware and modification of the hardware configuration, (adding and removing cards) can be done live, without impacting the running services.
- Software package handling.
  - The software is provided in one single package, including boot, file system, kernel, drivers and applications to secure that the combination works in perfect harmony. Even in systems that consist of many units, including chassis with multiple cards, the software load is delivered in one package.
  - Add signature info to guarantee the right software is loaded.

Utrechtseweg 38
1213 TV Hilversum
The Netherlands

phone +31 35 689 1900
sales@aimvalley.com
aimvalley.com/aimos

Apart from concrete system architecture, the software needs to be of high quality and should be capable of running over years without problem. This requires a long and expensive development process with many iterations of architecture discussions, design reviews, implementations and test/verification steps to build such a system. This lengthy process can be accelerated by using a platform that enforces proven concepts and patterns, that provides the right tooling. The right platform supports developers in their day-to-day work, by reducing labor intensive and repetitive tasks, thereby allowing them to focus on functionalities that make the product standout.

At AimValley, we have been using AimOS for many years and have improved and complemented the platform at each development cycle, up to a point where we can repeatedly deliver high quality systems with a proven low field return rate, ranging from very small and relatively simple systems to very large and complex ones. The key to this success is a strong architecture, flexible configuration management components, modular protocol and services support, a portable and efficient infrastructure, a hardware abstraction layer and build environment which eases the development process.

2. In a nut shell

AimOS is a collection of software assets and tools to develop portable, extendable and highly reliable embedded systems.

- Portable through a hardware and operating system abstraction layer.
- Extendable through formal defined interfaces that enable integration of off-the-shelf components and third party software.
- Reliable through platform enforced design patterns, the use of Design Specific Languages and code generation tools, static and dynamic code analyzers, automated tests and a strong process driven traceable development process.

AimOS is the development platform that AimValley uses to develop its own products.
It is a field proven platform with a first solution deployment in 2003 and has been used in over 100 000 systems. It is constantly being innovated and improved on, at every development cycle.

The key difference with similar platforms is that AimOS was not build as a product, but specifically developed as a platform to build products with.

AimValley delivers AimOS as a customized product development environment and it is designed for and verified on the customer hardware which ultimately results in a better fit and better performance. Enhancements, updates and bug fixes are delivered to the customer's platform and not in a generic new release that needs additional adaptation to the customer's product.

AimValley
Utrechtseweg 38
1213 TV Hilversum
The Netherlands

phone +31 35 689 1900
sales@aimvalley.com
aimvalley.com/aimos

3. Architecture

The objective of the architecture is to provide a platform that produces consistently and largely independently of the developer, an application that is predictable, has prioritized behavior, is efficient in resource usage, is highly portable and has a formalized interface to support third party applications.

3.1 Predictability

Predictability or deterministic behavior is achieved by minimizing dynamic resource allocation as much as possible. AimOS organizes the tasks, message definitions and system resource allocations from a central place. The advantage of this approach is, that the messages sent or received are defined beforehand which allows accurate scaling of the buffer pools and it supports the automatic generation of the message handling routines during build time. The latter also reduces the chance of out-of-sync message definitions and handling routines in sender and receiver tasks.

3.2 Prioritization

Predictability and prioritized behavior is further enforced by defining the tasks, task resources and priority settings in a centralized location. The benefits are twofold, it provides a clear overview of all the tasks and priorities in a system and secondly, message routing can be defined at compile time.
Task priority is further enforced as message routing and delivery is done in the context of the sending task, ensuring the right priority even during message routing.

3.3 Efficiency

Being efficient in using the available resources in an embedded system is key for the success of a product. AimOS uses a smart buffer concept that prevents memory thrashing and unnecessary copying of data between tasks. For instance, messages that are sent inside the same instance of AimOS, are not copied but result in mere exchange of ownership of the involved buffer.

3.4 Portability

The AimOS infrastructure provides a hardware and operating system abstraction layer for any real-time application. The hardware abstraction layer focuses on the controller related assets and the ASICs that are being used.

