Analogue Dynamics Engine (ADE)
Final Year Project 2005

Muiris Woulfe
B.A. (Mod.) Computer Science
Supervisor: Michael Manzke

Acknowledgements

I would like to express my gratitude to my supervisor, Michael Manzke, for his helpful advice and guidance throughout the project.

Additionally, I would like to take this opportunity to sincerely thank my parents for their continued support throughout my time at college.

Abstract

This report outlines the design and implementation of the Analogue Dynamics Engine (ADE). The ADE is a physics engine constructed from a hybrid, analogue and digital, computer. Software physics engines are becoming increasingly common in computer games, and the ADE was designed as a hardware equivalent to these software engines. Analogue computers, although currently rare, have useful properties such as their ability to evaluate functions in real-time. The physics engine exploits this functionality while using digital components to provide reconfigurability.

The core hybrid computer was constructed by connecting twenty nine custom designed reconfigurable analogue cells to thirty two bus lines, using programmable interconnect. Each cell can perform inversion, integration, addition and multiplication. At the periphery of this computer lie two ADCs and two DACs, so that the hybrid computer may provide a digital interface.

In order to make the engine suitable for use with games, it was decided to make simulations multiplexable, so that multiple simulations could be run "concurrently". This requires simulations to be executed faster than real-time. Additionally, state must be saved and restored, which was achieved through replicating the capacitors.

Finally, this report analyses the viability of this project for use in computer games. Ultimately, it was determined that an analogue computer could become a viable replacement for the software physics engines in use today. In fact, it offers benefits that cannot be obtained using today's software physics engines.

Contents

List of Figures

List of Tables

List of Code

CD Contents

`/Code/ADE/Mentor/Behavioural/` The entire behavioural level code and associated testbenches, contained in a Mentor Graphics' SystemVision 2002 project.

`/Code/ADE/Mentor/Behavioural/Simulations/` The simulations of the behavioural code. These files may be opened using Mentor Graphics' Waveform Analyzer.

`/Code/ADE/Mentor/Structural/` The entire structural level code and associated testbenches, contained in a Mentor Graphics' SystemVision 2002 project.

`/Code/ADE/Mentor/Structural/Simulations/` The simulations of the structural code. These files may be opened using Mentor Graphics' Waveform Analyzer.

`/Code/ADE/Xilinx/Behavioural/` The digital behavioural level code and associated testbenches, contained in a Xilinx ISE 6.2i project.

`/Code/ADE/Xilinx/Behavioural/Simulations/` The simulations of the digital behavioural code. These files may be opened using ModelSim.

`/Code/ADE/Xilinx/Structural/` The digital structural level code and associated testbenches, contained in a Xilinx ISE 6.2i project.

`/Code/ADE/Xilinx/Structural/Simulations/` The simulations of the digital structural code. These files may be opened using ModelSim.

`/Code/Execution Speed/Linux/` The program used in Section 15.4.3 to measure the execution speed of ODE, compiled for Fedora Core 3. A makefile is provided.

`/Code/Execution Speed/Microsoft Windows/VS .NET 2002/` The program used in Section 15.4.3 to measure the execution speed of ODE, compiled for Microsoft Windows using Visual Studio .NET 2002. The project files are provided.

`/Code/Execution Speed/Microsoft Windows/VS .NET 2003/` The program used in Section 15.4.3 to measure the execution speed of ODE, compiled for Microsoft Windows using Visual Studio .NET 2003. The project files are provided.

`/Code/ExecutionSpeed/ODE/` The nightly snapshot of ODE used throughout project, dated 2004-11-10.

`/Documentation/Report/` A hyperlinked PDF version of this report.

`/Documentation/Execution Speed/` The raw execution speed data obtained from the program discussed in Section 15.4.3, in PDF, CSV, Microsoft Excel and Microsoft Excel XML formats.

Part I

Background

Chapter 1

Introduction

This chapter defines the key idea behind the project. The chapter continues by describing the project's fundamental elements, namely physics engines, dedicated hardware, analogue computers and hybrid computers, while outlining the progression of the idea behind the project. After summarising the objective, the main advantages are highlighted.

1.1 Project Overview

The primary objective of this project was to construct a physics engine, using innovative technology. The following sections discuss the conception of the idea and its progression into its final incarnation.

1.1.1 Physics Engine

Today, an increasing number of applications are constructed using physics as their foundation. Physics allow objects modelled by the software to appear as they would in the real world. Typically, these physics calculations are performed by a software library termed a physics engine. These engines have resulted in computer games becoming increasingly realistic. As time progresses and computing power increases, interest in using physics similarly increases.

However, the complex calculations performed by physics engines make them relatively CPU intensive. Often, this requires game developers to make a tradeoff between graphics and physics. Physics engines would become much more useful if this tradeoff were unnecessary. Conceivably, if physics engines were to be implemented in dedicated hardware, this tradeoff would be overcome.

A more comprehensive overview of physics engines is provided in Chapter 2.

1.1.2 Dedicated Hardware

If software is implemented in dedicated hardware, then it will usually perform faster. The CPU is free to perform other tasks while the dedicated hardware concurrently performs its own. Traditionally, dedicated hardware has been created for CPU intensive software, such as Graphics Processing Units (GPUs) and MPEG decoders. Therefore, implementing a physics engine from dedicated hardware would be beneficial.

Digital hardware would be the accepted choice for implementing such dedicated hardware. However, digital hardware operates by successively applying operations to data. This means that, for example, finding the second derivative of a function would take twice as long as finding the first derivative, unless some optimisation was utilised. Since physics is based on complex mathematics, this successive application of operations would become a bottleneck. Consequently, digital hardware may not be the most suitable approach for constructing a physics engine. A more suitable approach could be to implement the hardware as an analogue computer.

1.1.3 Analogue Computer

An analogue computer is a computer that is constructed from analogue components. Therefore, analogue computers process analogue signals.

The main advantage provided by analogue computers is that they process signals in real-time, effectively eliminating all propagation delays. The derivative of a function may be obtained by inputting that function to a differentiator. If the derivative after one second were desired, then the output would be read after one second. In fact, the output is valid at a potentially infinite range of times. In contrast, a digital computer can only calculate the derivative at discrete time steps, and would require nonzero time to perform each calculation. This real-time behaviour is beneficial for computer games, to prevent any interruption arising from the time taken to perform a calculation.

Another advantage ensuing from this real-time behaviour is parallelism. Analogue parallelism ensures that the second derivative takes the same time to calculate as the first derivative. In other words, the two differentiators work in parallel. Such an advantage is typically impossible in digital hardware.

Further, analogue computers are ideal for implementing problems that may be described mathematically. Physics fulfils this criterion.

Therefore, it appears that many advantages would be obtained by using an analogue computer to implement a physics engine.

However, one major disadvantage remains with analogue computers: they are not dynamically reconfigurable, unlike their digital counterparts. In fact, the requirement for manual configuration is one of the primary reasons that digital computers have superseded analogue computers. Such a problem would be disastrous for a physics engine. A solution is to couple an analogue computer with digital components, creating a hybrid computer.

Additional information on analogue computers and their advantages is presented in Chapter 3.

1.1.4 Hybrid Computer

A hybrid computer is essentially an analogue computer coupled with digital components. Such a computer can exploit the advantages offered by both analogue and digital.

In a physics engine, a hybrid computer could utilise the parallelism and real-time behaviour inherent in analogue computation, while using the dynamic reconfigurability afforded by digital computers. A hybrid computer could also allow other useful digital hardware to be integrated into the solution, allowing for greater flexibility and functionality. Finally, digital hardware is a *sine qua non* for interfacing with the digital computer that will use the physics engine.

Supplementary information on hybrid computers is furnished in Chapter 4.

1.1.5 Project Definition

Essentially, this project involves the research, design and implementation of a hardware physics engine using a combination of analogue and digital computer components. Ultimately, this project will aim to determine whether such a proposition is viable and whether it offers an improvement over the software physics engines in use today. If it were viable, such hardware could eventually be placed on graphics cards or on new physics cards that communicate with the PC via the PCIe bus.

1.2 Advantages

The primary advantage obtained by the design and implementation of such a physics engine is that it should operate substantially faster than the software physics engines in use today. This speed gain is achieved primarily through the inherent real-time behaviour and parallel nature of analogue computers. These advantages will be discussed further in Section 3.4.1 and Section 3.4.2 respectively, after analogue computers have been discussed in greater depth.

Chapter 2

Physics Engines

This chapter provides an overview of physics engines, briefly discussing their underlying physics and mathematics. The chapter continues by providing a brief history of physics engines before enumerating some available engines. Finally, the applications of physics engines and the advantages that may be obtained through their use are outlined.

The primary objective of this chapter is to furnish the reader with an explanation of physics engines, in addition to the capabilities of currently available engines. The chapter analyses potential applications in order to determine the primary application of the project's physics engine. This ultimately allowed the design of the project's physics engine to be enhanced for its desired application.

2.1 Overview

A physics engine or physics software development kit (SDK) is a middleware solution that performs physics calculations on behalf of other software, to simulate realistically the behaviour of objects. Physics engines may be integrated with software that requires physics calculations to be performed.

Traditionally, physics engines have modelled rigid body dynamics, which describe the interactions between rigid bodies or solid objects. These are typically modelled by ordinary differential equations (ODEs), which are capable of expressing the time-varying behaviour of a system. Recently, physics engines have expanded their abilities beyond rigid body dynamics to include related fields. For this project, rigid body dynamics is the only field of concern, but other areas could easily be added at a later stage. Physics engines typically work with Newtonian physics, since the extra accuracy provided by Einsteinian physics is unlikely to be noticed but substantially increases the complexity of the calculations.

2.2 History

Observing the growing use of physics in games, MathEngine released the first physics engine, the Fast Dynamics Toolkit in 1998. However, the engine often created jittering objects in resting contact with a flat surface. According to Eberly [1, p. 4], this resulted from inaccuracies arising from the application of numerical methods to differential equations and from underdeveloped collision detection algorithms.

To resolve this problem, Hugh Reynolds and Dr Steven Collins founded Telekinesys in 1998. After renaming the company Havok.com, they released the Havok Game Dynamics SDK. This was the first physics engine to prove that sophisticated physics simulation could be achieved using consumer level CPUs.

Following the trend already established by GPUs, AGEIA announced PhysX [2], the first physics processing unit (PPU), on 7 March 2005. The PPU is capable of 32,000 rigid bodies, compared to the 200 typical for a CPU. It can handle 40,000 to 50,000 particles when simulating particle dynamics. If the PPU were unavailable, the physics calculations would be performed in software by the NovodeX engine, in the same way that software would be used to render graphics if no GPU were available. AGEIA intends to sell its PPU integrated circuits (ICs) to companies who will design and manufacture suitable boards, similar to what NVIDIA does with its GPUs.

2.3 Available Engines

A large number of commercial software physics engines are currently available.

The best known physics engine is Havok Physics [3]. The engine has been used for films, in addition to over one hundred games. To simplify development with Havok, plugins are available for Discreet's 3D Studio Max and Alias's Maya 3D. The software supports rigid body dynamics, vehicle dynamics, fluid dynamics, cloth simulation and ragdoll physics.

Recently, Meqon released the Meqon Game Dynamics SDK [4] physics engine. It supports the simulation of rigid bodies, vehicle dynamics, liquid surfaces, particle systems and characters. In addition, Meqon supplies the Meqon Simulator SDK, which provides greater realism for non-game simulations. This supports similar simulations to the Game Dynamics SDK.

RenderWare Physics [5], from Criterion Software, is based on the previously available MathEngine physics engine. It supports character simulation including ragdoll physics, in addition to rigid body dynamics. The software is available as part of a larger package, the RenderWare Platform, which also supports graphics, audio and artificial intelligence (AI).

The NovodeX Physics SDK [6] from AGEIA supports simulation of vehicle dynamics, ragdoll physics and characters, but highlights its collision detection algorithms as the most outstanding feature of the engine. Unlike its competitors, it supports multiprocessor systems and Intel and AMD's upcoming multicore processors.

SD/FAST [7] from PTC takes a radically different approach, even though it simulates rigid body dynamics. A description of the system is constructed and supplied to the software. The physics equations describing the system are then outputted in either C or Fortran, so that they may be integrated into the existing code base.

In addition to the commercial engines outlined above, there also exist some free, open source engines. However, these are typically less sophisticated.

The primary open source engine is the Open Dynamics Engine (ODE) [8]. This supports only rigid body dynamics, but its realistic simulations have led to its use in a relatively large number of games.

DynaMechs (Dynamics of Mechanisms) [9] allows for rigid body simulation, with a particular emphasis on articulated moving objects. However, it appears to be defunct as no updates have been made since July 2001.

AERO (Animation Editor for Realistic Object movements) [10] also offers rigid body dynamics, but it too appears to be defunct. No updates have been made since February 2001, but the last note from the developers had promised a complete rewrite of the engine.

In addition to the general purpose physics engines outlined above, there exist commercial and free physics engines designed for niche markets such as the simulation of only vehicles or robots.

Finally, due to the current demand for physics, some companies have started working on hardware physics engines or PPUs. The first of these, PhysX, will be available shortly. PhysX is capable of simulating rigid body dynamics, universal collision detection, finite element analysis, soft body dynamics, fluid dynamics, hair and clothing. More details are outlined in Section 2.2.

As can be seen from this discussion, a large number of physics engines are currently available. This highlights the growing demand for these engines.

2.4 Applications

There are two primary applications of physics engines: engineering analysis simulations and computer games. Both of these are discussed below.

2.4.1 Engineering Analyses

Engineering analysis simulations must use physics if they are to predict correctly the behaviour of a system. Examples of such applications would be those used to test if a designed bridge is sufficiently sturdy, or if a building is earthquake-proof.

Physics engines for these applications must be extremely accurate. If they were not, lives could be endangered. However, they do not need to perform their calculations rapidly. If lives depend on a simulation, it is deemed satisfactory if that simulation takes an entire day or longer to execute.

Since the project's physics engine is not particularly accurate, engineering analysis simulations are not its target market. The primary advantage of the project's physics engine is its real-time behaviour, which as outlined above, is unnecessary for these applications.

2.4.2 Computer Games

Many of today's foremost computer games use physics so that events modelled in the game appear as they would in the real world. Support for physics is increasing as both computing power and the demand for realism increases.

Physics could be used in games in order to model the suspension system of vehicles or the trajectory of thrown crates.

Physics engines for games must work in real-time. This is extremely important as the physics engine determines what the graphics engine renders. There must be no processing delay, as the graphics engine cannot wait for results, unless the game slows down. However, accuracy is not very important. Objects in the game must appear to behave correctly. Nevertheless, they do not need to behave exactly as predicted by physics. In fact, to satisfy the real-time constraints, software physics engines typically take shortcuts in the calculations performed, thereby reducing the accuracy but not the perceived accuracy.

Games are the main target for the project's physics engine, since games would benefit from the real-time behaviour of analogue computers but would be unaffected by the minor loss of accuracy.

2.5 Advantages

The three primary advantages that may be obtained using a physics engine are increased realism, leveraging of expertise and reduced expenditure. Each of these is discussed in the proceeding sections.

2.5.1 Realism

As outlined in Section 2.4.2, users are demanding increasing realism from computer games. Realism and accuracy are extremely important in engineering analyses (Section 2.4.1). It is extremely difficult to bluff this realism without a physics foundation. Reinforcing this view, Gary Powell of MathEngine plc stated "The illusion and immersive experience of the virtual world, so carefully built up with high polygon models, detailed textures and advanced lighting, is so often shattered as soon as objects begin to move and interact." [11, p. x] Based on these demands, many software developers now use physics engines to increase the realism of their products.

2.5.2 Expertise

The majority of game developers do not have a substantial knowledge of physics. Therefore, it is often desirable to use an existing physics engine instead of replicating the relevant parts inside a product under development. If an existing physics engine is used, the end developer is unburdened from having to understand the underlying physics. Moreover, the physics implementation is typically left to domain experts. These experts should have a greater knowledge of the relevant physics, thereby producing a superior physics engine. Therefore, physics engines are commonly used in order to leverage this expertise.

2.5.3 Expense

Due to the typical developer's lack of expertise in physics (Section 2.5.2), it is often time-consuming and expensive for companies to implement their own physics calculations.

If in-house physics software were to be developed, the largest cost would likely result from the need to polish the physics software, and not from traditional software engineering domains, as illustrated in Table 2.1. Therefore, this cost is typically concealed and unplanned, which could prove devastating for small companies.

Research and Development	10%
Implementation	10%
Debugging	20%
Perfecting the software	60%

Table 2.1: Costs arising from implementing in-house physics software [12, p. 23]

The cost of purchasing a physics engine may be minimal. Indeed, a number of free engines are already available, as discussed in Section 2.3. While the free physics engines are often less sophisticated than their commercial counterparts, their functionality is likely to be substantially greater than what could be implemented in-house on a limited development timeframe.

Overall, utilisation of an existing physics engine is likely to lead to savings for a development company.

Chapter 3

Analogue Computers

This chapter begins with an overview of analogue computers. It continues with their history before specifying their traditional application domains. Finally, the advantages and disadvantages of analogue computers are discussed.

The primary objective of this chapter is to provide the reader with an introduction to analogue computers. Their history provides an insight into their evolution, highlighting the different categories invented. Disadvantages are outlined so that their impact on the project's physics engine may be analysed.

3.1 Overview

An analogue computer is "a computer which operates with numbers represented by some physically measurable quantity, such as weight, length, voltage, etc." [13, p. 432].

It is possible to construct analogue computers that process a number of natural "analogue" phenomena such as hydraulics or mechanics, but the majority of analogue computers process only voltages. Consequently, all further discussion will be restricted to voltage processing or electronic analogue computers.

Analogue computers are based on mathematical operations. Therefore, unlike digital computers, they may be used to directly model equations that adhere to a number of restrictions. Consequently, the analogue computer is often regarded as a more natural tool for evaluating equations than a digital computer.

3.2 History

The history of analogue computers could be said to go back into antiquity, since devices like the slide rule can be considered to be analogue computers. However, as stated in Section 3.1, only electronic analogue computers will be considered here.

Between 1937 and 1938, George A Philbrick constructed the first electronic analogue computer while working at The Foxboro Company [14, pp. 131–135]. However, he referred to it as an "automatic control analyzer" as opposed to an analogue computer. It consisted of a hardwired analogue computer, with programming limited to potentiometer and switch settings. It operated much faster than real-time, displaying a steady output on the oscilloscope. The system was battery powered and attached to a built-in oscilloscope. Unlike later analogue computers, the system contained no operational amplifiers, as the devices had not yet been invented. Instead, networks of resistors, capacitors and amplifiers performed the mathematical operations. The machine was released as POLYPHEMUS.

Despite the seemingly limited programmability of the device, it was used for the simulation of a relatively wide variety of systems. For example, it was used for simulating liquid steam mixing baths and liquid level control processes. To simplify operation, a cardboard panel could be attached to the front. This panel illustrated the process under simulation and specified the purpose of the switches and potentiometers. These features meant that POLYPHEMUS was widely used for teaching and experimenting with control circuits.

A successor was constructed between 1945 and 1946. It operated on power from the mains, removing the batteries required by POLYPHEMUS. Additionally, two oscilloscopes were provided. Despite the success of these systems, they became the last of their kind.

Although they vary greatly from the operational amplifier based analogue computers used subsequently, they established the idea on which subsequent computers were based. They performed mathematical operations on voltages and displayed their output on an oscilloscope. Moreover, they could be quickly adjusted for rapid prototyping. The subsequent computers owe much to these founding computers.

In 1945, Bell Telephone Laboratories (BTL) began to analyse the viability of operational amplifier based analogue computers [15, p. 10]. Emory Lakatos directed the design of the resultant device, leading to its completion in 1949. However, the MIT's Dynamic Analysis and Control Laboratory (DACL), despite starting work after BTL, completed the world's first operational amplifier based analogue computer, the Flight Simulator, in 1948.

Despite these earlier computers, the first major development in operational amplifier based analogue computers was the establishment of Project Cyclone in 1946 [16, p. 3]. Constructed by Reeves Instrument Corporation and sponsored by the US Office of Naval Research (ONR), it was designed to simulate guided missile simulation. The system's functionality was expanded to form the first commercially available general purpose analogue computer, known as the Reeves Electronics Analog Computer (REAC). One of the defining characteristics of the REAC was that multiple computers could be purchased and combined to create one large computer. The system was a great success and was installed in many government laboratories, industrial laboratories and universities. At the ONR, the computer was used to run rapid missile simulations, typically lasting only one minute in duration. The accuracy was subsequently improved by executing the same simulation on a digital computer, which took from sixty to 130 hours. Project Cyclone produced the definitive analogue computer, whose design was mirrored by the analogue computer industry for many subsequent years.

Project Typhoon became Project Cyclone's successor when it was initiated by the ONR in 1947 [17, pp. 93–95]. The project, undertaken by the Radio Corporation of America (RCA), created a single purpose built computer that was never commercialised. The emphasis of the project was on high performance and high precision. This led to the development to the drift-free DC operational amplifier, which became a requisite component for all subsequent analogue computers.

Both projects had enormous influence on the future of the electronic analogue computer. According to Small, "Projects Cyclone and Typhoon were instrumental in establishing the technological basis for the postwar general-purpose electronic analogue computer industry in the USA." [17, p. 89]

In the UK, the Royal Aircraft Establishment (RAE) had a number of analogue computers built for missile simulation. These included the GEPUS (General-Purpose Simulator), TRIDAC (Three-Dimensional Analogue Computer) and G-PAC (General-Purpose Analogue Computer).