The first, controller infrastructure abstraction, provides an interface for most operating system dependent functions and relies on the POSIX compliance of the underlying operating system.
AimOS has been ported and tested on many POSIX compliant operating systems such as Linux, Solaris, Chorus and OSE. The controller infrastructure component provides tasks, messages, timers, semaphores, message routing facilities and other functions for the application.

Utrechtseweg 38
1213 TV Hilversum
The Netherlands

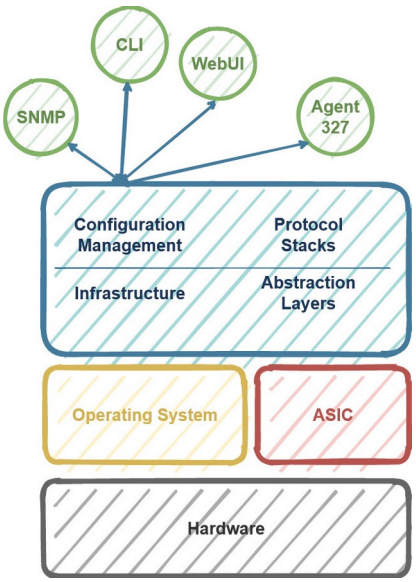
phone +31 35 689 1900
sales@aimvalley.com
aimvalley.com/aimos

The second, ASIC abstraction, is divided in two parts. A generated low level driver that is ASIC dependent and a high level driver that provides a stable and ASIC independent interface to the rest of the application. The low level driver is generated from a manually created register map. The register map description is also used at AimValley to generate register descriptions that are used during FPGA or ASIC design. This approach reduces the chance of inconsistencies between the register definitions and the driver and at the same time reduces the driver development effort when there is a small change in register layout. Only the low level driver needs to be regenerated from the new register map definition.

4. Design

AimOS captures the characteristics described in paragraph 3, in a few high level building blocks.

1. Configuration Management
2. Infrastructure
3. Hardware Abstraction
4. Protocol Stack
5. Service Integration
6. Build environment



4.1 Configuration Management

AimOS provides its service to the user by means of the AimValley Logical Interface (ALI). The ALI provides an ASCII oriented interface for the management agents. The interface supports a set of commands as defined in the ALI XML based model. The ALI model is used to generate the database model and the interface handling routines. ALI commands given by the management agents are first subjected to syntax checks. ALI takes care of the inter-process communication by means of Linux socket communication.

The configuration manager is the central point where ALI commands are interpreted and handled. The configuration settings are maintained in a database with access restricted to the configuration manager. Configuration commands from the management agents are subjected to validations (semantic checks) and passed further into the system. The configuration manager can perform checks/validations with the current status to verify the consistency of the provisioning. From the ALI model, hook-functions are generated as a template to implement the semantic validations.

Utrechtseweg 38
1213 TV Hilversum
The Netherlands

phone +31 35 689 1900
sales@aimvalley.com
aimvalley.com/aimos

Fault Management System

The Fault management system correlates the events or defects from the entire system to deduct the correct alarm condition. This condition is stabilized over a configurable integration period before reporting this to the user application or using this in any other way within AimOS. The contribution of defects to a specific alarm is described in terms of logical expressions. Per alarm the user application can provision the alarm to be reported. History alarms, event logs and other logs are stored in the database and can be retrieved by the management agents.

4.2 Infrastructure

The AimOS uses the POSIX interface definition that is present in most modern operating systems to claim portability across operating systems. The infrastructure provides a common interface to normal operating system functions that glues tasks, timers, buffers, messages, etc. together into an efficient framework. The framework also provides customization for those operating systems that do not support the POSIX standard fully, which makes AimOS portable across many operating systems.

Messages

AimOS provides a platform supporting efficient message based concepts with generalized functions for easy portability. The individual components provide their services in a multi-task environment and interact by means of messages and signals. Tasks within AimOS are blocked until a message request for a certain action is received. On a single board system AimOS is normally embedded within one process and the tasks run in a shared memory space and data is not copied when a message is exchanged between components. When AimOS is running in a distributed mode that requires multiple instances of AimOS running on multiple cards or virtual environments, the messages and data are copied when routed to remote instances.

Tasks

The infrastructure layer supports the concept of static and dynamic tasks. The former ones are created during startup and never deleted, the latter ones are created or deleted at run-time. When a new task is created, it is automatically provided with a message queue and the message router is informed about the task's existence to make the task reachable from other tasks.

Configuration messages are often send to multiple tasks that require the same information.
AimOS utilizes a message broker for this purpose, where a task can be registered as a recipient of a specific message. When the originator sends the message, the broker will distribute the messages to all the registered recipients. This concept allows the originator of a message to send it without any knowledge about the recipient.

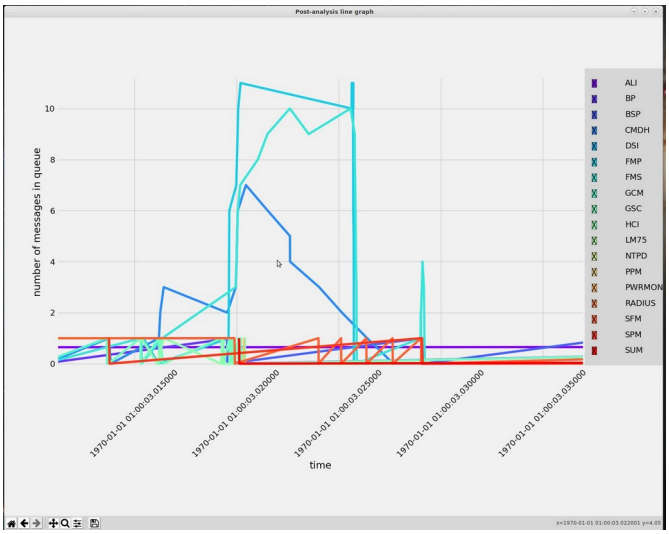
Utrechtseweg 38
1213 TV Hilversum
The Netherlands

phone +31 35 689 1900
sales@aimvalley.com
aimvalley.com/aimos

In a system which contains slots for multiple cards, newly inserted cards are often initialized to a non-operating mode where very few tasks are initialized and running. The AimOS routing system provides pre-delivery hooks per task that are called upon before messages and signals are put in a task's message queue. These hooks can be used to various purposes which includes the ability to create and start a task when it is not running by simply sending a message to the non existing task.

Debug Facilities

The debug facilities that a platform provides, can have a huge impact on the development time and turn around time of customer tickets. AimOS has a dedicated debug environment similar to a command line interface, which provides access to the internal data structures and functions. The debug environment includes a prioritized trace and log facility and a task queue monitoring facility. The latter sends queue utilization reports to a remote system that stores the accumulated data from all tasks in a database for later evaluation. This helps when profiling the application and spotting bottlenecks and priority issues.



Debug commands are organized in debug objects that are stored in a directory-like structure, similar to the files in a file system. Debug objects are accessible via an interpreter that processes the user input, locates the object in the directory tree and executes the command. The interpreter uses connectors that are bound to a particular IO service (console, sockets, telnet etc). The connectors and the physical implementation of the connectors are participating in a so called Bridge design pattern that makes it possible to run-time switch connectors and IO-devices.

Scalability

The configuration management, infrastructure and message architecture makes AimOS not only easily scalable for the use in single board systems, but also applicable to single chassis and multi-chassis systems (of which each chassis can be located at different physical locations).
We have successfully built very small single board systems and large scale management solutions for up to 1000 units.

Utrechtseweg 38
1213 TV Hilversum
The Netherlands

phone +31 35 689 1900
sales@aimvalley.com
aimvalley.com/aimos

4.3 Hardware Abstraction

Hardware abstraction is divided into two major parts in AimOS; controller abstraction and ASIC abstraction.