In the late 1940s, George A Philbrick Researches, Inc (GAP/R) constructed the first commercial analogue computers [15, p. 12]. Like POLYPHEMUS but unlike the first operational amplifier based computers, these were repetitive operation or "rep op" machines. These machines repeated their simulations much faster than real-time, so that a steady waveform could be displaced on an oscilloscope. This allowed parameters to be adjusted and the effects seen instantly. However, these machines failed to achieve commercial success.

Observing the continued reliance of the militaries of the US and UK on analogue computers, commercial interest grew substantially throughout the 1950s. This led to the formation of a substantial number of companies manufacturing general purpose analogue computers in the US, UK, USSR, Japan, West Germany and France. In 1955, ninety-five analogue computer installations existed in the US. However, the late 1950s saw a shift in the market. Repetitive operation or "rep op" machines started to gain popularity, eroding the market share of single shot analogue computers. Although GAP/R had always manufactured repetitive operation analogue computers, a vast improvement in electronic component design finally made these computers competitive.

The early 1960s saw a decline in the number of analogue computers, as hybrid computers began to grow in popularity (Section 4.2). However, the demise of analogue computers did not occur until the late 1970s, when the increasing speed and power of digital computers made analogue computers a less viable alternative. Digital computers were then able to match the real-time behaviour of analogue computers for many operations and offered greater flexibility and easy of use for many applications. Additionally, substantial work in the development of algorithms and numerical analysis combined to improve the accuracy of digital computers to be greater than that offered by analogue computers [18, p. 59].

3.3 Applications

Analogue computers were used for two primary purposes: simulation and control. Both of these are discussed below.

3.3.1 Simulation

Analogue computers were used to simulate various systems, modelling entities using representations or analogues of those entities. These systems were used where it would be too hazardous or too expensive to experiment directly. Often simulations were “person in the loop”, meaning they were interactive.

For example, in the early 1960s, NASA ran simulations which placed pilots or astronauts into capsules that were “flown” in simulated orbits or under the sea [18, p. 57]. Other systems included the English Electric Saturn, designed for modelling an entire nuclear power station, and the RAE’s TRIDAC (Section 3.2), designed for the simulation of aircraft and missile systems [19, pp. 80–85].

In this project, analogue computers were used for simulation, as the purpose of a physics engine is to simulate physics systems.

3.3.2 Control

Analogue computers were also used in control systems. “A control system is a group of components assembled in such a way as to regulate an energy input to achieve the desired output.” [20, p. 2] Analogue computers can be used to control the behaviour of a closed loop system, whereby the system’s output is used to influence its input.

Analogue computers were regularly used in the control systems of aircraft, for example in automatic pilot control systems [21, pp. 94–97]. Such systems were designed to keep the aircraft flying on a fixed compass bearing. The control systems monitored the flight of aircraft for deviations and controlled the rudders to perform corrections as necessary. Therefore, these closed loop control systems were able to keep the flight path of aircraft straight. More sophisticated analogue computer control systems were found on the Concorde, which used analogue computers to implement fly-by-wire.

3.4 Advantages

An analogue computer operates in real-time, with an inherent parallelism, providing results with potentially infinite accuracy. Each of these benefits is described in sequence.

3.4.1 Real-Time

When a digital computer performs a calculation, there is a delay called the propagation delay before the result is obtained. This delay is primarily influenced by the complexity of the calculation being performed. As computing power increases, this propagation delay is gradually reduced and becomes less of a problem. However, complex calculations, such as the computation of large prime numbers, still require large amounts of computational power and time. The elimination of this propagation delay would be of great benefit.

Analogue computers eliminate this propagation delay, as they operate in real-time. While this is useful for even a single calculation, the real advantage becomes apparent when the same value must be recalculated at myriad time steps. A single calculation may be left running for a specified period with samples taken at the necessary intervals.

Suppose a variable must be differentiated with regard to time, using one millisecond intervals, until one second has elapsed. A digital computer must perform one thousand calculations to acquire the desired data. In contrast, an analogue computer consisting of a differentiator would be left running for one second, with the output sampled every millisecond. Although modern digital computers should be able to match the performance of the analogue computer in this simple example, this would not hold for cases that are more complex.

If real-time is too slow or too quick for the required application, time compression or time expansion can be used to accelerate or decelerate the computation respectively. This may be achieved through modifying the equations used to construct the analogue computer, using simple mathematical manipulation.

3.4.2 Parallelism

Parallelism is both a widely researched and widely implemented method for gaining greater performance from digital hardware. However, for analogue computers, parallelism needs neither to be researched nor implemented, for it exists by default.

As outlined in Section 3.4.1, analogue computers operate in real-time. This real-time behaviour permits analogue computers to achieve greater parallelism than their digital counterparts.

Figure 3.1 shows a block diagram of a sample analogue computer. This computer integrates its two inputs and adds the two resultant signals to generate the output. In this computer, the results would be outputted from both integrators simultaneously. Nevertheless, two identical integrators in digital hardware would also work in this way. However, the subsequent addition will happen in effectively zero time since its output will be generated concurrently with the outputs of the integrators. In other words, the adder will instantaneously add the outputs of the integrators. To recapitulate, the two integrations and the addition are performed in parallel. In contrast, digital hardware would typically perform the integrations first, with the addition performed subsequently, leading to theoretically greater delays.

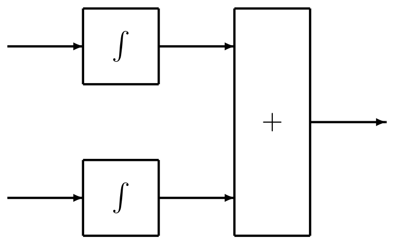


Figure 3.1: Parallelism in analogue computers

This real-time nature also results in the automatic synchronisation of signals, something that often requires additional logic in digital hardware. For example, if the result of a multiplication and an integration are to be added, both results will reach the adder simultaneously. In digital hardware, the integration would likely take longer to perform than the multiplication, so waiting in coordination with the clock signal would be needed to synchronise the adder's inputs.

To summarise, in an analogue computer the propagation delay is effectively zero, irrespective of the complexity of its function. This is the greatest advantage offered by analogue computers.

3.4.3 Potentially Infinite Accuracy

Digital computers represent numbers using a finite number of bits. Consequently, digital signals always have a certain finite granularity or resolution. The only way to increase this resolution is to increase the word length, which is essentially the standard number of bits used to represent an entity in a particular architecture, thereby consuming more die area and making the integrated circuits more expensive to manufacture. This finite granularity is clearly a problem since manufacturers are continually attempting to increase the word length of their CPUs, as evidenced by the recent migration of both the IBM PowerPC and Intel 80x86 architectures from 32 to 64 bits.

This has led to the formation of the study of numerical analysis or numerical methods. It involves reformulating mathematical problems in order to avoid truncation and rounding errors resulting from this finite number of bits. Such solutions only slightly improve accuracy and do not work in all situations. This represents a problem. As Stoer and Bulirsch state, "Assessing the accuracy of the results of calculations is a paramount goal" [22, p. 1]. It would be advantageous if these problems could be completely disregarded.

Analogue computers represent numbers using continuous signals, with a potentially infinite range of values. Consequently, analogue computers do not suffer from any granularity problem. Therefore, the problems associated with digital may essentially be completely disregarded. This has resulted in computer theorists referring to analogue computers as real computers, which operate on the set of real numbers.

In summation, digital computers represent entities using an approximation while analogue computers represent entities using analogues to those entities.

3.5 Disadvantages

Analogue computers are rare today. This indicates that analogue computers have disadvantages. The main issues are noise, inflexibility and that they are not dynamically reconfigurable. This section discusses each of these in turn and highlights how they are unproblematic for the project's physics engine.

3.5.1 Noise

Theoretical analogue computers offer infinite accuracy (Section 3.4.3). However, practical analogue computers cannot realise this ideal.

Analogue computers, like all analogue devices, are affected by noise. Every analogue component will introduce artefacts on waveforms, as a side effect of that component's primary function. This is one of the primary reasons for the current lack of interest in analogue components. Meanwhile, digital components use only two voltage levels. An erroneous voltage can be corrected to the nearest permissible voltage. Furthermore, adding error detection and correction circuits to digital computers is unproblematic, but adding such circuits to analogue computers is relatively difficult.

There are, however, various techniques for reducing noise in analogue computers. The most successful technique is to use higher quality and, consequently, more expensive components. Additionally, there are a number of design rules that attempt to reduce noise. One such design rule is specified in Section 6.6.5.

As outlined in Section 2.4.2, the primary application for the project's physics engine is computer games. High accuracy is unimportant for games. Objects must only appear realistic; their behaviour does not need to be entirely accurate. In the project's physics engine, any deviations were, as predicted, well within the limits of human perception. Therefore, for this physics engine, noise was almost completely disregarded.

3.5.2 Inflexibility

Analogue computers are inflexible in that they are limited in the variety of functions that they can perform. For computation on an analogue computer, a program must have a mathematical basis. In comparison, digital computers only require programs to have a Boolean algebraic basis, substantially increasing the variety of programs that can be created. For example, a program based on string manipulation could be expressed in Boolean algebra, but not in traditional mathematics. This inflexibility was one of the principal reasons why analogue computers have decreased in popularity.

Physics is based on mathematics. Therefore, the inflexibility of analogue computers did not restrict the project's physics engine. If additional functionality were desired, the digital circuits of hybrid computers could be used to supplement the capabilities of analogue computers, as discussed in Section 4.4.2.

3.5.3 Not Dynamically Reconfigurable

Traditional analogue computers cannot be dynamically reconfigured. Instead, their components are wired by hand for each objective. Wiring involves connecting the necessary components together through a "patch-board" or "patch panel" consisting of terminals, similar to manually operated telephone exchanges. Sophisticated analogue computers had detachable patch-boards that could be removed from the "patch bay" and later replaced, similar to the way programs can be saved on a digital computer.

This could have been problematic for the project's physics engine. However, as outlined in Section 1.1.4, hybrid computers offer a solution to this problem. Hybrid computers are discussed in Chapter 4.

Chapter 4

Hybrid Computers

Firstly, an overview of hybrid computers is presented. Next, their history is depicted before their traditional application domain is described. Finally, the advantages and disadvantages of hybrid computers are discussed.

This chapter's objective is to provide the reader with an introduction to hybrid computers. Their history provides an insight into the different hybridisation concepts proposed. Disadvantages are outlined so that their influence on the project's physics engine may be analysed.

4.1 Overview

A hybrid computer is a computer that consists of analogue and digital computational elements. There is a cornucopia of definitions describing what exactly constitutes a hybrid computer, but for the purposes of this report, a hybrid computer is defined to be an analogue computer making use of some digital components.

The construction of such a system is not trivial, since analogue and digital are extremely dissimilar. The outputs generated by analogue components are continuous, while the outputs generated by digital components are discontinuous.

4.2 History

The first hybrid systems constructed consisted of an existing analogue computer connected to an existing digital computer. They were not entirely new systems but the interconnection of two readily available systems.

The expansion of the US intercontinental ballistic missile (ICBM) programme in 1954, followed by the space race of the 1960s demanded greater computational power. Greater computational power meant that both analogue computers and their programs would substantially increase in size, making programming difficult and error-prone [17, p. 239]. Hybrid computers were proposed as the solution.

Convair Astronautics designed the first hybrid system in 1954, to perform simulation studies for the Atlas ICBM [15, p. 13]. It consisted of an Electronic

Associates, Inc (EAI) PACE analogue computer connected to an IBM 704 digital computer, via a converter called an Add-a-Verter manufactured by Epsco, Inc.

In 1955, Ramo-Wooldridge Corporation developed a second hybrid system, for the same purpose as Convair's. It also used an Add-a-Verter and a PACE analogue computer, but used a UNIVAC 1103A for its digital computation.

Commercially available computers typically used designs that were more primitive. For example, the HYDAC 2000 added some digital control elements to an analogue computer. It was much later before more sophisticated designs became available.

Despite the quantity of hybrid computers constructed, there was often disagreement as to the best way to unite analogue and digital, leading to a multitude of esoteric designs. For example, the TRICE system designed by Packard Bell for NASA's spaceflight simulations used a pulse frequency modulated signal as the information carrier, whereas most used conventional signals [18, p. 59].

Hybrid computers declined around the late 1970s during the demise of analogue computers (Section 3.2). After analogue computers were deemed obsolete, many believed the analogue component of the hybrid computer was a hindrance and that a digital computer was a better investment. Furthermore, it became increasingly obvious that hybrid computers were losing their niche as the abilities of digital computers improved [17, p. 264].

4.3 Applications

Hybrid computers can potentially be used where either analogue or digital computers are used. However, they were typically used for simulations that needed to model elements too complex for pure analogue computers. In particular, hybrid computers were often used for scientific applications, in contrast to pure analogue computers, which were often used only for engineering applications.

For example, Professor Vincent Rideout from the University of Wisconsin, in associated with NASA, modelled the cardiovascular respiratory system of the human body in 1972 [23, pp. 10–12]. This simulation used 120 differential equations to simulate the human heart, circulatory system, lungs and control systems.

4.4 Advantages

The advantages associated with hybrid computers are their ease of reconfigurability and the way they unite the advantages of analogue and digital. These advantages are outlined below.

4.4.1 Reconfigurability

The greatest advantage attained through the fusion of analogue and digital computing elements is reconfigurability. As outlined in Section 3.5.3, pure analogue computers are not reconfigurable.

Digitally controlled analogue routing elements can be created to dynamically route analogue signals. Therefore, for increased flexibility, the control signals for these elements can originate from additional digital computing elements. Instead of manually wiring the computer, programs may be written that automatically reconfigure the computer. Such solutions alleviate the difficulty of manually programming an analogue computer. Indeed, one of the motivating factors in the development of hybrid computers was the increasing complexity of analogue computers and the laboriousness of interconnecting their components.

Hybrid solutions impart the propensity to conceive reconfigurable analogue computers. This is the catalyst that made the project's physics engine viable.

4.4.2 Leveraging Analogue and Digital

As outlined in Section 3.5.2, analogue computers have the ability to solve only a subset of the problems solvable on digital computers. However, as outlined in Section 3.4, analogue computers have certain advantages over their digital counterparts.

Hybrid computers can be constructed with analogue and digital partitions of equal status. Then, all functions that can be performed on an analogue computer would be performed using the analogue components. All other necessary functions would be performed using the digital components. This would allow for the computation of all functions that can be performed using a digital computer while leveraging the unique properties of analogue computers.

Hybrid computers offer a substantial increase in ability over analogue computers. However, it transpired that this functionality was not required by the project's physics engine.

4.5 Disadvantages

Although hybrid computers solve many of the issues arising from analogue computers, the problem of noise remains partially unsolved. This issue is examined below.

4.5.1 Noise

The analogue components in a hybrid computer will suffer from noise. Moreover, since results generated by these analogue components will be used by digital components, the digital components will also essentially be affected by noise. A more thorough analysis of noise is provided in Section 3.5.1.

As with analogue computers, this noise is likely to prove unproblematic for the project's physics engine since it is unlikely to reduce the perceived accuracy of the engine.

Chapter 5

Analogue

The primary objective of this chapter is to outline how analogue is still widely used today, despite advances in digital technology.

5.1 Overview

Analogue signals are defined over a continuous range of amplitudes. Typically, they are defined over a continuous range of times, as illustrated in Figure 5.1a. However, in ICs analogue signals are often defined only at discrete time values, as illustrated in Figure 5.1b [24, p. 2]. These are referred to as discrete time or sampled data analogue signals.

In contrast, digital signals are defined only at discrete times and discrete amplitudes, as illustrated in Figure 5.1c. They represent objects using only two values, zero and one.

Since digital is discrete in both time and amplitude, continuous analogue signals potentially allow for more accurate modelling of many objects. The properties of objects rarely have a finite number of levels but an infinite number.

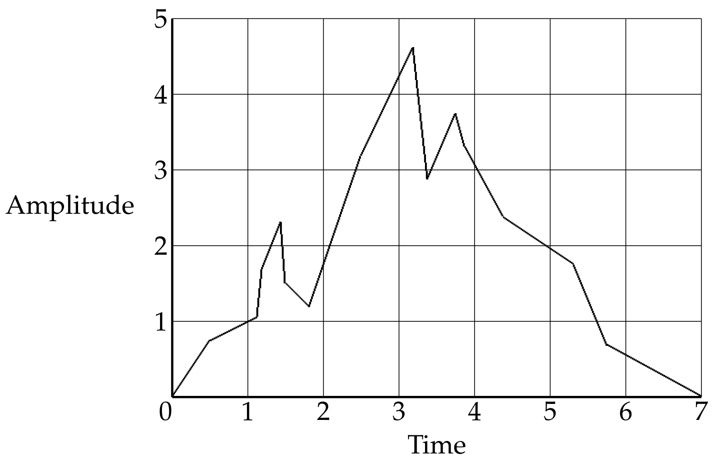
5.2 Applications

Many believe analogue is obsolete, having been superseded by digital. The proliferation of digital computers, digital television, digital cinema, digital radio, digital music, digital cameras, digital camcorders and digital mobile telephony reinforces this viewpoint. However, this belief is unfounded.

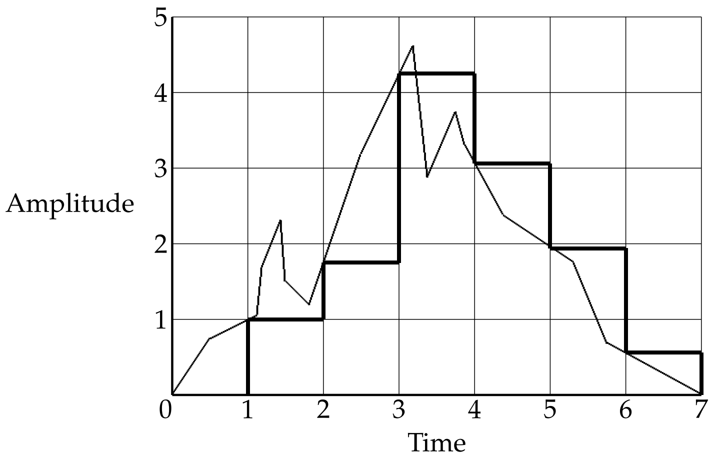
There are many applications where analogue is still regularly used. Some of these applications are discussed below.

5.2.1 Wireless Communications

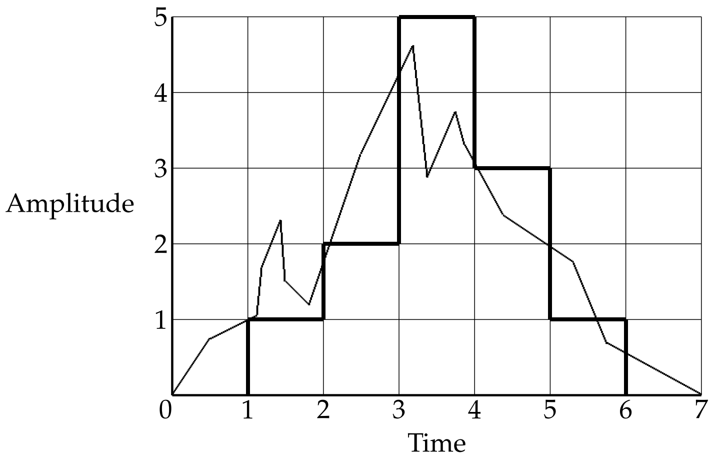
Today, there is substantial enthusiasm for wireless communications and this enthusiasm is constantly growing. Mobile telephones, wireless local area networks (LANs) and wireless broadband all illustrate this trend.



(a) Continuous time analogue signal



(b) Discrete time analogue signal



(c) Digital signal

Figure 5.1: Analogue and digital signals

The signals used for wireless communications are often transmitted close to the 1 GHz frequency range with only a few millivolts of amplitude. They will incur substantial noise during transmission. Therefore, a wireless receiver must amplify the appropriate part of the signal and filter out noise before processing, while operating at a very high speed. This can only be performed in analogue electronics, even for digital communications.

Therefore, these wireless devices typically must include both digital and analogue circuits on the same IC. This has led to a growth of interest in mixed-signal design. New languages have been proposed in order to simplify design, while tools allow for a more automated design flow. This is discussed further in Section 7.1.

5.2.2 Wireline Communications

Broadband technologies such as digital subscriber line (DSL) and cable broadband are analogue-based. These technologies use multiple voltage levels, as opposed to only two digital levels. For example, four levels could be used, sending two merged bits simultaneously. This reduces the bandwidth required. Additionally, to reduce the effects of attenuation and noise, signals are often modulated onto a carrier wave. Both of these techniques result in a conversion of the digital signal to an equivalent analogue one.

However, technologies with a digital basis such as LANs and fibre optic also use analogue processing techniques. In the case of communication over a fibre optic channel, the transmission is made using a laser diode, while the received signal is observed by a photodiode. Both devices are analogue in nature. Moreover, additional analogue processing must be performed to amplify the signal before conversion to digital.

5.2.3 Sensors

Sensors are also inherently analogue in nature. This applies for mechanical, electrical, acoustic and optical sensors. For example, the phototransistors in a digital camera produce analogue signals, which are only converted to digital at a later processing step. A different type of sensor is used to control the airbag release mechanism in vehicles [25, pp. 4–5]. This uses a specially constructed capacitor, which detects the sudden change in velocity indicating that the airbag should be released.

Recent developments in very large scale integration (VLSI) design allow for the sensor's analogue and digital processing elements to be placed on the same IC as the sensor. This increases the level of mixed signal integration, fuelling the need for greater automation of analogue design.