- The first part focuses on the controller asset IO devices such as GPIO and I2C interfaces. Devices connected to these interfaces share a common interface definition and it is worthwhile to use a framework that provides all the possible functions and not have those sloppy copies for every IO device.
- The second part focuses on ASIC abstraction. Many ASIC suppliers provide either an Sofware Development Kit (SDK) or library to access the ASIC that needs to be integrated into the application or they provide a register map (and any variant in between).

AimOS handles the first type of ASIC interface via a high level driver interface, where the ASIC model is isolated and translated into the internal model, ensuring that the application is truly agnostic of the ASIC used. In the second option, the register map is accessed via a low level driver (that can be generated from the provided register map) and the aforementioned high level driver. Only the high level drivers need to be coded/implemented manually since they are not auto-generated.

4.4 Protocol Stacks

AimOS was originally designed for managing various connectivity products. It was used for SDH, Ethernet, SONET, ATM, OTN and CPRI based systems and for each type of system various protocol stacks are supported.

Protocol implementations are Operating System and ASIC agnostic and utilize only the internal AimOS data model. The hardware abstraction layer shields the hardware specifics from the protocol implementations.

4.5 Service Integration

There are several degrees of integration in which Linux services can be included into AimOS. Ranging from a full integration that includes the integration of the service data model into the AimOS internal data model, to a light integration where the service configuration data is maintained outside AimOS and only a limited set of provisionable data and functions are accessible and under control of AimOS.

Protocol packet trapping or event handling can be integrated similar to the configuration data. Where the data can be handed-off early to the Operating Service or handled/processed internally by AimOS. The framework supports both and any hybrid form that is required.

Utrechtseweg 38
1213 TV Hilversum
The Netherlands

phone +31 35 689 1900
sales@aimvalley.com
aimvalley.com/aimos

4.6 SDK/Build Environment

The build environment runs in a Linux environment and makes use of makepp as an automated build utility. The build tool is menu driven and produces all the required images, libraries and other components of the final product. The principle behind AimValley's build environment is that the complete product can be recreated from the sources/components and that the result is the same every time, except for the data and time stamp of the build. This guarantees that at any given moment in time we can debug and fix issues on any delivered product.

The result of the build process is typically a catalog image that contains all the resources required for the embedded system such as a boot, kernel, file system and various applications. Note that on a distributed system, the system catalog includes a catalog per unit type. The unit catalog is sent to the remote unit during an upgrade.

The build environment allows AimValley to build a specific set of (sub)products for various customers from the same source tree and at the same time separating the customer specific sources from each other. The same principle is used to include AimValley's common assets into a new build. These assets are not stored in the platform source tree but retrieved during compile time, based upon a label directly from the asset storage. The build environment can also extract the sources and build for a specific customer. This allows us to easily distribute a source package to different customers.

The build environment also provides several host and simulation images. These host and simulation images are often used during the development phase because the host is more accessible than an embedded system and more resources and tools are available on the development platform than on an embedded system. This has shown to be very powerful in the past where hardware was often available late in the development process and allowed us to prepare and verify most of the software before any hardware was available. Depending on the application hybrid, setups are possible where simulated units and embedded units are used in the same setup. This is extremely useful for instance during protocol development or during large scale integration testing because we can simulate a large number of units and mix them with real embedded units.

Last but not least the build environment includes the automated build server "Jenkins" to take care of continuous static code analysis runs, regression suite runs and build actions when a new branch of code is pushed for review in Gerrit.

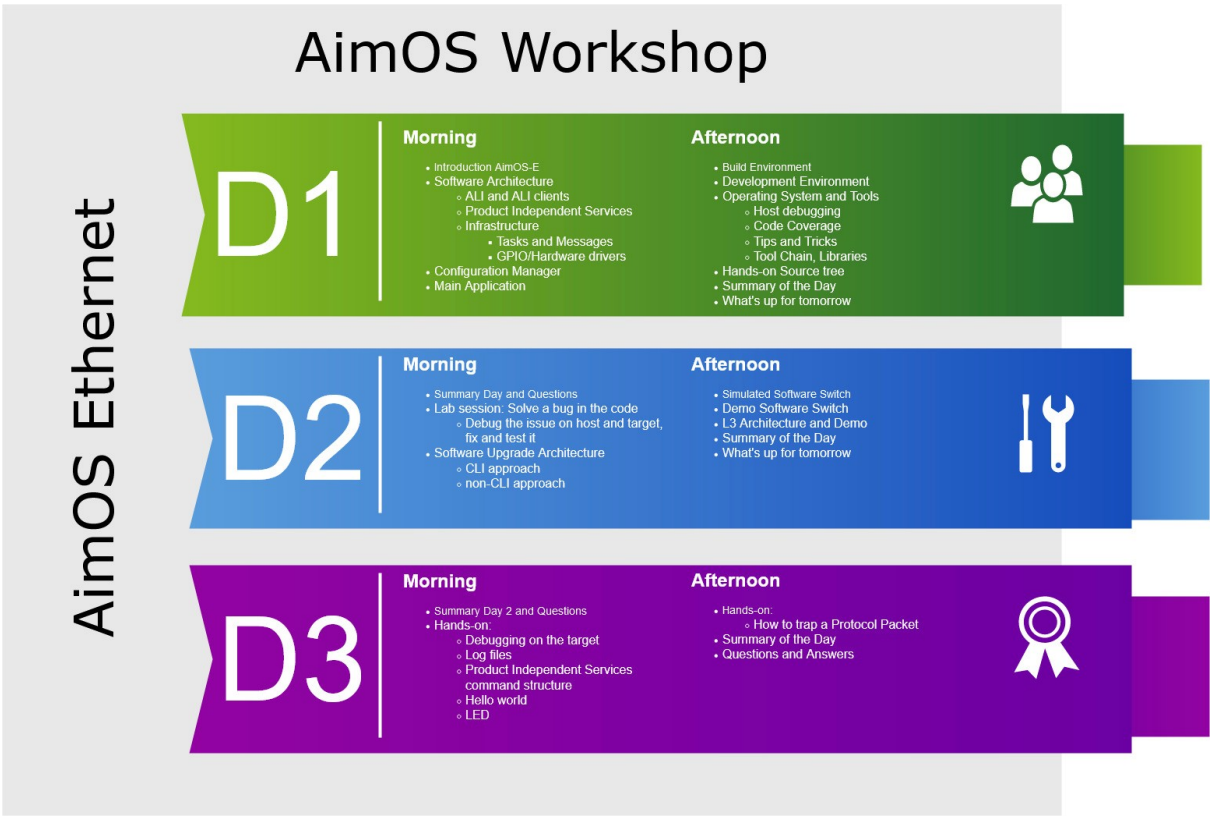
Utrechtseweg 38
1213 TV Hilversum
The Netherlands

phone +31 35 689 1900
sales@aimvalley.com
aimvalley.com/aimos

5. Joint Development

The build environment also enables a joint development process with our customers if they desire to do so. At AimValley we use Gerrit as a version control and peer review system which allows our customers to really participate in the development process. In order to further accommodate our customers we have a documentation set that describes how to use, modify and extend the environment.

There is also a 3-day workshop for customers, aimed towards knowledge transition which eases the development process.



6. Source Lines of Code

To give an idea of the size of the framework SLOCcount is run on the repository.
The AimOS framework, without Operating System and no application, reports around 160 000 lines of code.

```
Total Physical Source Lines of Code (SLOC)                = 160,598
Development Effort Estimate, Person-Years (Person-Months) = 41.41 (496.86)
 (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months)                         = 2.20 (26.46)
 (Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule)  = 18.78
Total Estimated Cost to Develop                           = $ 5,593,308
 (average salary = $56,286/year, overhead = 2.40).
SLOCcount, Copyright (C) 2001-2004 David A. Wheeler
SLOCcount is Open Source Software/Free Software, licensed under the GNU GPL.
SLOCCount comes with ABSOLUTELY NO WARRANTY, and you are welcome to
redistribute it under certain conditions as specified by the GNU GPL license;
see the documentation for details.
Please credit this data as "generated using David A. Wheeler's 'SLOCCount'."
```

```
Total Physical Source Lines of Code (SLOC)                = 237,720
Development Effort Estimate, Person-Years (Person-Months) = 62.50 (750.03)
 (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months)                         = 2.58 (30.94)
 (Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule)  = 24.24
Total Estimated Cost to Develop                           = $ 8,443,269
 (average salary = $56,286/year, overhead = 2.40).
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
SLOCCount is Open Source Software/Free Software, licensed under the GNU GPL.
SLOCCount comes with ABSOLUTELY NO WARRANTY, and you are welcome to
redistribute it under certain conditions as specified by the GNU GPL license;
see the documentation for details.
Please credit this data as "generated using David A. Wheeler's 'SLOCCount'."
```