5.2.4 Disk Drives

Disk drives read and write digital data to disks. When this data is read from the disk, the result is a high noise, low amplitude signal. This requires analogue processing involving amplification and filtering. Finally, conversion to a digital format is performed. These actions become more challenging as the speed of disks increases.

5.2.5 Microprocessors and Memories

Microprocessors and memories are digital. However, analogue issues arise in the design of these devices. For example, analysis of the distribution of high speed signals requires these signals to be treated as if they were analogue. Non-idealities in the device such as parasitic resistances and capacitances require knowledge of analogue design. In addition, the high speed sense amplifiers in memories are analogue circuits. Based on this, Razavi claims that "High-speed digital design is in fact analog design." [25, p. 5]

5.2.6 Game Controllers

A game controller is a device used to control a computer game. They were originally entirely digital devices, but they are gradually gaining more analogue components.

Joysticks were the first devices to see this transition. Digital joysticks typically had four or eight possible directions, corresponding to the compass points. In contrast, analogue joysticks have a much greater number of possible directions and can determine the force with which they are being directed. However, there are a finite number of possible directions as the analogue output of the joystick is later converted to digital.

Buttons on game controllers are also making the transition to analogue. Previously, buttons were either fully depressed or unpressed. Analogue buttons, in contrast, can determine the level to which they are depressed.

By transitioning game controllers to analogue, game developers are empowered to make more immersive computer games that respond more precisely to the user's wishes.

5.2.7 Zoom

Camcorders and digital cameras typically provide both an analogue optical and digital zoom. The analogue zoom manipulates the movement of a lens, which controls the magnification of the photograph by modifying the light that hits the charge coupled device (CCD) or complementary metal oxide semiconductor (CMOS) sensor. In contrast, the digital zoom uses the centre portion of the light hitting the CCD or CMOS sensor, which is then interpolated to create a full size image. The analogue zoom acquires new data whereas the digital zoom works with existing data. Therefore, the analogue zoom allows for greater quality photographs. The digital zoom typically provides lower quality photographs.

Chapter 6

Operational Amplifiers

This chapter provides an overview of operational amplifiers, which are the core elements of the analogue computer constructed as part of the project's physics engine. Operational amplifier characteristics are briefly summarised before key operational amplifier circuits are demonstrated.

6.1 Overview

The operational amplifier or "op amp" is the core component of analogue computers. The operational amplifier is an amplifier with a very high open loop gain and a very low output impedance. It can be wired up with auxiliary passive components, which cause the operational amplifier to perform a specific mathematical function. The relation of the output signal to the input signal is determined solely by the arrangement and magnitude of the other circuit elements.

6.2 History

In the early 1940s, George A Philbrick developed the first operational amplifier using vacuum tubes [26, p. 541]. Later, in 1962, Burr-Brown Corp and GAP/R developed the first IC-based operational amplifiers. However, in 1963, Fairchild Semiconductor's μA702 became the first commercially available IC-based operational amplifier.

The best selling operational amplifier of all time is the 741. It was the first internally compensated operational amplifier available, meaning it required no external compensatory components. Released in 1968, it was invented by Dave Fullagar while working at Fairchild Semiconductor. The 741 is still manufactured by a number of companies, including National Semiconductor [27] and Texas Instruments [28], and has remained popular to this day. However, many operational amplifiers have since surpassed the 741's characteristics, exploiting developments in semiconductor fabrication technologies.

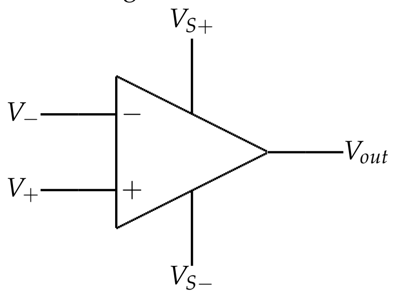
Immediately following their introduction, operational amplifiers were a commercial success. In fact, they proved so popular that the commonly accepted analogue design rules were reformulated to accommodate these devices. Their success is largely attributable to the availability of high performance amplifiers in discrete component form [29, p. 2]. To a large extent, the intricacies of an operational amplifier do not need to be comprehended. As a result, they may be treated as a black box, greatly simplifying circuit design.

6.3 Terminals

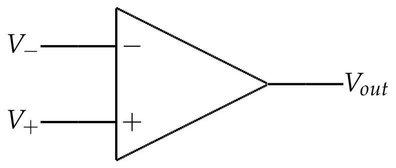
An operational amplifier typically has five terminals, as shown in Figure 6.1a:

V_+	noninverting input
V_-	inverting input
V_{out}	output
V_{S+}	positive power supply
V_{S-}	negative power supply

V_{S+} and V_{S-} must always be connected to appropriate power supplies, typically $+15$ V and -15 V. Therefore, circuit schematics usually eliminate these terminals, creating the circuit symbol depicted in Figure 6.1b. For all practical purposes, V_- will be connected to ground.



(a) Complete



(b) Simplified

Figure 6.1: Operational amplifier circuit symbols

V_- and V_{out} are used to control the mathematical operation performed by the operational amplifier. One component is typically placed in front of V_-. Another is placed on the feedback loop created by connecting V_- to V_{out}. Such a configuration is illustrated in Figure 6.2.

More sophisticated operational amplifiers have additional terminals, but the aforementioned terminals are the only ones germane to the system being constructed.

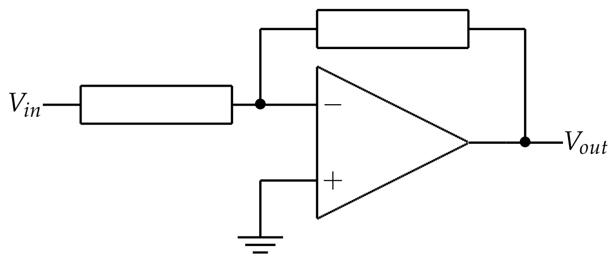


Figure 6.2: Typical operational amplifier configuration. The boxes illustrate where circuit elements may be inserted.

6.4 Ideal Operational Amplifier

An ideal operational amplifier exhibits five main characteristics [29, p. 3]:

Infinite open loop gain The amplification from input to output with no feedback applied is infinite. This makes the performance entirely dependent on input and feedback networks.

Infinite input impedance The impedance viewed from the two input terminals is infinite. This means that no current will flow in or out of either input terminal.

Infinite bandwidth The bandwidth range extends from zero to infinity. This ensures zero response time, no phase change with frequency and a response to direct current (DC) signals.

Zero output impedance The impedance viewed from the output terminal with respect to ground is zero. This ensures that the amplifier produces the same output voltage irrespective of the current drawn into the load.

Zero voltage and current offset This guarantees that when the input signal voltage is zero the output signal will also be zero, regardless of the input source impedance.

These characteristics imply a number of useful properties that may be exploited when constructing operational amplifier circuits [21, pp. 9–12]:

Differential input An operational amplifier has a differential input. In other words, it only responds to the difference between V_+ and V_-. The value of $V_+ - V_-$ is essentially the input signal of the operational amplifier.

AC and DC An operational amplifier can process both alternating current (AC) and DC signals.

Polarity The polarity of the output voltage depends on the polarities of the input voltages.

6.5 Practical Operational Amplifier

The practical operational amplifier closely approximates the behaviour of the ideal operational amplifier. It deviates from the ideal in a number of details. Most of these may effectively be ignored and the analogue circuit designer can treat the operational amplifier as a black box. However, some of these details have an impact on the design of the project's physics engine and must be considered. The most relevant of these are:

Finite bandwidth The range of frequency that may be inputted to the operational amplifier is limited. This partially limits the range of signals that may be processed by the project's physics engine.

Finite open loop gain The maximum amplification provided by the operational amplifier is limited by the magnitude of the supply voltages. This results in upper and lower bounds being placed on the voltages that may be processed by the project's physics engine.

In addition, it must be noted that some values of passive components will not work in the fashion suggested by the black box model. The values of these components must be within certain ranges if their behaviour is to coincide with that required by the operational amplifier.

Another deviation from ideality is that operational amplifiers typically have a voltage offset. Even when their inputs are zero they generate a small output voltage, which changes over time. This problem is easily overcome as many operational amplifiers include terminals to tune the amplifier.

6.6 Circuits

A vast number of circuits may be constructing using an operational amplifier at their core. The operational amplifier circuits relevant to the project's physics engine are described below.

6.6.1 Inverter

The inverter is the basic operational amplifier configuration. This circuit can be constructed by placing a resistor at the V_- input and another resistor on the feedback loop of an operational amplifier, as illustrated in Figure 6.3. The mathematical function performed by the circuit is:

$$V_{out} = -V_{in}\left(\frac{R_f}{R_{in}}\right)$$

where

V_{out}	$=$	Voltage at V_{out} terminal
V_{in}	$=$	Voltage at V_{in} terminal
R_f	$=$	Resistance of R_f resistor
R_{in}	$=$	Resistance of R_{in} resistor

To perform inversion, R_f must be equal to R_{in}.

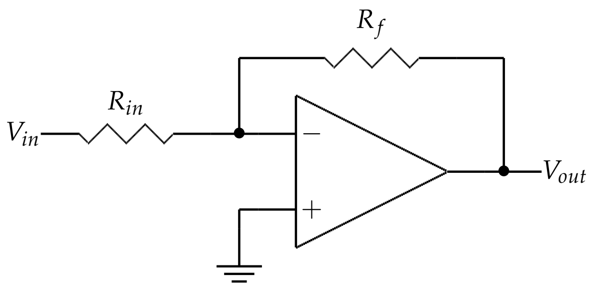


Figure 6.3: Inverter

6.6.2 Multiplier and Divider

By choosing the values of R_f and R_{in} appropriately, the inverter circuit (Section 6.6.1) may be used to perform multiplication or division by a constant. To construct a multiplier, R_f should be set greater than R_{in}. To construct a divider, R_f should be set less than R_{in}.

6.6.3 Adder

The adder can potentially sum infinite sources, unlike its digital equivalent. Each source is allocated an input line, connected to the operational amplifier's V_- terminal through a resistor. A resistor is placed on the feedback loop. This configuration is illustrated in Figure 6.4. The mathematical function performed by the circuit is:

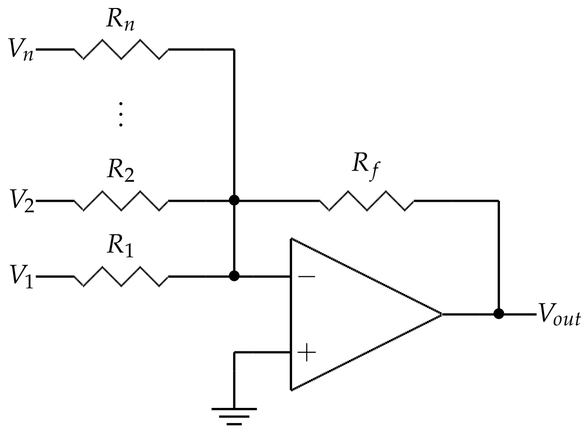


Figure 6.4: Adder

$$V_{out} = -R_f \left(\frac{V_1}{R_1} + \frac{V_2}{R_2} + \cdots + \frac{V_n}{R_n} \right)$$

where

V_{out}	=	Voltage at V_{out} terminal
R_f	=	Resistance of R_f resistor
V_1	=	Voltage at V_1 terminal
R_1	=	Resistance of R_1 resistor
V_2	=	Voltage at V_2 terminal
R_2	=	Resistance of R_2 resistor
V_n	=	Voltage at V_n terminal
R_n	=	Resistance of R_n resistor

As with the inverter, multiplication and division may be performed by weighting the resistor values appropriately. Each input may be multiplied or divided by a different value.

The disadvantage associated with this configuration is that the output is inverted, so pure addition would require the use of both an adder and an inverter.

6.6.4 Integrator

The integrator integrates the input signal with respect to time. It is constructed by placing a resistor at the V_- input and a capacitor on the feedback loop of an operational amplifier, as illustrated in Figure 6.5. The mathematical function performed by the circuit is:

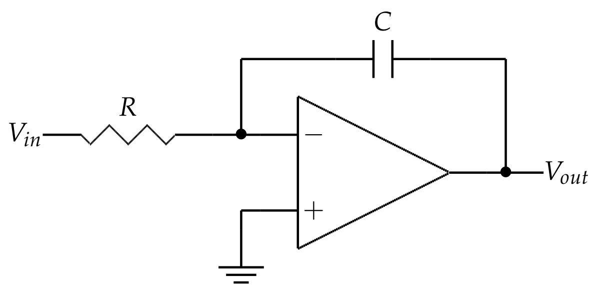


Figure 6.5: Integrator

$$V_{out} = -\frac{1}{RC}\int_0^t V_{in}\,\mathrm{d}t$$

where

V_{out}	=	Voltage at V_{out} terminal
R	=	Resistance of R resistor
C	=	Capacitance of C capacitor
V_{in}	=	Voltage at V_{in} terminal
t	=	Time

Again, by appropriately weighting the values of the resistor and capacitor, multiplication and division may also be performed by this circuit. Furthermore, the output is inverted by this operation.

6.6.5 Differentiator

The differentiator differentiates the input signal with respect to time. It is constructed by placing a capacitor at the V_- input and a resistor on the feedback loop of an operational amplifier, as illustrated in Figure 6.6a. The mathematical function performed by the circuit is:

$$V_{out} = -RC\frac{\mathrm{d}V_{in}}{\mathrm{d}t}$$

where

$$\begin{aligned} V_{out} &= \text{Voltage at } V_{out} \text{ terminal} \\ R &= \text{Resistance of } R \text{ resistor} \\ C &= \text{Capacitance of } C \text{ capacitor} \\ V_{in} &= \text{Voltage at } V_{in} \text{ terminal} \\ t &= \text{Time} \end{aligned}$$

This circuit may also perform multiplication and division if the values of the resistor and capacitor are appropriately weighted. As with the other circuits, the output is inverted by this operation.

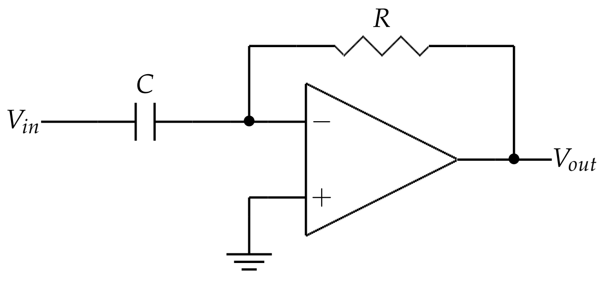
However, the differentiator is a problematic analogue computer element. The differentiation process strongly accentuates noise, which is always present in analogue electronic circuits. Noise tends to have sudden abrupt changes, appearing as “voltage spikes”. Since the output of a differentiator is proportional to the rate of change of its input, these sudden changes are greatly amplified. Therefore, using the differentiator in its current incarnation would result in very poor performance by the analogue computer.

In order to rectify this problem, a circuit referred to as the practical or low frequency differentiator is often constructed instead. This consists of the previous differentiator circuit with a resistor placed before the input capacitor, as illustrated in Figure 6.6b. The mathematical function of the circuit remains as before. The frequency range over which this device operates is determined by R_{in}. This works because a capacitor is essentially a short circuit at high frequencies, converting the circuit to an inverter. Furthermore, an appropriate value of R_{in} will be overshadowed by the capacitance at lower frequencies.

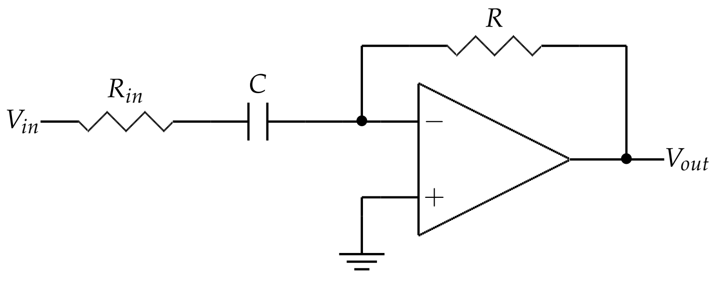
While this circuit performs its stated function correctly, it introduces an additional problem. Analogue computers have traditionally been constructed for a particular experiment with the inputs having a known frequency range. However, the project’s physics engine had to simulate an extensive range of problems. Limiting the input frequency to the operational amplifiers would have substantially reduced this flexibility. Therefore, an alternative solution was desirable.

The standard solution involves reformulating the problem definition, so that differentiation is transformed into integration. This can be performed by using the following formula:

$$\begin{aligned} y &= a\frac{\mathrm{d}x}{\mathrm{d}t} \\ \Rightarrow x &= \frac{1}{a}\int y\,\mathrm{d}t \end{aligned}$$



(a) Theoretical



(b) Practical

Figure 6.6: Differentiator

where

y	$=$	Variable
a	$=$	Constant
x	$=$	Variable
t	$=$	Time

After this reformulation, the standard unproblematic integrator may be used and all differentiators may be eliminated, solving all of the aforementioned problems.

6.6.6 Logarithm Calculator

The logarithm calculator is constructed by placing a resistor at the V_- input and a diode on the feedback loop of an operational amplifier, as illustrated in Figure 6.7. For a PN diode:

$$I = I_o \left(e^{KV} - 1\right) \tag{6.1}$$

where

I	$=$	Current through diode
I_o	$=$	Saturation current
e	$=$	Euler's constant
K	$=$	Variable
V	$=$	Voltage across diode

The mathematical function performed by the circuit is:

$$V_{out} = -\frac{1}{K} \ln \frac{V_{in}}{RI_o}$$

where

V_{out}	$=$	Voltage at V_{out} terminal
K	$=$	Variable
V_{in}	$=$	Voltage at V_{in} terminal
R	$=$	Resistance of R resistor
I_o	$=$	Saturation current

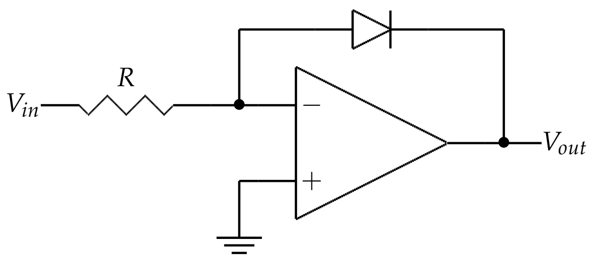


Figure 6.7: Logarithm calculator

The functionality of this circuit is dependent on the characteristics of the diode. Practical diodes do not exhibit the theoretical characteristics suggested by Equation 6.1, as a diode's behaviour depends on temperature. Consequently, this is not an ideal logarithm calculator.

Based on these complications and the observation that logarithm is a rarely used function in physics, it was decided not to use these components in the project's physics engine.

6.6.7 Antilogarithm Calculator

The antilogarithm calculator works in reverse to the logarithm calculator (Section 6.6.6), due to the presence of a diode (Figure 6.8). The same problems apply to this as to the logarithm calculator. Therefore, it was also decided not to use these components in the project's physics engine.

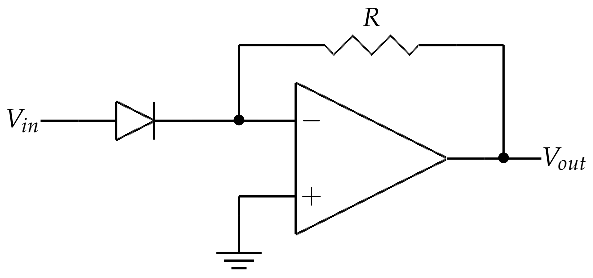


Figure 6.8: Antilogarithm calculator

6.6.8 Further Operations

Many other circuits may be constructed using operational amplifiers, such as an amplifier, difference amplifier, voltage follower, comparator, Schmitt trigger and inductance gyrator. However, these are of little use for analogue computers. Therefore, these circuits are not discussed in this report.

Part II

Design and Implementation

Chapter 7

Implementation Background

This chapter outlines some decisions that were made before implementation proceeded, such as the tools and software used to implement the project. It also highlights how multiple abstraction levels and testbenches were utilised during the implementation.

7.1 VHDL-AMS

The traditional means for designing analogue and mixed signal circuits is schematic entry or schematic capture. This involves creating a circuit schematic by interconnecting components. It is a relatively intuitive way of constructing circuits.

For digital design, schematic entry could also be used. However, hardware description languages (HDLs) effectively replaced digital schematic entry. Schematic entry provides only a structural level of abstraction, where the components and their interconnections are made explicit. In contrast, HDLs often provide multiple levels of abstraction, which provides flexibility to describe each entity at its most natural abstraction level. In addition, searching and reuse are easier with HDLs, as the graphical natural of schematics hinders both of these tasks.

Until recently, there were few analogue and mixed signal HDLs. This situation changed in 1999, when the Institute of Electrical and Electronics Engineers (IEEE) standardised VHDL-AMS [30], the analogue and mixed signal extensions to digital VHDL. There is currently considerable interest in VHDL-AMS, although few companies have started using it. This interest is attributable to the continual growth in integration density, which makes it is no longer necessary to split the digital and analogue parts of a design onto different ICs [31, p. 199]. It is probable that in the future, it will be possible to fabricate an IC based on a VHDL-AMS description. Currently, this is an area of active research [32, 33]. Today, many electronic design automation (EDA) tools that previously supported VHDL have been upgraded to support VHDL-AMS.

A competing language is Verilog-AMS, which is based on digital Verilog. This has received less support as the standard has not yet been finalised by the IEEE.

For this project, it was decided to use VHDL-AMS, as its standardisation provides a vendor neutrality impossible with schematic entry. Vendor neutrality is becoming increasingly important due to the emphasis on interoperability between heterogeneous systems. Moreover, the use of a single language for the digital and analogue parts of the design simplifies implementation. Finally, multiple abstraction levels, searchability and reusability are all advantageous to this project.

7.2 Software

Mentor Graphics' SystemVision and Xilinx ISE were the two software applications used for implementing this project. These two applications are described below.

7.2.1 Mentor Graphics' SystemVision

Mentor Graphics Corp is one of the major players in the EDA industry. It provides a multitude of software products, spanning the entire EDA spectrum.

For this project, Mentor Graphics' SystemVision was used for creating the VHDL-AMS descriptions. An associated program, Waveform Viewer, was used to observe the outputs of the VHDL-AMS simulations.

SystemVision supports VHDL-AMS, SPICE, C and schematic entry. The application allows for pure VHDL-AMS descriptions without vendor specific extensions. SystemVision supports only a subset of the VHDL-AMS standard, but conformance is increasing with each version of the application. The most important parts of the standard have been implemented. The version of the application used for this project was 2002, although a 2003 version was available. The 2002 version was used, as it was the only version available to the college during the project.

7.2.2 Xilinx ISE

Xilinx is the world leader in field programmable gate arrays (FPGAs). In addition to manufacturing the devices, it also provides the software required to program and analyse the devices.

For this project, Xilinx ISE was used for creating the pure digital VHDL descriptions. ModelSim was used to simulate these descriptions.

Xilinx ISE supports VHDL, Verilog and EDIF. The application can be used to check the synthesisability of VHDL designs. This process determines if the design may be fabricated. SystemVision does not check for synthesisability as analogue and mixed signals designs are not currently synthesisable. The version of this application used for the project was 6.2i.

7.3 Behavioural and Structural

In VHDL-AMS, the two principal levels of abstraction at which descriptions may be written are behavioural and structural. Behavioural code describes the behaviour of a system at a high level of abstraction. For analogue hardware, this usually means a differential equation defining the behaviour of the hardware. Structural code describes the structure of a system or the hierarchy of components from which it has been constructed. In VHDL-AMS, this involves the use of port maps.

Behavioural descriptions have been provided for all the components described in the following chapters. These descriptions are used as reference models since they describe idealistic components. The structural description may later be compared against this ideal model.

In the project, all non-core components have also been provided with a structural definition. Core components are components such as resistors and capacitors, which can have no valid structural definition. The structural definition is the most important as it describes accurately how the system would be constructed and how it would behave.

In summation, behavioural and structural definitions were created for almost all components. Behavioural descriptions are optional but greatly assist in the debugging and validation of designs.

7.4 Testbenches

For the project, testbenches have been constructed to ensure the correct behaviour of every component. Some components have been provided with multiple testbenches where it was deemed beneficial for either testing or comprehension.

The same testbenches are used for both the behavioural and structural descriptions. This allows for deviations between the two descriptions to be more easily analysed. The behavioural definition is typically the idealistic component against which deviations in the structural definition should be measured.

In a digital system, it is often desirable to run several test cases for each component in order to verify that it works as stated, for every possible scenario. It is plausible that a design could be 100% tested. However, this is not possible for most analogue designs. The range of input values to an analogue design is potentially infinite compared to a maximum of two for digital. Moreover, analogue designs can be more easily affected by temperature or process variations so that testbenches can never fully test a real design. Considering these complexities, the created testbenches test some standard cases, which demonstrate the standard behaviour of the components. It was deemed superfluous to stress test the designs because temperature and process variations have not been considered.

Chapter 8

Prototype 1: Mass-Spring-Damper

The chapter outlines the first prototype constructed for the project. Firstly, information is provided on the physics underlying the system. This is followed with information on the design and construction of the individual entities constituting the prototype. The chapter is concluded with a discussion of the implemented prototype and analyses the deviation of the device from ideality, to check the viability of this project.

8.1 Background

In order to gain familiarity with the design and construction of analogue computers, it was decided to first implement a prototype. In addition, this prototype served as an introduction to VHDL-AMS (Section 7.1) and SystemVision (Section 7.2.1). Such a prototype needed to be a pure analogue non-hybrid computer. In other words, it would not feature reconfigurable elements. Moreover, it needed to implement a single relatively simple equation that would be of practical use.

Based on these criteria, the mass-spring-damper system, one of the simplest problems in classical Newtonian physics, was chosen for the prototype. Before discussing the mass-spring-damper system, the simpler mass-spring system is described.

8.1.1 Physics of Mass-Spring

The mass-spring system consists of a mass suspended on the end of a spring. At the opposite end of the spring is a fixed or relatively fixed object to which the spring is attached. This system is illustrated in Figure 8.1.

If the fixed object is disturbed, the mass attached to the spring will oscillate perpetually with a constant amplitude, creating a sine wave, as illustrated in Figure 8.2.

However, such a system is only theoretically possible; it cannot exist in reality. Various forces such as air resistance and internal resistances of the spring

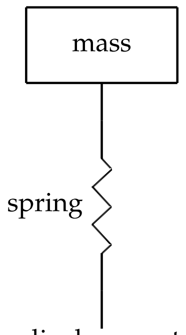


Figure 8.1: Mass-spring

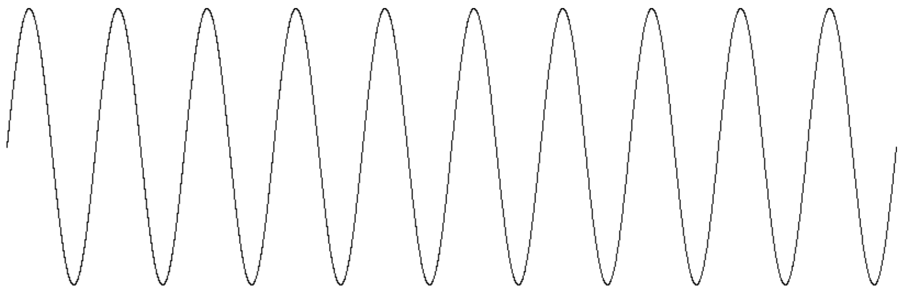


Figure 8.2: Behaviour of a mass-spring

will result in the system's oscillation being damped over time. Consequently, from the point of view of a physics engine, this model is unrealistic and of little use. A more developed system that models the damping was necessary.

8.1.2 Physics of Mass-Spring-Damper

The mass-spring-damper system is a development of the mass-spring system. It consists of the mass-spring system with a damper connected in parallel with the spring, as illustrated in Figure 8.3.

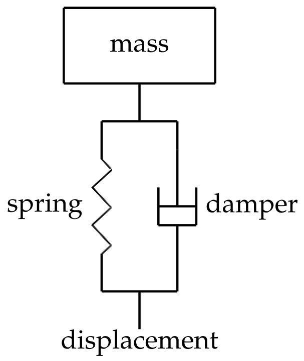


Figure 8.3: Mass-spring-damper

If the fixed object is disturbed, the mass attached to the spring will oscillate. However, it will not oscillate perpetually, since the damper will impede the motion of the spring. Instead, it will oscillate with a wave whose amplitude starts at a large value and gradually decreases until it reaches zero. This behaviour is illustrated in Figure 8.4.

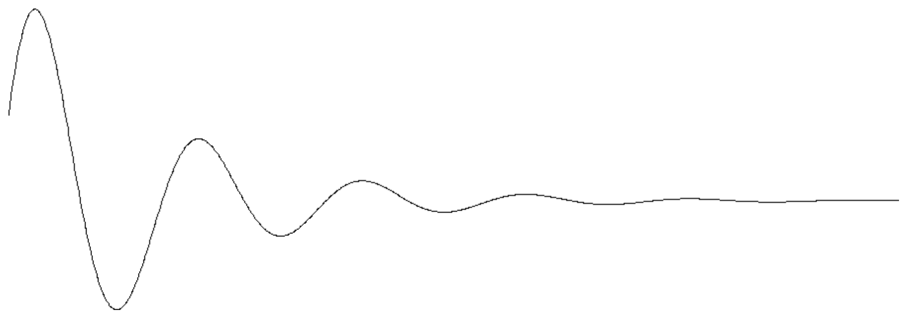


Figure 8.4: Behaviour of a mass-spring-damper

This system is feasible, since the damper may be constructed to model the forces inherent in any assembled mass-spring system. In addition, this system is used for many purposes. In particular, vehicle suspension systems are often based on some derivative of this system. This aspect will be discussed further in Chapter 9.

8.2 Voltage Sources

VHDL-AMS testbenches for analogue or mixed signal entities typically consist of the device under test (DUT) and one or more voltage sources. Since there were many analogue or mixed signal testbenches, it was necessary to construct voltage sources, before proceeding with the construction of the physics engine. Only behavioural level descriptions were created for these sources. Structural level descriptions were not created, since voltage sources are standard entities and their construction details are unimportant.

8.2.1 Voltage Source

A voltage source provides a potential difference.

The project's voltage source can be used to generate a square waveform or any waveform with square edges. It is instantiated with an array of times and an array of voltages. After the duration specified by the first element in the time array elapses, the voltage specified by the first element in the voltage array is outputted. This procedure is repeated for each element in the arrays.

The output voltage is initialised to 0 V. Inside a process, a loop iterates through the arrays. This loop waits for the duration specified by the first element. Afterwards, the output voltage is changed to that specified by the first element. The loop continues until every element of the array has been processed. The final voltage will be retained indefinitely. This code is illustrated in Code 8.1.

```vhdl
process
begin
  for i in amplitudes'range loop
    wait for times(i);
    voltage_output_duplicate <= amplitudes(i);
  end loop;
  wait;
end process;
```

Code 8.1: Excerpt from `Behavioural/voltage_source.vhd`

Instead of supplying the entity with arrays during instantiation, it would have been more flexible and more natural to supply it with a filename whose contents contained the values of the arrays. However, such a design was impossible as SystemVision 2002 (Section 7.2.1) does not support the VHDL file input/output (I/O) packages (Section 12.3).

The testbench requests the voltage source to output a stepped waveform, to verify that a range of values may be obtained from the entity.

8.2.2 Sinusoidal Voltage Source

A sinusoidal voltage source provides an AC potential difference in the form of a sine wave.

The previously described voltage source (Section 8.2.1) could be used to generate sinusoidal waveforms. However, doing so would require calculating the precise voltage at numerous timesteps, which would involve substantial effort. Moreover, these waveforms would be of poor quality, only approximating real sinusoidal waveforms. However, sinusoidal waveforms were essential to fully test many of the project's entities. To resolve these issues, a sinusoidal voltage source was constructed.

The project's sinusoidal voltage source is instantiated with the frequency and amplitude of the desired waveform.

A sine wave may be described using a simple formula:

$$x(t) = a \sin(\pi f t)$$

where

$$\begin{aligned} x &= \text{Displacement} \\ t &= \text{Time} \\ a &= \text{Amplitude} \\ f &= \text{Frequency} \end{aligned}$$

This formula was mapped into the code shown in Code 8.2.

```
  voltage == amplitude * sin(math_2_pi * frequency * now)
     ;
```

Code 8.2: Excerpt from `Behavioural/sinusoidal_voltage_source.vhd`

The testbench requests the sinusoidal voltage source to output a sinusoidal waveform.

8.3 Packages

Many functions were used repeatedly throughout the project. Following the principles of modular design, these functions were placed inside four packages.

8.3.1 Arithmetic

This package provides a function for the addition of two `std_ulogic_vectors`, since the packages supplied with SystemVision 2002 (Section 7.2.1) do not contain such a function. This addition is done by converting the `std_ulogic_vectors` to `std_logic_vectors`, summing them and converting the result back to an `std_ulogic_vector`. Although perhaps slightly inefficient, it utilises the standard packages to implement the functionality, which results in a reduced likelihood of errors, as the supplied functions have presumably been fully tested. Moreover, this source of inefficiency is of little consequence, as this function will only be used during simulations. The function would be unnecessary in synthesised code since hardware has no concept of types.

This package also provides a function for testing two real numbers for equality. The two numbers are said to be equal if they lie within a certain range of each another. This is necessary because, due to various sources of errors in floating point arithmetic, it is nearly impossible to get two "equal" real numbers to have exactly the same value.

The testbench checks the validity of this package using assertions.

8.3.2 Constants

This package defines constants that were used throughout the rest of the code. These will be discussed in further detail when appropriate (Section 8.5.2 and Section 10.1).

The testbench checks the validity of this package using assertions.

8.3.3 Type Conversion

A large amount of type conversion was performed in many entities since all of the project's digital signals are either of type `std_ulogic` or `std_ulogic_vector` instead of the more common `std_logic` or `std_logic_vector`. These "unresolved" types were used since they can only have a single driver. They are recommended in preference to their resolved counterparts in cases where there should be no more than one driver, since the simulator will warn if this is untrue.

This package provides a function for converting an `integer` to an `std_ulogic_vector`. This function works by performing a sequence of conversions, through different types, in order to utilise inbuilt functions. The package also provides a function for performing the opposite conversion, from an `std_ulogic_vector` to an `integer`. Again, this function works by performing a sequence of conversions.

In a similar fashion, this package provides functions for converting a `real` to an `std_ulogic_vector` and vice versa.

Finally, this package provides a function for checking if an `std_ulogic_vector` contains any unknown bits. This function is utilised by the conversion functions, as attempting to convert an `std_ulogic_vector` with unknown bits will result in errors. This function is typically provided in standard libraries, but the necessary functions were unavailable in SystemVision 2002.

The testbench checks the validity of this package using assertions.

8.3.4 Types

This package defines a new type, `time_vector`, which is an array of time objects. This type is awaiting IEEE approval for addition to the standard libraries.

The testbench only checks if the creation and initialisation of such an object is valid.

8.4 Core Entities

For the purposes of this project, core entities are the basic entities used to construct more complex entities. These entities are resistors, capacitors and operational amplifiers. All of these entities were only described at the behavioural level. As they are standard entities, their implementation details are unimportant. Therefore, no structural level description was necessary. Further, resistors and capacitors can have no valid structural level description, as they cannot be constructed from other entities.

8.4.1 Resistor

A resistor impedes the flow of current.

The resistance of the project's resistor is specified at instantiation, making it somewhat variable.

The behaviour of an ideal resistor can be described using Ohm's Law:

$$V = IR$$

where

V	$=$	Voltage across resistor
I	$=$	Current through resistor
R	$=$	Resistance

While this equation describes an ideal resistor, the majority of practical resistors closely approximate this ideal. Therefore, this equation provides a satisfactory description for the project. It was translated into the code shown in Code 8.3.

```
voltage == current * res;
```

Code 8.3: Excerpt from `Behavioural/resistor.vhd`

The testbench attaches a sinusoidal voltage source to one terminal of the resistor, so that the voltage at the other terminal may be observed. A single resistor wired in this fashion should not modify the inputted voltage. Consequently, the output is the same as the input.

8.4.2 Capacitor

A capacitor stores a quantity of electrical energy.

The capacitance of the project's capacitor is specified at instantiation, making it somewhat variable.

The behaviour of an ideal capacitor can be described by the capacitor equation:

$$I = q\frac{\mathrm{d}V}{\mathrm{d}t}$$

where

I	$=$	Current through capacitor
q	$=$	Charge
V	$=$	Voltage across capacitor
t	$=$	Time

While this equation describes an ideal capacitor, the majority of practical capacitors closely approximate this ideal. Therefore, this equation provides a satisfactory description for the project. It was translated into the code shown in Code 8.4.

```
current == cap * voltage'dot;
```

Code 8.4: Excerpt from `Behavioural/capacitor.vhd`

The testbench attaches a sinusoidal voltage source to one terminal of the capacitor, so that the voltage at the other terminal may be observed. A single capacitor wired in this fashion will slightly modify the input voltage while it is charging. After a short period of time, the voltage will become identical to the input voltage.

8.4.3 Operational Amplifier

An operational amplifier is an amplifier that is capable of performing mathematical operations. It is described in detail in Chapter 6.

The project's operational amplifier has three terminals, corresponding to V_+, V_- and V_{out}. The other terminals were deemed unnecessary to the core functionality of the entity and were consequently omitted.

As outlined in Section 6.5, despite continual improvements, operational amplifiers cannot achieve ideality. Therefore, an ideal model of the operational amplifier would be insufficient. Consequently, the operational amplifier was described by a simple mathematical model, which took into account a number of deviations. Some deviations that only affect operational amplifiers while working outside their normal operating ranges were not modelled. These simplifications were deemed acceptable as the project's operational amplifiers were only used within their normal operating ranges.

The testbench connects the operational amplifier in a negative feedback configuration, as shown in Figure 8.5. The input voltage is connected to V_+, and V_- forms a feedback loop with V_{out}. In this configuration, the output voltage always matches the input voltage.

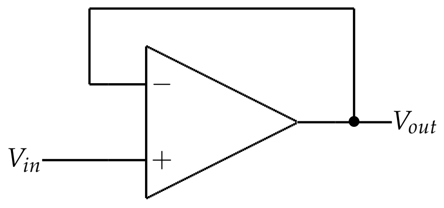


Figure 8.5: Operational amplifier in negative feedback configuration

8.5 Derived Entities

For the purposes of this project, derived entities are the entities constructed from the core entities described in Section 8.4. These entities are described at both the behavioural and structural levels. The behavioural level describes these entities in terms of differential equations, which provide a natural way to model these entities. The structural level describes the construction or assembly of these entities.

8.5.1 Inverter

The inverter was described in detail in Section 6.6.1.

The inverter is the simplest analogue computing entity constructed for this project. The project's inverter has two inputs, each of which is multiplied by a different specified factor. The two multiplied inputs are summed and the result is inverted.

The behavioural level description consists of an equation describing this behaviour, as shown in Code 8.5. The structural level description instantiates three resistors and an operational amplifier, connecting them in the configuration shown in Figure 8.6. The corresponding code is shown in Code 8.6

```
voltage == -((weight1 * input1'reference) + (weight2 *
    input2'reference));
```

Code 8.5: Excerpt from `Behavioural/inverter.vhd`

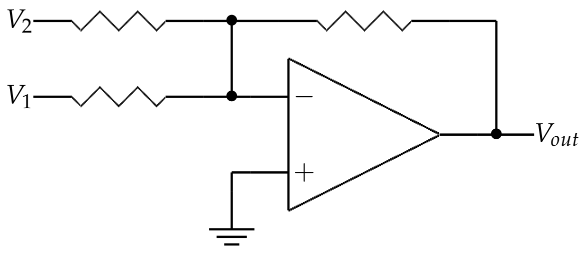
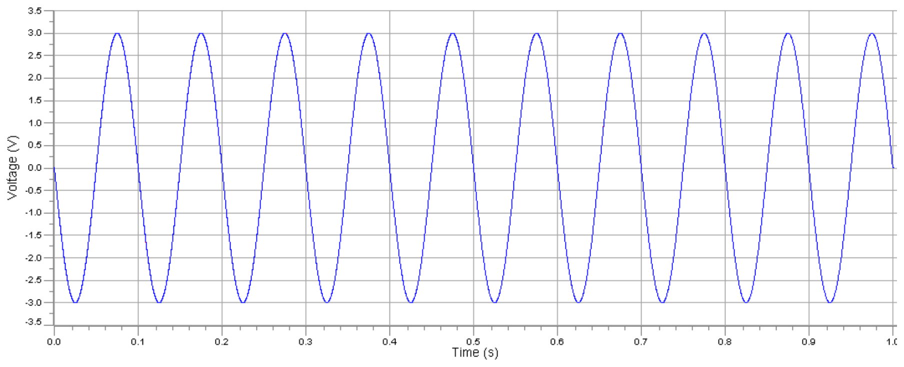


Figure 8.6: Schematic of inverter

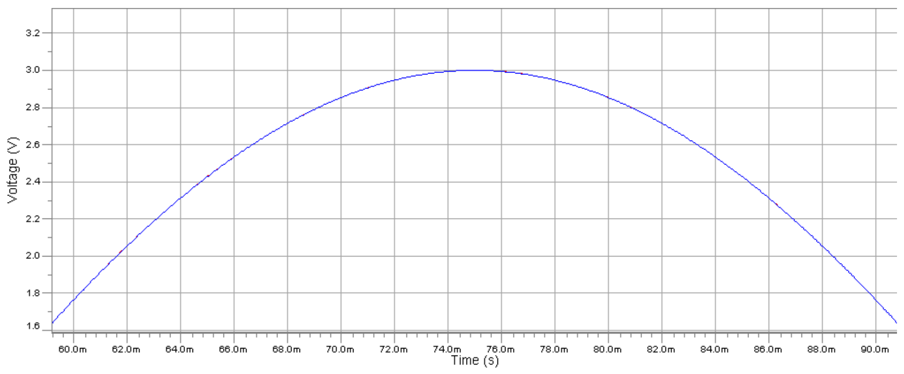
```
resistor1: resistor
  generic map (
    res => resistance1
  )
  port map (
    terminal1 => input1,
    terminal2 => weighted_input
  );
resistor2: resistor
  generic map (
    res => resistance2
  )
  port map (
    terminal1 => input2,
    terminal2 => weighted_input
  );

parallel_resistor: resistor
  generic map (
    res => basic_resistance
  )
  port map (
    terminal1 => weighted_input,
    terminal2 => output
  );
op_amp: operational_amplifier
  port map (
    positive_input => electrical_ref,
    negative_input => weighted_input,
    output => output
  );
```

Code 8.6: Excerpt from `Structural/inverter.vhd`

The testbench supplies the inverter with two sinusoidal waveforms multiplied by different factors and verifies that the output is the inverted sum of these, using assertions. The testbench output shown in Figure 8.7 demonstrates that the deviation of the structural model from ideality is extremely miniscule. The difference may only be observed in the highly magnified waveform. Therefore, it may be assumed that this difference is insignificant and would be invisible to a computer game player, if the inverter were part of a physics engine.