The same skeleton AimOS framework built into an executable, reports around 240 000 lines of code. The difference is because of the generated interfaces, components, database models, state machines and common asset that are included.

The AimOS Ethernet application gives an idea of the size of a single board L2/L3 Ethernet application. Note that the AimOS Ethernet repository does not include an Operating System or file system.
SLOCcount reports around 1,1 million lines of code before building.

```
Total Physical Source Lines of Code (SLOC)                = 1,128,244
Development Effort Estimate, Person-Years (Person-Months) = 320.67 (3,847.99)
 (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months)                         = 4.80 (57.59)
 (Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule)  = 66.82
Total Estimated Cost to Develop                           = $ 43,317,642
 (average salary = $56,286/year, overhead = 2.40).
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
SLOCCount is Open Source Software/Free Software, licensed under the GNU GPL.
SLOCCount comes with ABSOLUTELY NO WARRANTY, and you are welcome to
redistribute it under certain conditions as specified by the GNU GPL license;
see the documentation for details.
Please credit this data as "generated using David A. Wheeler's 'SLOCCount'."
```

```
Total Physical Source Lines of Code (SLOC)                = 1,238,817
Development Effort Estimate, Person-Years (Person-Months) = 353.74 (4,244.91)
 (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months)                         = 4.98 (59.78)
 (Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule)  = 71.01
Total Estimated Cost to Develop                           = $ 47,785,830
 (average salary = $56,286/year, overhead = 2.40).
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
SLOCCount is Open Source Software/Free Software, licensed under the GNU GPL.
SLOCCount comes with ABSOLUTELY NO WARRANTY, and you are welcome to
redistribute it under certain conditions as specified by the GNU GPL license;
see the documentation for details.
Please credit this data as "generated using David A. Wheeler's 'SLOCCount'."
```

After building AimOS Ethernet SLOCcount, reports around 1,2 million lines of code.

AimValley
Utrechtseweg 38
1213 TV Hilversum
The Netherlands
phone +31 35 689 1900
sales@aimvalley.com
aimvalley.com/aimos

7. Why AimValley?

Based on years of experience in developing telecom and datacom systems, AimValley helps you with your development based on AimOS. This collection of software assets and tools can be used for any type of carrier-grade switch. AimOS supports for example; Broadcom, NXP and Marvell switches.

Services range from consultancy during your architecture phase, porting AimOS to your own hardware, designing your hardware, up to a full system development cycle including production, maintenance & support.

- Reliable partnership
- Design flexibility
- Extensive experience in developing telecom and datacom systems
- Strong track record in delivering as planned and within budget
- Evaluation of jump-start product development
- Support of various hardware platforms
- Key software and hardware expertise available for the Broadcom® StrataConnect and StrataXGS

We take care of very step in your development process, either based on your requirements or as a joint development project.

8. Further information

Our experienced engineering team with expertise in systems engineering, software, hardware and ASIC/FPGA design can support you through all phases in your product development. The Broadcom® experience of our teams is well known in the industry.

For further information contact sales@aimvalley.com

AimValley
Utrechtseweg 38
1213 TV Hilversum
The Netherlands

phone +31 35 689 1900
sales@aimvalley.com
aimvalley.com/aimos