(a) Original



(b) Magnified

Figure 8.7: Comparison of inverter's behavioural and structural descriptions. Red indicates behavioural; blue indicates structural.

8.5.2 Integrator

The integrator was described in detail in Section 6.6.4.

As with the inverter, the project's integrator has two inputs, each of which is multiplied by a different specified factor. The two multiplied inputs are summed and the result is inverted and integrated with respect to time.

The behavioural level description consists of an equation describing this behaviour, as shown in Code 8.7. The structural level description instantiates two resistors, a capacitor and an operational amplifier, connecting them in the configuration shown in Figure 8.8.

```
voltage'dot == -((weight1 * input1'reference) + (
    weight2 * input2'reference));
```

Code 8.7: Excerpt from `Behavioural/integrator.vhd`

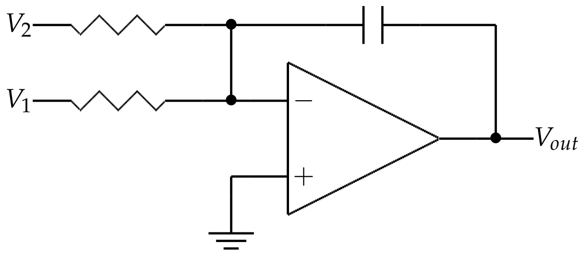
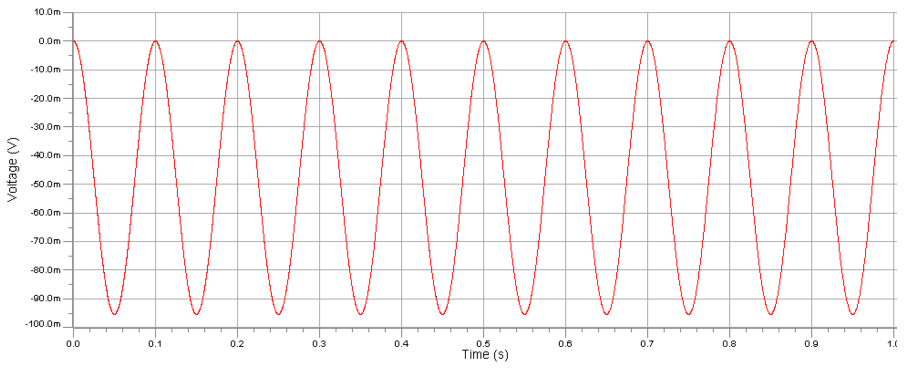


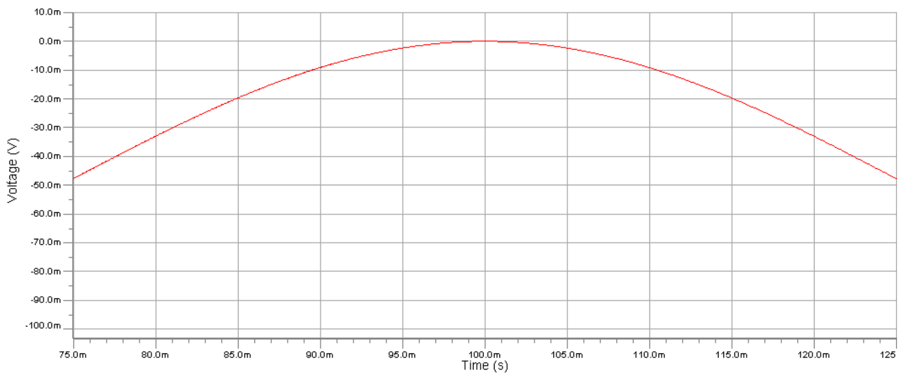
Figure 8.8: Schematic of integrator

A difficulty arose during the construction of this entity regarding the basic quantities that should be assigned to the resistors and capacitors. Empirical evidence suggested that a large range of resistances were suitable, but that the range of capacitances was very limited. In essence, should the value of the capacitors have been too great, the output waveform would be completely distorted. Low capacitances worked well, but if the values were too low, the waveform would also be somewhat distorted. Moreover, it was unsatisfactory to fix the capacitors to some small capacitance and leave the resistors at 1 Ω because this meant a multiplicative factor of one could never be obtained from the circuit. Consequently, a high resistance had to be chosen to counteract the effects of the low capacitance. After repeated experimentation, it appeared that the values specified in the `constants` library, 100 kΩ resistance and 10 μF capacitance, provided the least distorted waveform possible.

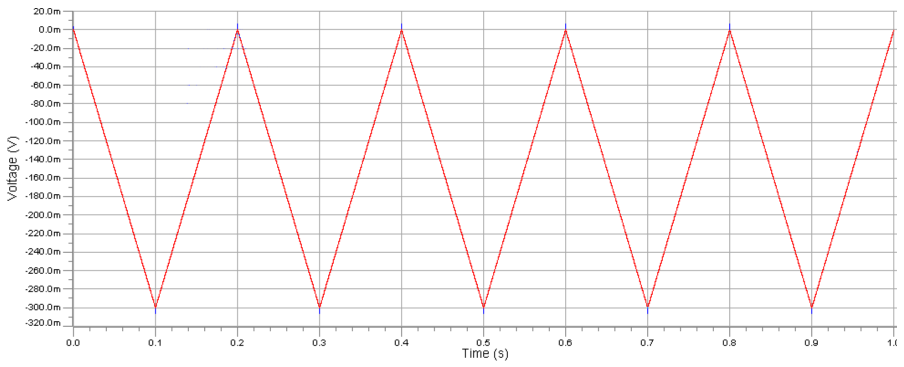
The first testbench supplies the integrator with two sinusoidal waveforms multiplied by different factors. The output is a sinusoidal waveform phase shifted by 90^o or $\frac{\pi}{2}$ rad. The second testbench supplies the equivalent square waveform, creating a triangular waveform output. As illustrated by Figure 8.9, the distortion to a sinusoidal waveform is negligible. Nevertheless, there is minor distortion caused to a square wave input. However, the distortion offered is only 0.03846%. Such minimal distortion is likely to prove impossible to perceive in a computer game using these calculations.



(a) Original, using a sinusoidal waveform

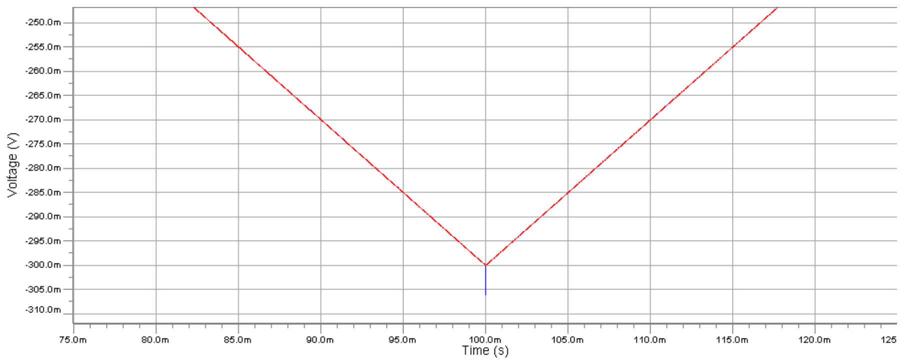


(b) Magnified, using a sinusoidal waveform



(c) Original, using a square waveform

Figure 8.9: Comparison of integrator's behavioural and structural descriptions. Red indicates behavioural; blue indicates structural.



(d) Magnified, using a square waveform

Figure 8.9: Comparison of integrator's behavioural and structural descriptions. Red indicates behavioural; blue indicates structural.

8.5.3 Differentiator

The differentiator was described in detail in Section 6.6.5.

It was noted that use of the differentiator should be avoided in analogue computers due to the high level of noise introduced by these components. In order to study the impact of a differentiator on the project, a differentiator entity was constructed and simulated.

As with the integrator, the project's differentiator has two inputs, each of which is multiplied by a different specified factor. The two multiplied inputs are summed and the result is inverted and differentiated with respect to time.

The behavioural level description consists of an equation describing this behaviour, as shown in Code 8.8. The structural level description instantiates a resistor, two capacitors and an operational amplifier, connecting them in the configuration shown in Figure 8.10. The basic resistances and capacitances used were those that were deemed most appropriate for the integrator.

```
  voltage'integ == -((weight1 * input1'reference) + (
      weight2 * input2'reference));
```

Code 8.8: Excerpt from `Behavioural/differentiator.vhd`

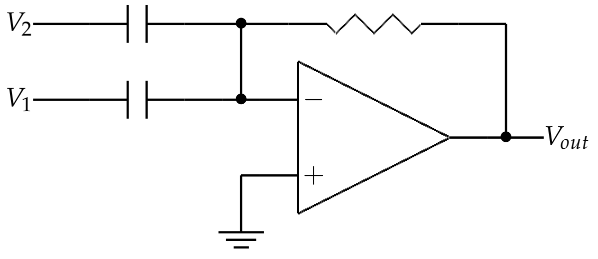
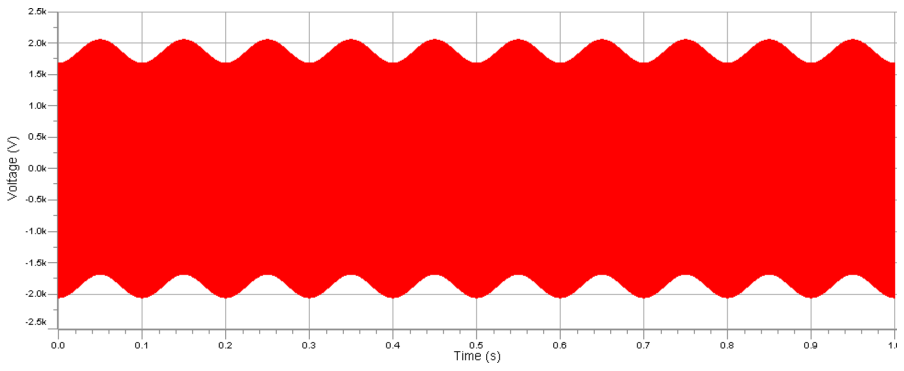
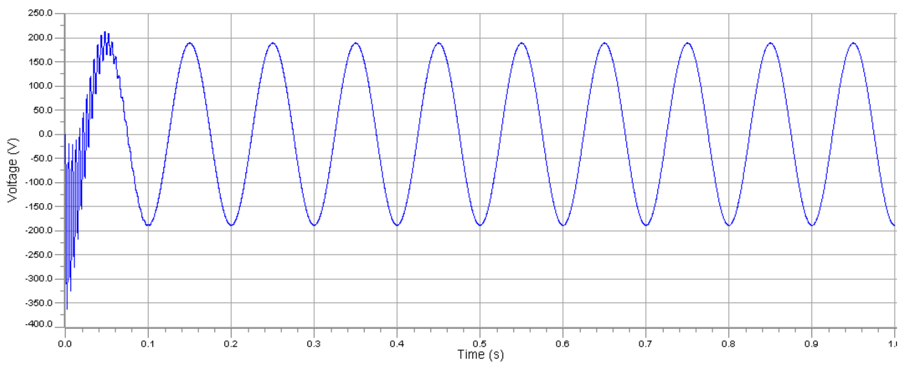


Figure 8.10: Schematic of differentiator

The testbench supplies the differentiator with two sinusoidal waveforms multiplied by different factors. The output should be a sinusoidal waveform phase shifted by 90^o or $\frac{\pi}{2}$ rad. However, this is not the case. As can be seen from Figure 8.11, the output is heavily distorted at both the behavioural and structural levels. In fact, the behavioural level description suffers from heavier distortion, since the nonideality of the operational amplifier serves to distort the output into a signal roughly approximating that expected. However, the lesser distortion is still completely unusable and it appears that, as discussed, the differentiator should be discarded and that it should be substituted with the integrator. The ease of programming offered by differentiators does not outweigh the substantial errors they incur.



(a) Behavioural



(b) Structural

Figure 8.11: Differentiator

8.6 Mass-Spring-Damper

The mass-spring-damper is provided with three parameters, the mass, the spring constant of the spring and the viscosity of the damper. An input displacement is specified and the output displacement and velocity are generated.

The mass-spring-damper uses an input and output displacement. It would have been more common to use a single displacement with the output connected back to the input, as was often done with analogue computers. However, the simulator refused to simulate this system as the input source and looped back output constituted multiple line drivers (Section 12.4). To solve this problem, the dual displacement version of the mass-spring-damper equation was used instead.

The mass-spring-damper is specified by a second order differential equation, which may be used to calculate the position of the mass at any particular point in time. This equation is [34, p. 31]:

$$F = -b\frac{\mathrm{d}\left(x_{out} - x_{in}\right)}{\mathrm{d}t} - k\left(x_{out} - x_{in}\right) \tag{8.1}$$

where

F	=	Force
b	=	Damper's viscosity
x_{out}	=	Displacement out
x_{in}	=	Displacement in
t	=	Time
k	=	Spring constant

This equation could not be implemented directly, as differentiation had to be replaced by integration:

$$\begin{aligned}
F &= -b\frac{\mathrm{d}\left(x_{out} - x_{in}\right)}{\mathrm{d}t} - k\left(x_{out} - x_{in}\right) \\
\text{But } F &= ma \\
\therefore ma &= -b\frac{\mathrm{d}\left(x_{out} - x_{in}\right)}{\mathrm{d}t} - k\left(x_{out} - x_{in}\right) \\
\text{But } a &= \frac{\mathrm{d}V}{\mathrm{d}t} \\
\therefore m\frac{\mathrm{d}V}{\mathrm{d}t} &= -b\frac{\mathrm{d}\left(x_{out} - x_{in}\right)}{\mathrm{d}t} - k\left(x_{out} - x_{in}\right) \\
\frac{\mathrm{d}V}{\mathrm{d}t} &= -\frac{b}{m}\frac{\mathrm{d}\left(x_{out} - x_{in}\right)}{\mathrm{d}t} - \frac{k}{m}\left(x_{out} - x_{in}\right) \\
V &= -\frac{b}{m}\left(x_{out} - x_{in}\right) - \frac{k}{m}\int\left(x_{out} - x_{in}\right)\,\mathrm{d}t \qquad (8.2) \\
\text{But } V &= \frac{\mathrm{d}x_{out}}{\mathrm{d}t} \\
\therefore \frac{\mathrm{d}x_{out}}{\mathrm{d}t} &= -\frac{b}{m}\left(x_{out} - x_{in}\right) - \frac{k}{m}\int\left(x_{out} - x_{in}\right)\,\mathrm{d}t \\
x_{out} &= -\frac{b}{m}\int\left(x_{out} - x_{in}\right)\,\mathrm{d}t - \frac{k}{m}\iint\left(x_{out} - x_{in}\right)\,\mathrm{d}t\,\mathrm{d}t \qquad (8.3)
\end{aligned}$$

where

F	$=$	Force
b	$=$	Damper's viscosity
x_{out}	$=$	Displacement out
x_{in}	$=$	Displacement in
t	$=$	Time
k	$=$	Spring constant
m	$=$	Mass
a	$=$	Aceleration

To model the output displacement and velocity, Equation 8.2 and Equation 8.3 were implemented. These equations are readily implementable as an analogue computer, as illustrated in Figure 8.12a, by using an adder or an integrator with an inverter for each operation in these equations. However, this solution is nonoptimal and may be optimised through the elimination of duplicate inversions. An optimised computer is illustrated in Figure 8.12b. Note that the number of operational amplifiers has been reduced from nine to seven, which is a 22.2% reduction. This is important because the greater the number of operational amplifiers, the greater the error in the output, the greater the area consumed and the greater the expense involved. In more complex analogue computers, the decrease is often on a larger scale and is consequently of greater importance.

The behavioural level description consists of the derived equations, as shown in Code 8.9. The structural level description instantiates inverters and integrators, connecting them in the configuration shown in Figure 8.12b.

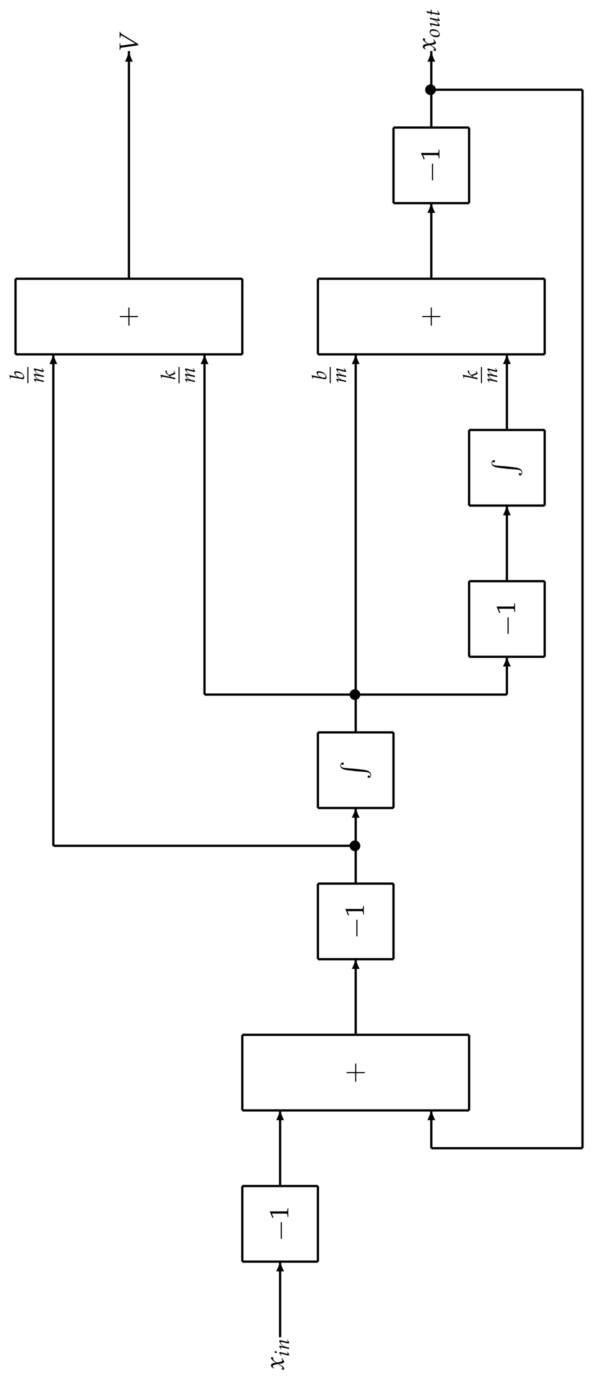
```
voltage_difference == voltage_displacement_out -
    displacement_in'reference;
voltage_velocity == -((viscosity / mass) *
    voltage_difference) - ((spring_constant / mass) *
    voltage_difference'integ);
voltage_displacement_out == voltage_velocity'integ;
```

Code 8.9: Excerpt from `Behavioural/mass_spring_damper.vhd`

The testbench uses a square wave to perturb the mass-spring-damper. It supplies a mass of 0.45 kg, a spring constant of 10 N/m and a viscosity of 0.75 m^2/s as the parameters to the mass-spring-damper. These parameters have no special meaning; they were chosen because they demonstrate the damped oscillatory behaviour of the system illustrated in Figure 8.13. It was important to determine the error between the ideal behavioural model and the real structural model, because if the error were too great, corrective measures would have needed to have been taken before proceeding with the implementation of the reconfigurable physics engine. However, examination showed that the error was very low, well below the limits of human perception and not of consequence if the calculations were used in a computer game.

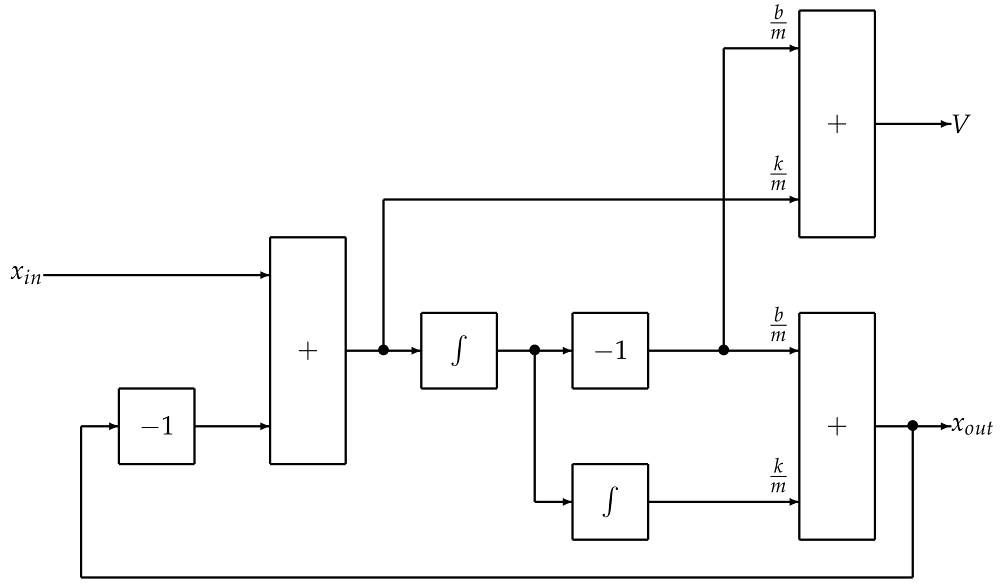
8.7 Simplified Mass-Spring-Damper

This is the same as the mass-spring-damper except the velocity output has been removed. This is of little importance at this stage, but it is required for the



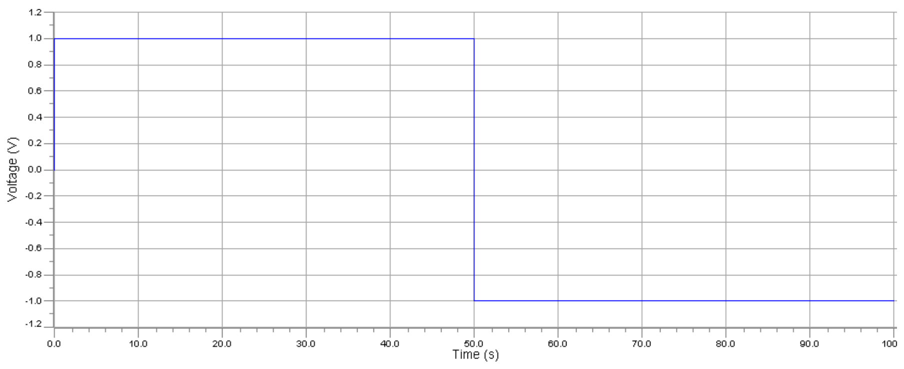
(a) Original

Figure 8.12: Mass-spring-damper analogue computers

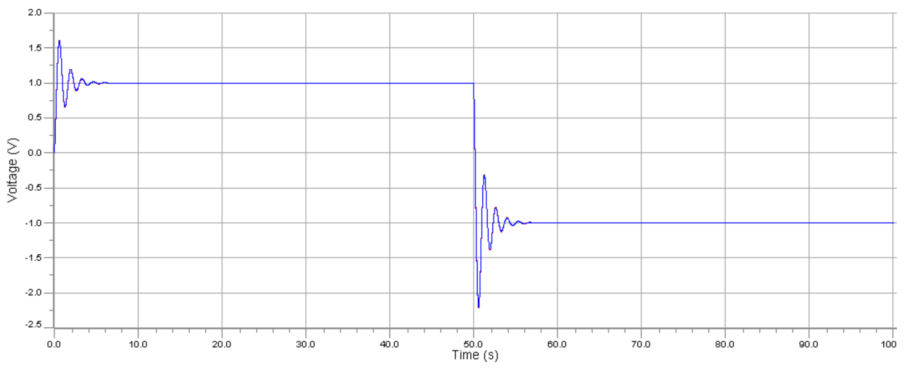


(b) Optimised

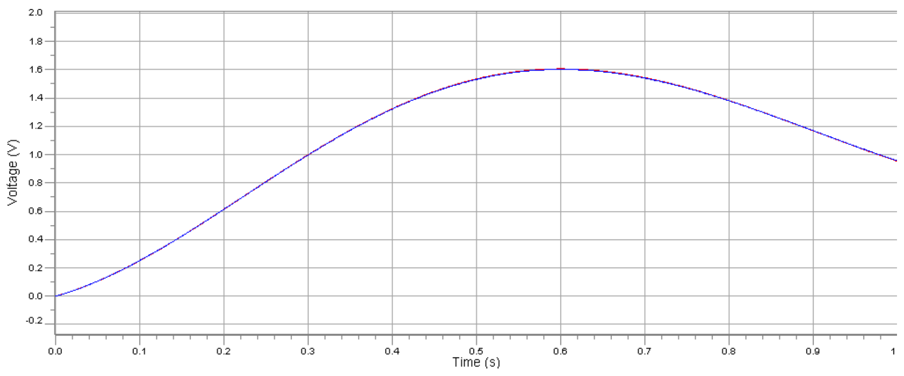
Figure 8.12: Mass-spring-damper analogue computers



(a) Displacement in

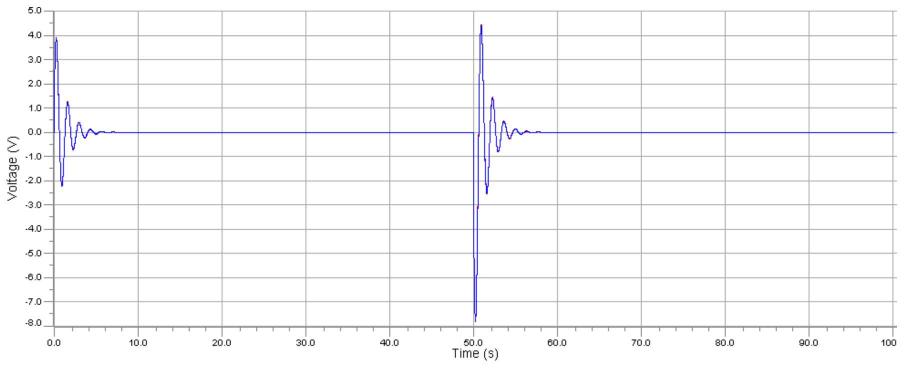


(b) Displacement out

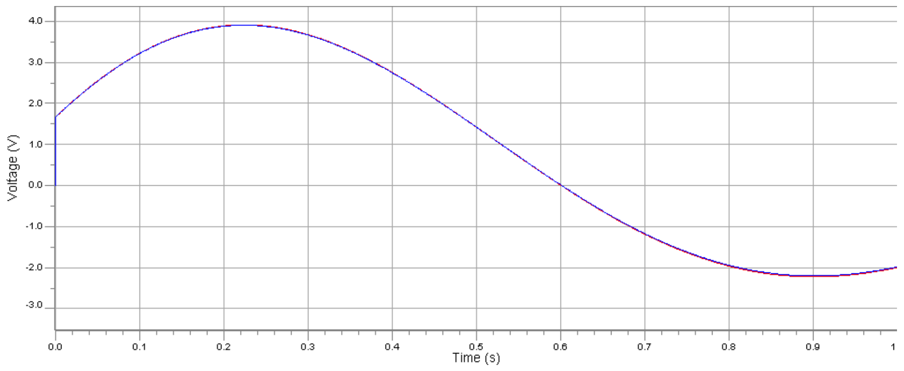


(c) Displacement out magnified

Figure 8.13: Comparison of mass-spring-damper's behavioural and structural models. Red indicates behavioural; blue indicates structural.



(d) Velocity



(e) Velocity magnified

Figure 8.13: Comparison of mass-spring-damper's behavioural and structural models. Red indicates behavioural; blue indicates structural.

construction of the second prototype, discussed in Chapter 9.

The testbench used is identical to that used for the mass-spring-damper. The results obtained are the same and the amount by which the results deviate from ideality is the same.

Chapter 9

Prototype 2: Vehicle Suspension System

This chapter outlines the second prototype constructed for the project. Firstly, information is provided on the physics underlying the system. This is followed with a discussion of the implemented prototype.

9.1 Background

The mass-spring-damper represents a real world system. However, it is unlikely to be of much use in computer games, which are the main target of the project. Consequently, it was decided to construct a more useful prototype to demonstrate that analogue computers are suitable for the modelling of game physics.

Based on these criteria, it was decided to model a vehicle suspension system. One advantage of this system is that it is based on the previously constructed mass-spring-damper, allowing for reuse of existing components.

9.1.1 Physics of Vehicle Suspension System

Many variations of suspension system are used in vehicles today. The vast majority of vehicle suspension systems can be closely modelled using standard equations. However, these equations are complex and this has led to many simplified variations of the equations. This simplification is essential for software physics engines, since modelling the suspension systems of many vehicles simultaneously would require a prohibitively large amount of computational power.

However, these simplified equations are typically unsuitable for analogue computers, as they require functions that are difficult to obtain on such a computer. Moreover, these simplifications do not provide any benefit in terms of computation speed, due to the parallelism inherent in analogue computers. The only benefit obtained would be a reduction in the number of terms and therefore, the quantity of hardware required.

The original, more complex, equations work best on an analogue computer. The model illustrated in Figure 9.1 is referred to as the walking-beam model [35, p. 68]. The model has three degrees of freedom. This model only takes into account a single axle; the two axles are independent. To consider the other axle, the system would need to be duplicated. The system consists of one mass-spring-damper representing the springiness of each tyre. The two mass-spring-dampers' outputs are connected to an axle. Above the axle is another mass-spring-damper representing the suspension of the vehicle. The output of this mass-spring-damper represents the motion of the vehicle's body. The model is very accurate for modelling suspension systems and it is commonly used to model vehicles such as cars. Moreover, it is suitable for modelling offroad driving, which may be useful in some computer games.

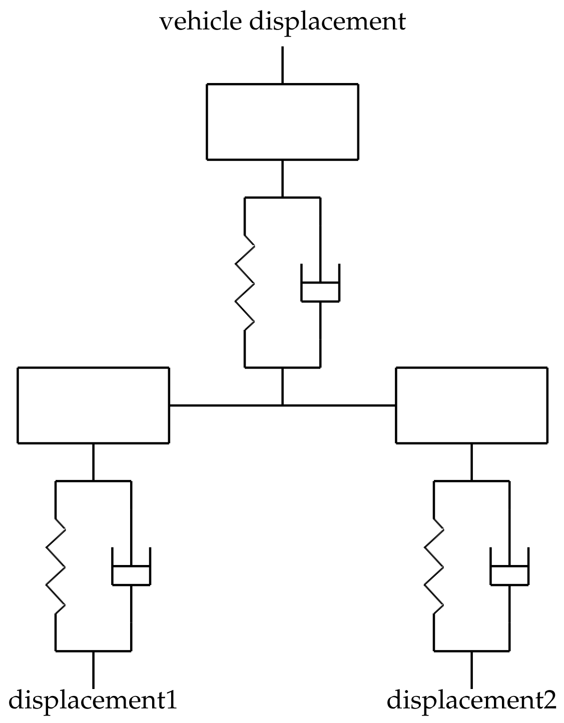


Figure 9.1: Walking-beam model for vehicle suspension system

9.2 Vehicle Suspension System

The project's vehicle suspension system is provided with six parameters, the mass of each wheel, the spring constant of each wheel, the viscosity of each wheel, the mass of the vehicle body, the spring constant of the suspension system's spring and the viscosity of the suspension system's damper. Two input displacements are specified and the output displacement is generated.

The vehicle suspension system was constructed using three of the previously constructed simplified mass-spring-dampers and two extra operational amplifiers. The two extra operational amplifiers emulate the axle by summing half of the displacement resulting from each wheel. No new entities needed to be constructed to implement the vehicle suspension system. This system is illustrated in Figure 9.2. This system only implements one axle of the vehicle suspension system. The second axle could be modelled in the same way, but it would require twice the number of entities. Chapter 11 offers a neater solution to this problem.

The testbench uses two square waves, with one wave applied to each wheel, simulating a bump on the road surface. It supplies a wheel mass of 0.45 kg, a wheel spring constant of 0.75 N/m, a wheel viscosity of 10 m^2/s, a vehicle body mass of 0.45 kg, a suspension system spring constant of 10 N/m and a suspension system viscosity of 0.75 m^2/s as the parameters to the vehicle suspension system. These parameters have no special meaning; they were chosen because they demonstrate the behaviour of the system illustrated in Figure 9.3. In fact, these parameters would not work well for a real vehicle as there is insufficient damping, which would lead to a very rough drive. Naturally, the parameters could easily be modified to provide this extra damping.

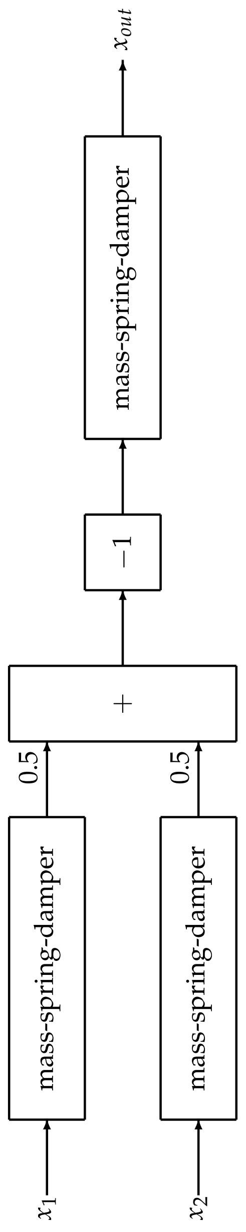
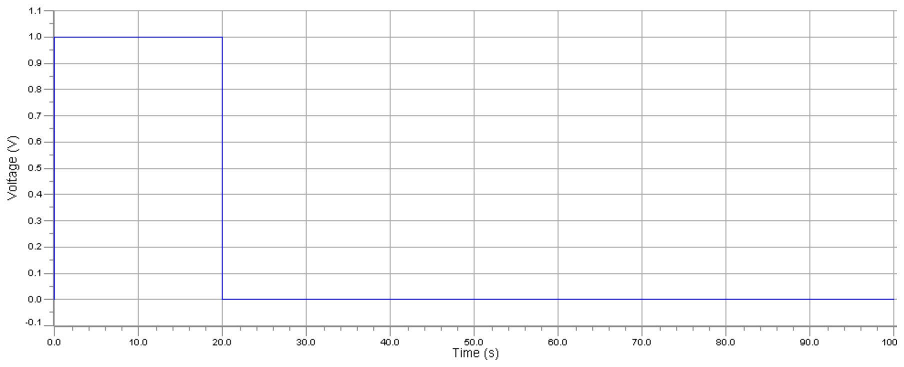
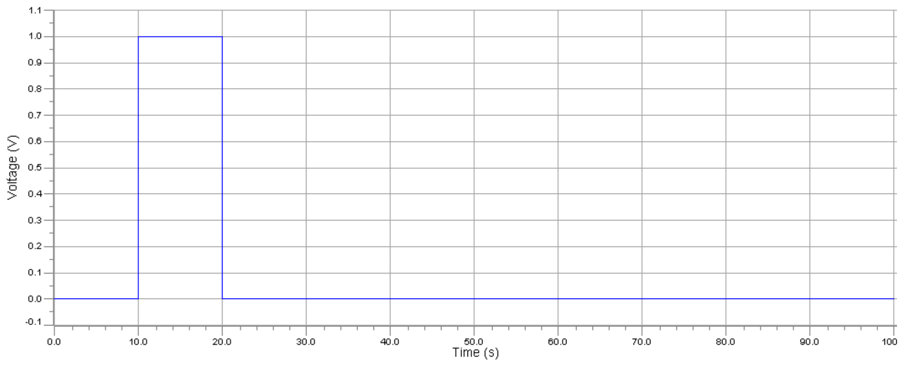


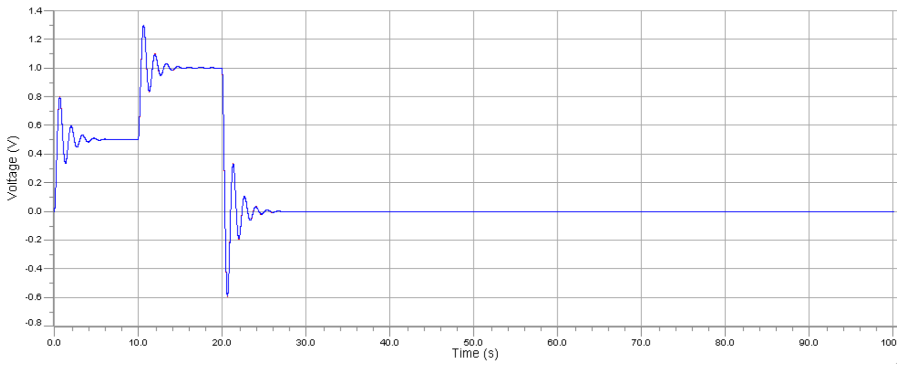
Figure 9.2: Vehicle suspension system analogue computer



(a) Wheel 1 input

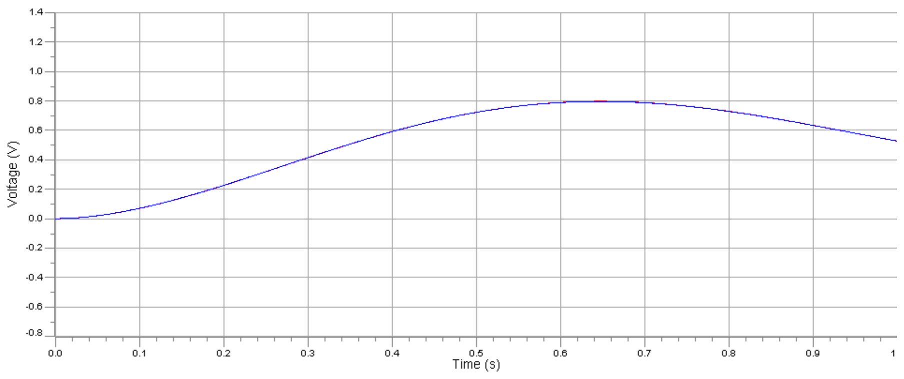


(b) Wheel 2 input



(c) Displacement

Figure 9.3: Comparison of vehicle suspension system's behavioural and structural descriptions. Red indicates behavioural; blue indicates structural.



(d) Displacement magnified

Figure 9.3: Comparison of vehicle suspension system's behavioural and structural descriptions. Red indicates behavioural; blue indicates structural.

Chapter 10

Reconfigurable Hybrid Computer

This chapter describes the reconfigurable hybrid computer, which was constructed after the second prototype. The entities that constitute the reconfigurable hybrid computer are described, before the computer itself is outlined.

10.1 Switch

A switch is used to control which circuit paths are currently active and which are disabled. A switch in an IC would typically be implemented using a single transistor.

The project's switch has a digital control signal. When the signal is high, the switch is closed and current flows. When the signal is low, the switch is open and current does not flow.

The behavioural level description of the switch is idealistic. The entity breaks the circuit when open but effectively disappears when closed. However, this model is unrealistic as no circuit element can effectively disappear. To take into account these nonidealities, the structural level description was created. If the switch is closed, it has a low resistance and current can easily pass through the entity. If the switch is open, it has a high resistance and current cannot easily pass through the entity. This description works on the basis that the switch will be on one possible circuit path. Current will flow down the path of lesser resistance. If there is only one path, the switch will serve no useful purpose.

The testbench supplies a sinusoidal waveform to the switch, while the control signal is tested in both positions. For the behavioural description, when the switch is closed, the output voltage becomes the input voltage, but when the switch is open, the output voltage becomes zero. For the structural description, since there is only one circuit path in the testbench, the current will flow down this path and the output voltage will stay at the level of the input voltage irrespective of the position of the switch.

10.2 Variable Resistor

A variable resistor or potentiometer is a resistor whose resistance may be modified. In this case, the variable resistor is a digitally controllable variable resistor or a digital potentiometer, meaning that its resistance may be specified using a digital control signal. These entities consist of a chain of small resistors, which may be individually switched on and off. Many of these entities provide a serial interface using a technology such as I^2C or SPI. Others only allow the resistance to be raised or lowered by one at any given time.

The resistance of the project's variable resistor may be modified by supplying a control byte specifying the desired resistance, providing simpler control than the commercially available devices. The lack of flexibility in the commercially available devices is likely due to the cost and area consumed by adding extra pins. Since no pins are required for an internal variable resistor, this entity provides a parallel interface, which is simpler to use and requires less supporting hardware. Since a byte is provided for selecting the resistance, up to 256 values may be selected. The granularity provided is 50 kΩ, leading to a range of possible values spanning from 0 Ω to $12,750$ kΩ. This granularity was chosen because a value of 0.5 is required for certain applications, such as the vehicle suspension system. Moreover, it allows for results of reasonable accuracy in many simulations. A multiplicative factor of 127.5 should be sufficiently large for the majority of applications.

Only a behavioural level description was created for this device, since such an entity is readily available. The control byte is multiplied by the appropriate values to determine the resistance that the entity should assume. Ohm's Law is then used with this resistance value to create the variable resistor.

The testbench is similar to the one used for the resistor. The testbench attaches a sinusoidal voltage source to one terminal of the variable resistor, so that the voltage at the other terminal may be observed. A single resistor wired in this fashion should not modify the inputted voltage, even while its resistance is changing. Consequently, the output is the same as the input.

10.3 Switchable Resistor

The project's switchable resistor consists of a resistor bracketed by two switches, as illustrated in Figure 10.1, so that the resistor may be switched in and out of the circuit as necessary.

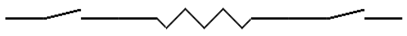


Figure 10.1: Schematic of switchable resistor

The behavioural level description of the device alternates between Ohm's Law and zero volts depending on the state of the switches. The structural level description instantiates a resistor and two switches in the formation illustrated in Figure 10.1.

The testbench is similar to the one used for the resistor. The testbench attaches a sinusoidal voltage source to one terminal of the switchable resistor, so that the voltage at the other terminal may be observed. Meanwhile, the switches are opened and closed. A single resistor wired in this fashion should not modify the inputted voltage. Consequently, the behavioural level description will show the input voltage at the output while the switch is closed, but zero volts while the switch is open. However, at the structural level, the switches should not modify the inputted voltage, for reasons discussed in Section 10.1. Consequently, at this level, the output is the same as the input.

10.4 Switchable Capacitor

The project's switchable capacitor consists of a capacitor bracketed by two switches, as illustrated in Figure 10.2, so that the capacitor may be switched in and out of the circuit as necessary.

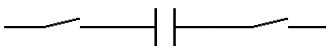


Figure 10.2: Schematic of switchable capacitor

The behavioural level description of the device alternates between the capacitor equation and zero volts depending on the state of the switches. The structural level description instantiates a capacitor and two switches in the formation illustrated in Figure 10.2.

The testbench is similar to the one used for the capacitor. The testbench attaches a sinusoidal voltage source to one terminal of the switchable capacitor, so that the voltage at the other terminal may be observed. Meanwhile, the switches are opened and closed. A single capacitor wired in this way will slightly modify the input voltage while it is charging. After a short period of time, the voltage will become identical to the input voltage. Consequently, the behavioural level description will show a slightly modified input voltage at the output while the switch is closed, but zero volts while the switch is open. However, at the structural level, the switches should not modify the inputted voltage, for reasons discussed in Section 10.1. Consequently, at this level, the output is approximately the same as the input, with a slight deviation while the capacitor is charging.

10.5 Cell

The cell is the basic reconfigurable entity of the project's hybrid computer, used to perform a single operation.

The required functions had to be decided. Inversion is often necessary in analogue computers, since almost all functions invert and this extra inversion would need to be eliminated. Integration is often necessary for differential equations, which describe much of physics. Moreover, integration can be used to eliminate differentiation, which offers poor performance in analogue computers as outlined in Section 6.6.5. Addition is also necessary, but any entity can be turned into an adder by providing it with multiple inputs. Multiplication is also necessary, but again, any entity can be turned into a multiplier by providing it with variable inputs.

Based on this analysis, it was decided to implement inversion and integration as the two functions performed by the cell. Two inputs are provided to allow for addition. If addition of more signals is required, further cells can be used. The resistor strengths on the input lines are variable, to allow for multiplication. It appears that it is possible to construct every simulation required of a physics engine from these functions. This system is based on a reduced instruction set computer (RISC) philosophy. It implements only the required functions; all others may be constructed in terms of these basic functions.

The elimination of differentiation provides an additional advantage. Differentiation is the only function that uses a capacitor at its inputs. The other functions implemented use resistors at their inputs. Therefore, under the current scheme only resistors would need to be provided at the cell inputs, reducing the amount of hardware required.

The functionality of the project's cell is entirely decided by digital control signals. It consists of an operational amplifier and its associated components, and performs one operation at any given time.

The behavioural level description switches between the equations for inversion and integration used in previous entities. The structural level description constructs the circuit illustrated in Figure 10.3. This consists of an operational amplifier with a fixed resistance switchable resistor and a fixed capacitance switchable capacitor in parallel. The switches allow either the resistor or the capacitor to be activated, disabling the other. There are variable resistors on both input wires. By controlling the variable resistors, many different multiplicative factors may be achieved, and each input may be multiplied by a different amount.

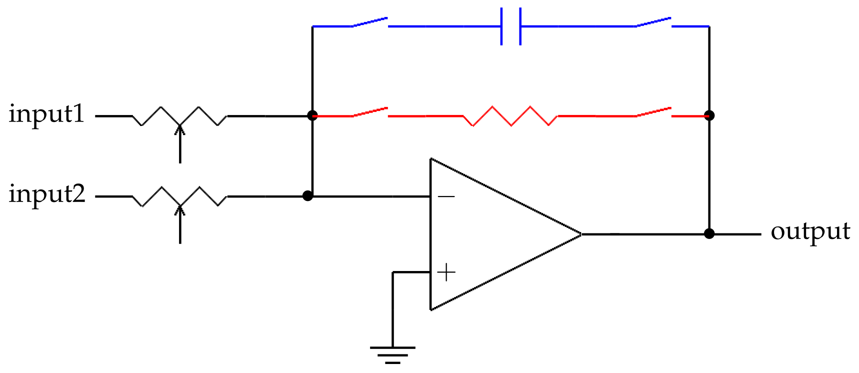


Figure 10.3: Schematic of cell. Red indicates inversion; blue indicates integration.

Instead of supplying variable resistors at the inputs, the values of the capacitor and resistor in parallel with the operational amplifier could have been made controllable. However, this would have resulted in both inputs being multiplied by the same factor, offering less flexibility. Moreover, variable resistors are a standard circuit building block whereas variable capacitors are not. For these reasons, it was decided to use variable resistors at the inputs. This adheres to the RISC philosophy, as there is the potential to make the entire set of entities variable. Such a scheme would lead to a lot of underutilised hardware and questionable gain.

The first testbench supplies a sinusoidal waveform, while changing the operation, so that the output shows both integration and inversion. The first half of the waveform is phase shifted while the second half is inverted. The second testbench supplies a square waveform, requesting the same sequence of events from the cell. The first half of the waveform is triangular while the second half is an inverted square waveform.

10.6 Hybrid Systems

The hybrid systems are hybrid albeit hardwired versions of the simplified mass-spring-damper, mass-spring-damper and vehicle suspension system. These systems are constructing from the cell. These are only reconfigurable when instantiated, by choosing the parameters supplied to them. All other variable elements have been fixed.

These systems were designed to observe the differences between the pure analogue computers and their reconfigurable counterparts. Switches will obviously lead to leakages since an open switch offers a large but not infinitely large resistance. However, this leakage should be small. These systems enable the accurate measurement of this leakage before constructing more complex elements. If the leakage were too great, solutions would need to be found and implemented at this stage.

The behavioural level descriptions use the relevant differential equations, so that their descriptions are the same as their non-hybrid equivalents. The structural level descriptions connected the cells in the required arrangement, mimicking the arrangements used for the prototypes.

The testbenches used are the same as for the prototypes. This allows for the results obtained from the two to be compared. All analyses suggested that these entities worked perfectly. The minor leakage is illustrated in Figure 10.4 and Figure 10.5.

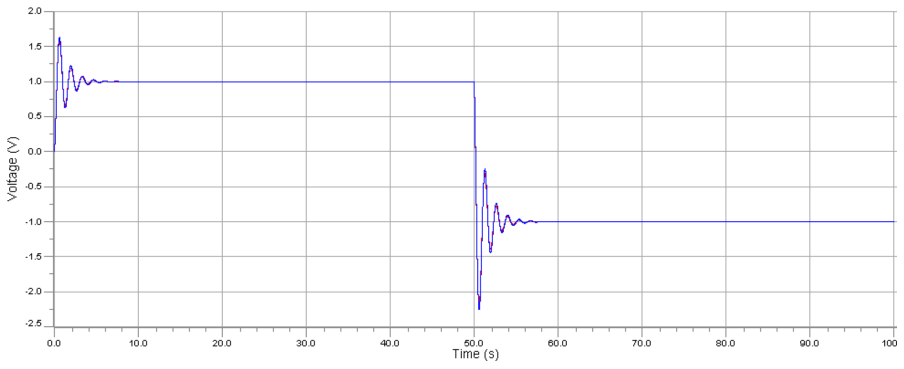
10.7 Router

A routing entity was required to route the appropriate signals to the cell's input. This entity would likely consist of switches, to create and terminate connections as required.

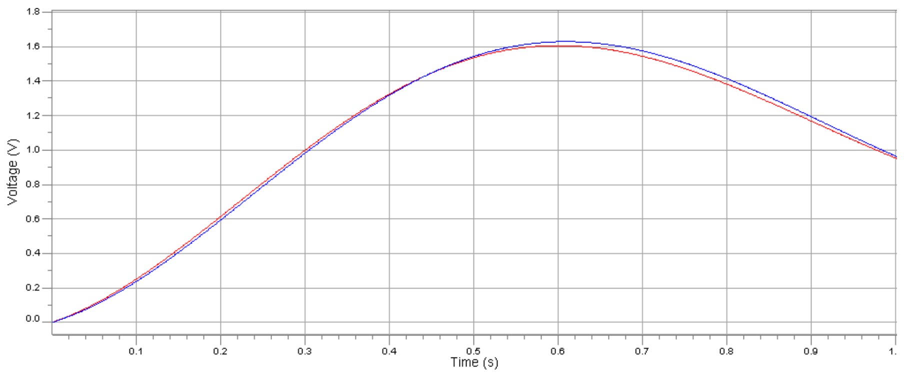
The project's router takes a 5-bit digital select signal, allowing thirty two inputs. The first input is always connected to ground, allowing thirty one input lines to be connected. Ground is necessary for when an input must be disabled, such as when only one of a cell's two inputs is used. Two outputs are provided, corresponding to the cell's two inputs.

Only a behavioural level description of the router was created. This description assigns the voltage of the chosen input signal to the output signal. A structural level description seems an obvious choice for this entity. However, this description required thirty two switches, exceeding the design size limits of the simulator (Section 12.1).

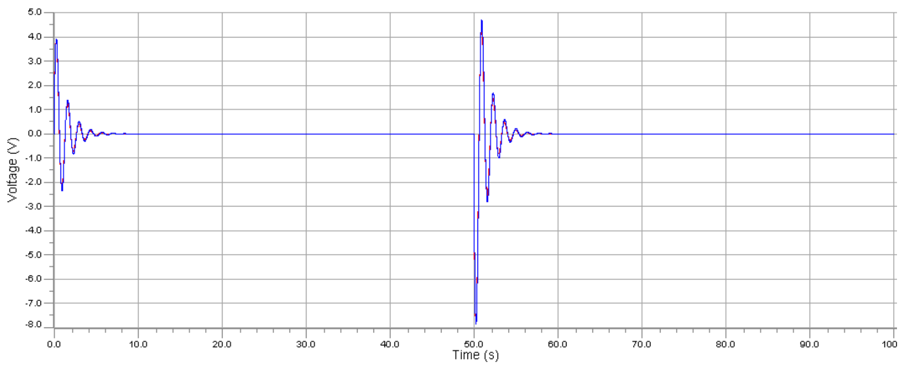
The testbench generates thirty one input signals of differing amplitudes. Meanwhile, the control signal is incremented, so that each signal will be dis-



(a) Displacement out

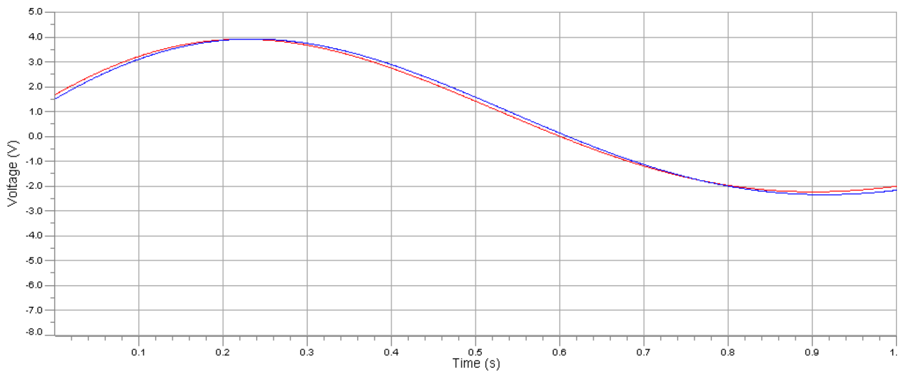


(b) Magnified displacement out



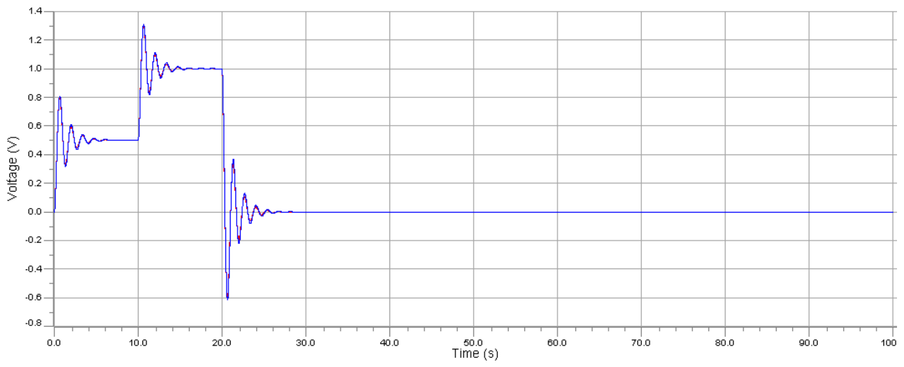
(c) Velocity

Figure 10.4: Comparison of hybrid mass-spring-damper's behavioural and structural descriptions. Red indicates behavioural; blue indicates structural.

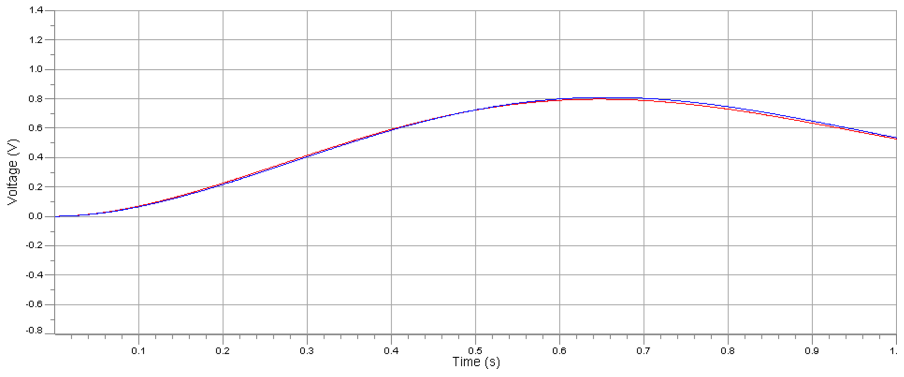


(d) Magnified velocity

Figure 10.4: Comparison of hybrid mass-spring-damper's behavioural and structural descriptions. Red indicates behavioural; blue indicates structural.



(a) Displacement



(b) Magnified displacement

Figure 10.5: Comparison of hybrid vehicle suspension system's behavioural and structural descriptions. Red indicates behavioural; blue indicates structural.

played at the output. The output waveforms are discontinuous sine waves that gradually increase in amplitude.

10.8 Cell Router

The project's cell router consists of a cell and a router. It simply provides a convenient entity with which to construct more complex entities.

The behavioural level description merely merges the behavioural descriptions of the cell and the router. A more natural description cannot be created. The structural level description creates the circuit illustrated in Figure 10.6.

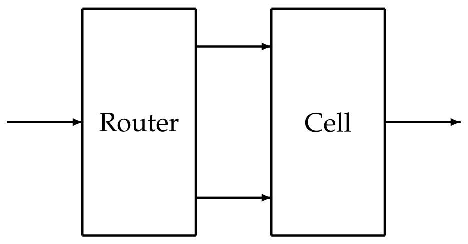


Figure 10.6: Schematic of cell router

The first testbench supplies a sinusoidal waveform, while changing the operation, so that the output shows both integration and inversion. The first half of the waveform is phase shifted while the second half is inverted. The second testbench supplies a square waveform, requesting the same sequence of events from the cell. The first half of the waveform is triangular while the second half is an inverted square waveform.

10.9 Computer

The computer constitutes the bulk of the reconfigurable hybrid computer. It consists of cells connected via programmable interconnect. The inputs and outputs are analogue but the control signals are digital.

The main challenge lay in designing the interconnect. A few ideas were considered, based around the general ideas of a mesh and a bus.

The mesh would consist of cells arranged in a matrix formation. Each cell would get its inputs from the cells to its immediate left. The first cells would also get inputs from the rightmost cells. The mesh's inputs would come in the left and outputs would leave from the right. At first glance, this arrangement appears useful. A great deal of hardware is not required, since the maximum number of inputs to each cell is equal to the number of cells to its immediate left. However, on attempting to map systems to such an architecture, problems are quickly uncovered. Firstly, such an architecture is difficult to program. This is a disadvantage, but not sufficient to make this design worthless. However, the lack of flexible routing in this design prevents the mapping of many standard physics systems, such as the vehicle suspension system. Methods could be designed to overcome these problems, but such methods would likely remove all advantages gained by the use of such a system, while substantially increasing the difficulty involved in programming such a system.

The alternative to the mesh was a bus-based architecture. This concept would use a bus line for each input and a bus line for each cell. Each cell would be connected to every bus line. A cell's output would be connected to a specific bus line for each cell, so that only one cell would drive each bus line. The last bus lines would be used as the outputs. Therefore, if a specific signal needed to be present outside the analogue computer, it could be processed by the last cell, which would output it to an appropriate bus line. Such a solution results in the use of a lot of hardware to route the correct signal to the cell's inputs. However, it appears to be the only solution sufficient to map physics problems, such as the vehicle suspension system, to the hybrid computer. In addition, this solution is substantially easier to program that the aforementioned mesh.

Based on this analysis, the bus solution was chosen. Two inputs were required. The design of the cell router left twenty nine remaining bus lines, so twenty nine cells were created. The last two bus lines were used as the outputs. This is sufficient for relatively large problems, such as the vehicle suspension system.

The behavioural level description simply merges the behavioural descriptions of twenty nine cell routers, connected to thirty one bus lines. A more natural description cannot be created. The structural level description creates thirty one bus lines and instantiates twenty nine cell routers as illustrated in Figure 10.7.

A possible optimisation involves removing the complete flexibility permitted by the current solution, and only allowing a subset of bus lines to be routed to each cell. However, analysing the prototypes constructed suggested that such a solution would be difficult to implement correctly. For example, designing an optimised solution that works with the vehicle suspension system could result in a solution that fails to work with a multitude of other necessary systems. If more problems were to be analysed, patterns could possibly be detected and such an optimisation could then be implemented.

The testbenches test the system by configuring it to simulate the simplified mass-spring-damper, the mass-spring-damper and the vehicle suspension system problems, using the same parameters as before. These testbenches ensure that the same level of functionality may be achieved through the computer, as through the prototypes. Moreover, the prototypes provide a benchmark against which the computer's error may be analysed. The outputs of the mass-spring-damper testbench compared with the mass-spring-damper prototype are shown in Figure 10.8. The outputs of the vehicle suspension system testbench compared with the vehicle suspension system prototype are shown in Figure 10.9. Both highlight that the deviation from ideality is minimal.

10.10 DAC

Digital to analogue converters (DACs) are used to convert digital signals to equivalent analogue signals. They provide a means by which digital electronics can communicate with analogue electronics.

The project's DAC performs its conversion with a granularity of 8 bits.

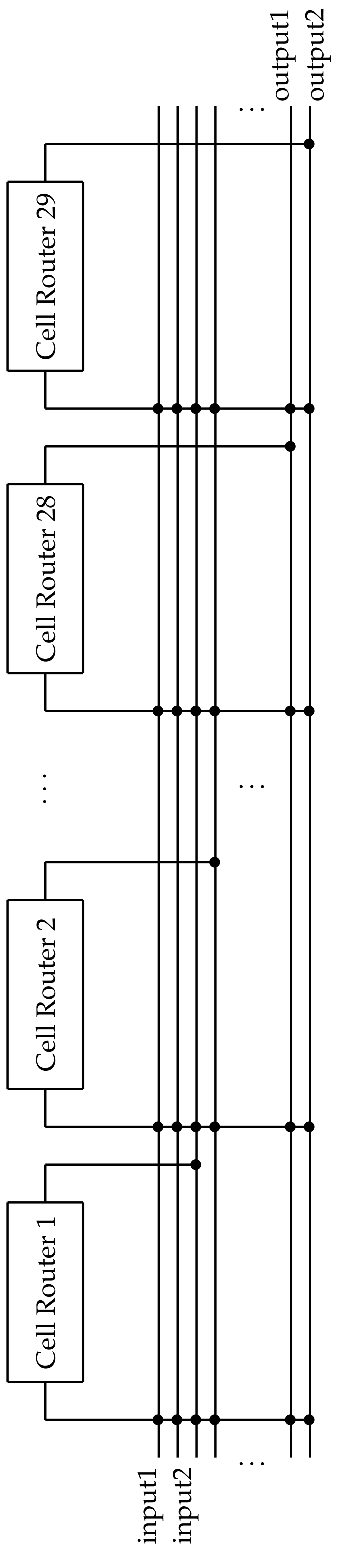
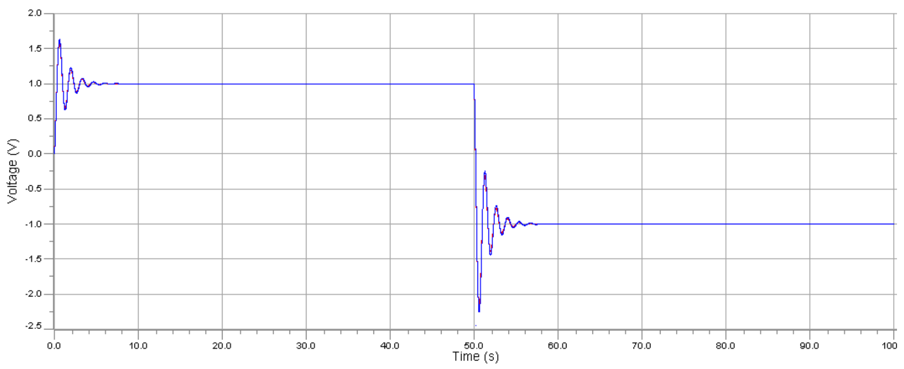
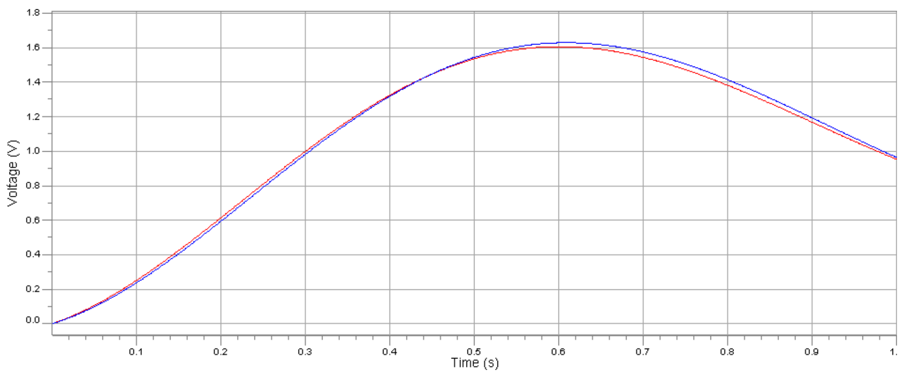


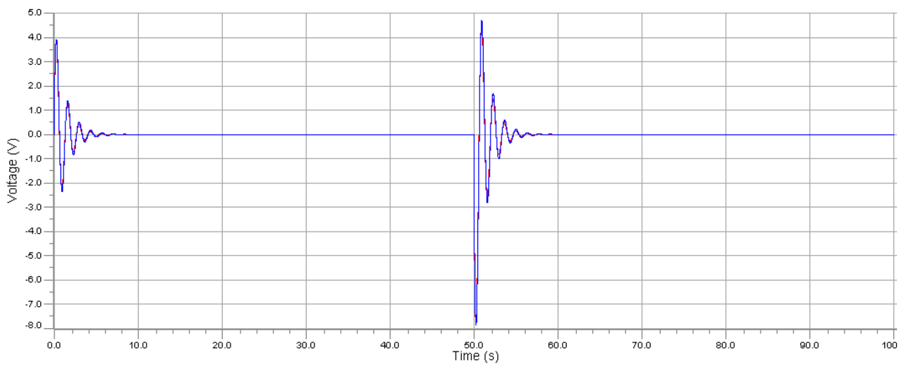
Figure 10.7: Schematic of computer



(a) Displacement out

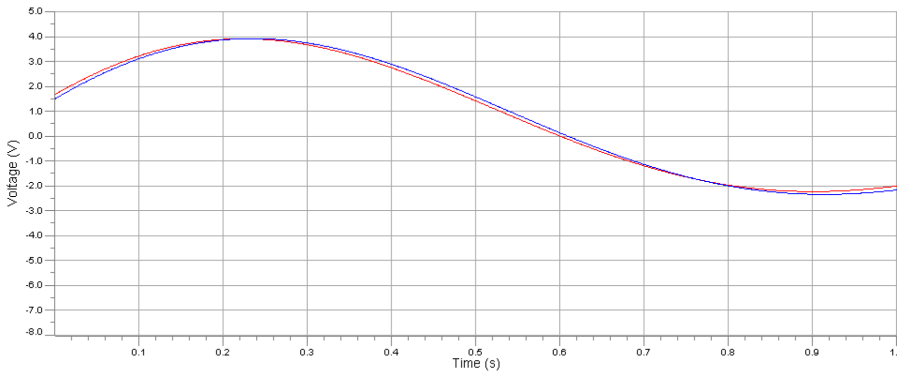


(b) Magnified displacement out



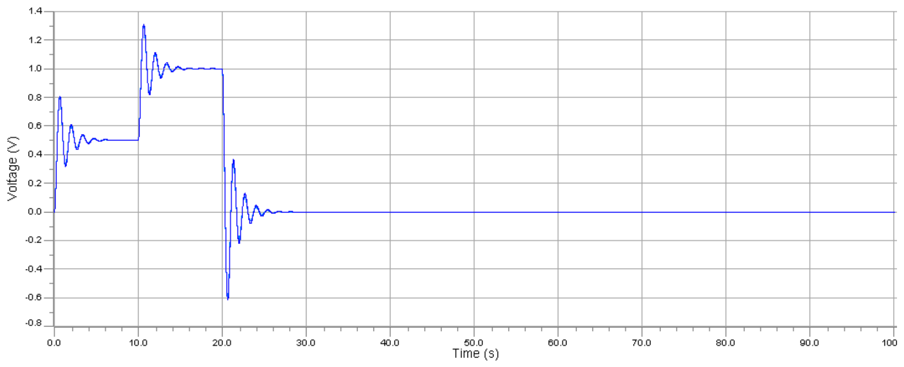
(c) Velocity

Figure 10.8: Comparison of computer's structural mass-spring-damper test-bench against mass-spring-damper's behavioural description. Red indicates mass-spring-damper; blue indicates computer.

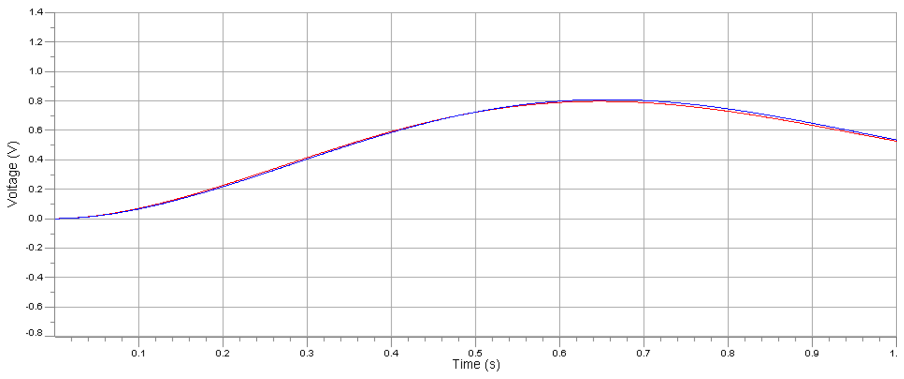


(d) Magnified velocity

Figure 10.8: Comparison of computer's structural mass-spring-damper testbench against mass-spring-damper's behavioural description. Red indicates mass-spring-damper; blue indicates computer.



(a) Displacement



(b) Magnified displacement

Figure 10.9: Comparison of computer's structural vehicle suspension system testbench against vehicle suspension system's behavioural description. Red indicates vehicle suspension system; blue indicates computer.

More sophisticated DACs are widely available but are expensive. As stated in Section 2.4.2, accuracy is not very important for computer games. Objects only need to appear as if they are behaving realistically. Higher accuracy would lead to more expense that would probably not be observed. Each bit inputted to the DAC represents 0.1, instead of the more standard 1, to allow for greater accuracy. This works well because most systems simulated will use numbers close to zero, so achieving accuracy for lower numbers at the expense of higher numbers becoming impossible is satisfactory. Additionally, the DAC has clock and reset inputs. All output voltage changes will occur on the rising edge of the clock, and the active low reset input forces the device to output 0 V. Although these inputs serve little purpose for this entity, they were added to create an inverse of the analogue to digital converter (ADC) (Section 10.11).

Only a behavioural level description of the DAC was constructed because DACs are standard entities. There are many different ways a DAC could be implemented, but these details are unimportant for the project. The description applies type conversion functions to implement the DAC whenever a rising edge of the clock is observed. The core process of the DAC implementation is displayed in Code 10.1.

```vhdl
process(clock, reset)
  variable registered_output_voltage: voltage;
begin
  if reset = '0' then
    registered_output_voltage := 0.0;
  elsif rising_edge(clock) then
    registered_output_voltage := to_real(input) *
        voltage_per_bit;
  else
    registered_output_voltage :=
        registered_output_voltage;
  end if;
  temporary_output_voltage <= registered_output_voltage
      ;
end process;
```

Code 10.1: Excerpt from `Behavioural/dac.vhd`

The testbench supplies the DAC with the binary values representing 10 and -10, which produces a square waveform of amplitude 2 V at the output.

10.11 ADC

ADCs are used to convert analogue signals to equivalent digital signals. They provide a means by which analogue electronics can communicate with digital electronics.

The project's ADC performs its conversion with a granularity of 8 bits. More sophisticated ADCs are widely available but are expensive. As stated in Section 2.4.2, accuracy is not very important for computer games. Each bit outputted from the ADC represents 0.1, instead of the more standard 1, to allow for greater accuracy. This works well because most systems simulated will use numbers close to zero, so achieving accuracy for lower numbers at the expense of higher numbers becoming impossible is satisfactory. Additionally, the ADC has clock and reset inputs. All output voltage changes will occur on the rising edge of the clock, and the active low reset input forces the device to output zero. Although real ADCs do not use these signals, they were added to allow for synchronous behaviour. A synchronous ADC could easily be constructed by placing a register after an ADC.

Only a behavioural level description of the ADC was constructed because ADCs are standard entities. There are many different ways an ADC could be implemented, but these details are unimportant for the project. The description applies type conversion functions to implement the ADC whenever a rising edge of the clock is observed. The core process of the ADC implementation is displayed in Code 10.2.

```
process(clock, reset)
begin
  if reset = '0' then
    output <= "00000000";
  elsif rising_edge(clock) then
    output <= to_std_ulogic_vector(input'reference /
        voltage_per_bit, 8);
  end if;
end process;
```

Code 10.2: Excerpt from `Behavioural/adc.vhd`

The testbench supplies the ADC with a waveform that steps from 0 V to 9 V, so the digital output should rise from zero to ninety in steps of ten.

10.12 Digitised Computer

The digitised computer provides a digital interface to the hybrid computer. Inputs, outputs and control signals are all specified in a digital format. This facilitates the use of this entity in a purely digital system, such as a standard computer.

The behavioural level description combines the description of the computer with the descriptions of two DACs and two ADCs. The structural level description connects these components together as illustrated in Figure 10.10. The code implementing this is shown in Code 10.3.

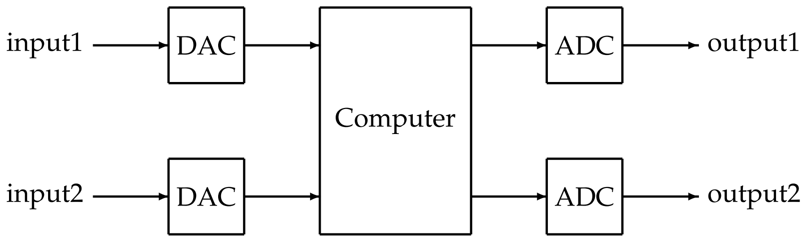


Figure 10.10: Schematic of digitised computer

```vhdl
convert_input1: dac
  port map (
    clock => clock,
    reset => reset,
    input => input1,
    output => analogue_input1
  );
convert_input2: dac
  port map (
    clock => clock,
    reset => reset,
    input => input2,
    output => analogue_input2
  );

compute: computer
  port map (
```

⋮

```vhdl
  );

convert_output1: adc
  port map (
    clock => clock,
    reset => reset,
    input => analogue_output1,
    output => output1
  );
convert_output2: adc
  port map (
    clock => clock,
    reset => reset,
    input => analogue_output2,
    output => output2
  );
```

Code 10.3: Excerpt from `Structural/digitised_computer.vhd`

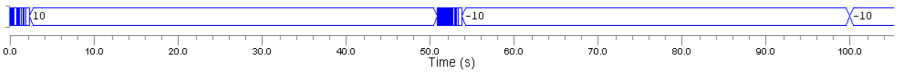
The testbenches test the system by configuring it to simulate the simplified mass-spring-damper, the mass-spring-damper and the vehicle suspension system problems, using the same parameters as before. These testbenches ensure that the same level of functionality may be achieved through the digitised computer, as through the prototypes. Moreover, the prototypes provide a benchmark against which the digitised computer's error may be analysed.

10.13 ADE

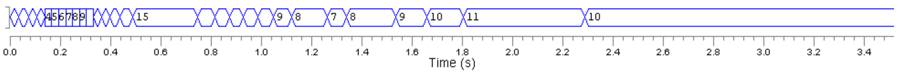
The ADE is the top level entity. Currently, it only consists of the digitised computer and consequently it is unbeneficial to create such an entity. The reasons for creating this entity will be outlined in Chapter 11.

The behavioural level description is the same as that used for the digitised computer. The structural level description instantiates a single instance of the digitised computer.

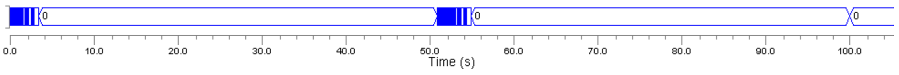
The testbenches test the system by configuring it to simulate the simplified mass-spring-damper, the mass-spring-damper and the vehicle suspension system problems, using the same parameters as before. These testbenches ensure that the same level of functionality may be achieved through the ADE, as through the prototypes. Moreover, the prototypes provide a benchmark against which the ADE's error may be analysed. The outputs of the mass-spring-damper testbench are shown in Figure 10.11. The outputs of the vehicle suspension system testbench are shown in Figure 10.12. In both cases, the oscillatory behaviour may be observed by noting how the numbers change.



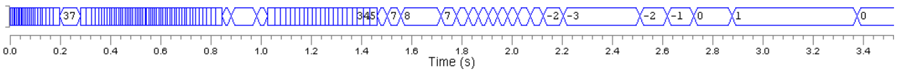
(a) Displacement out



(b) Magnified displacement out

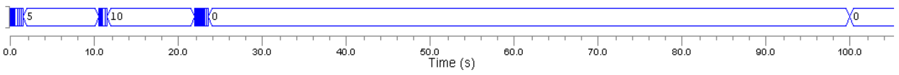


(c) Velocity

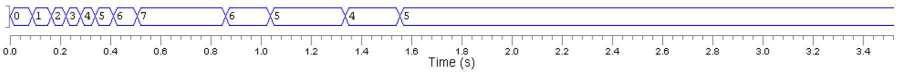


(d) Magnified velocity

Figure 10.11: ADE's structural description, using the mass-spring-damper testbench



(a) Displacement



(b) Magnified displacement

Figure 10.12: ADE's structural description, using the vehicle suspension system testbench

Chapter 11

Multiplexing

This chapter is concerned with discovering a way to run multiple simulations "simultaneously", quickly switching between iterations. The background concept is outlined, before the modifications that were carried out to the reconfigurable hybrid computer are discussed.

11.1 Background

The previously outlined physics engine will perform the vast majority of physics calculations with the desired accuracy. However, if such a system were to be used in practice, one problem remains: the system is capable of running only a single physics simulation at any particular time. For example, with the above system, one vehicle suspension system can be simulated at any particular time. However, it is likely that a racing game would need to simulate many vehicle suspension systems simultaneously.

To facilitate this, it was necessary to decide on a scheme through which simulations could be multiplexed. In other words, a scheme through which many simulations could be run during the timeframe of one.

11.1.1 Key Concept

All simulations previously described have been executed in real-time. However, if many simulations are being run in the place of one, the simulations must obviously be executed faster that real-time. For example, fifty simulations must each be executed in one fiftieth of real-time.

As outlined in Section 3.4.1, one of the most desirable properties of analogue computers is that they operate in real-time. However, analogue computers may also perform time compression or time expansion. To perform time compression, each constant multiplier in the equation should be multiplied by the speed gain. To perform time expansion, each constant multiplier in the equation should be divided by the speed decrease. In the project, the variable resistors can be used to modify the equation's constant multipliers.

Suppose the equation

$$ax^2 + bx + c = 0$$

describes the system being simulated. If it were necessary to run the system at twice the speed of real-time, the equation

$$2ax^2 + 2bx + 2c = 0$$

would be modelled instead. If it were necessary to run the system at half the speed of real-time, the equation

$$\frac{a}{2}x^2 + \frac{b}{2}x + \frac{c}{2} = 0$$

would be modelled instead.

11.1.2 Multiplexing Suggestions

Multiplexing simulations requires more than time compression. Therefore, some scheme for saving and restoring the current state must be designed. This section analyses three possible schemes.

11.1.2.1 Suggestion 1: Iterating Outputs

This kernel of this idea is to store output values so that they may be later reapplied to the inputs.

In this scenario, a register file would be placed in the digital part of the computer. Initial values for each simulation would be read into the registers from outside the physics engine. For the first iteration, the values in the registers would be supplied to the system sequentially. The outputs of this iteration would be saved in the registers after one timestep, overwriting the initial values. To do this, the final feedback loop of each analogue computer would be removed. Instead, the loop would go through an ADC, DAC and register file. After each iteration, the user would read the output values from the register file. Afterwards, the process would continue using the saved values. A simple version of this system is illustrated in Figure 11.1.

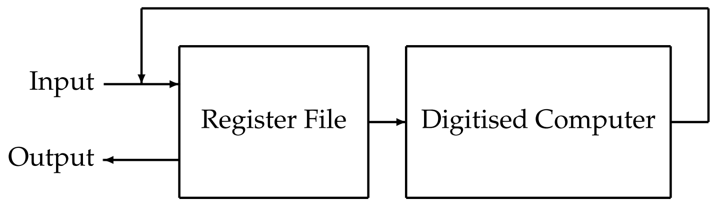


Figure 11.1: Multiplexed computer using suggestion one

This system works on the principle that the vast majority of analogue computing problems, such as the mass-spring-damper system, can be designed so that their input and output are connected. This design was not used for this project, since it would have required two signals driving the same line, which the simulator cannot simulate (Section 12.4). In the scenario outlined here, the output of the register would constitute a single line driver, so this should not be a problem. However, it is not obvious that the vehicle suspension system can be redesigned to have a final loop.

There are larger problems as to why this suggestion will not work. State is not determined by the output value. Supplying the previous output value will work in the same way as supplying the first input value. Since the signal varies with time, the state must depend on time. Only the integrators work with respect to time; all other components work based on the instantaneous input signal. This time-varying behaviour is provided by the charge stored in the capacitors, used by the integrators. Therefore, to save state, the charges in the capacitors need to be stored, instead of the output value.

Consequently, this suggestion is invalid and it could not work.

11.1.2.2 Suggestion 2: Simulating Change

In order to formulate a viable solution, the inputs and outputs of the previously constructed systems were analysed for patterns. Based on this, two key observations were made: firstly, that outputs only change because of a change in inputs; secondly, that outputs will eventually stabilise to a constant value, unless very unrealistic parameters have been chosen. These properties appear to hold true for every possible system and could provide a solution.

To supply the input change, a register file would be placed at the inputs to the system. This register file would contain two registers for each possible simulation. The first register would contain the “previous input” and the second register would contain the “current input”. The first register would be supplied to the system to restore the previous state and afterwards, the second register would be supplied to apply the transition. If the previous state were supplied for a sufficiently long duration, then the capacitors should revert to their previous state. In this system, the inputs would be completely determined by the user; the system does not feed output values back into the system.

Another register file would be placed at the outputs of the system. This register file would contain many registers for each possible simulation. These registers would start recording the value of the output shortly after the current input is applied to the inputs. One register would be used for each timestep. If the length of a timestep was chosen well, the output should be constant by the time the last register is used. Therefore, the software using the physics engine could retain the previous value until another transition occurs.

A simplified version of this system is illustrated in Figure 11.2.

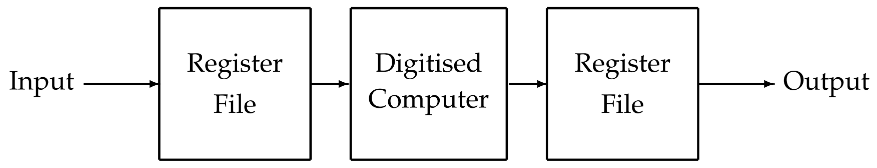


Figure 11.2: Multiplexed computer using suggestion two (simplified)

In summation, there are three distinct phases in which this system operates: firstly, the input value loading phase during which two values are read for each simulation; secondly, the actual simulation; thirdly, the output value reading phase during which the results of the simulation are obtained.

However, such a system would be very slow. The loading and reading phases will each take time away from simulation, particularly if many output registers are used. In addition, part of the time allocated to simulation will be consumed trying to restore the previous state, during which time the system is effectively idle. Moreover, the system would need to run each simulation for a relatively long period of time, after which the probability of not having reached stability is very low. If the software using the physics engine only required the first few readings, the physics engine would still record many results, several of which would be worthless. Therefore, the physics engine would have expended time performing worthless actions. To summarise, simulation results will take considerable time to appear, but they will come in clusters. The biggest problem with this is that results are typically required instantaneously and concurrently. Therefore, this solution is not ideal.

There is another problem associated with this solution. The time required to load the previous state is not easily determined, due to the different charges stored in each capacitor. The only way to fully ensure the system reaches its previous state is to leave it charging for as long as the simulation has run previously. However, this would be an extremely long delay and clearly unsuitable.

A modification of this solution would be to connect a register to each capacitor so that the capacitor's charge is stored locally. This would be difficult to implement and would likely be problematic. More importantly, it does not solve any issues.

Although this solution could work, a better solution would be desirable.

11.1.2.3 Suggestion 3: Capacitor Replication

Capacitors store charge. Instead of using registers to store the value of the capacitors' charge, capacitors can be used directly.

This solution increases the number of capacitors. Through switching, each capacitor is used for only one simulation. If switching is performed slowly, the charges will leak from the capacitors, but if switching is performed quickly, the charges will likely stay in the capacitors while the switches are closed.

This solution rectifies the timing issues associated with the second idea. Moreover, it simplifies both the solution and the programming of the computer. Therefore, this was the suggestion used to implement multiplexing in the project. This solution required modification of the already constructed components. The modifications made are discussed in the subsequent sections.

11.2 Capacitor Stack

The capacitor stack allows for switching between capacitors. A digital control signal operates the switching.

The behavioural level description consists of three instances of the capacitor equation. The structural description consists of three swichable capacitors in parallel, as illustrated in Figure 11.3.

Since only three capacitors have been provided, only three simulations can be multiplexed. It would be desirable to include many more capacitors, but this was impossible due to the design size constraints imposed by the simulator, outlined in Section 12.1. However, using three capacitors and one resistor is

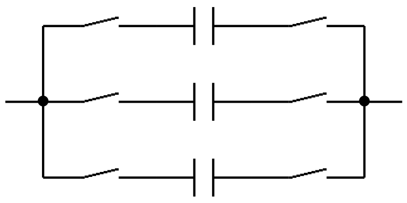


Figure 11.3: Schematic of capacitor stack

somewhat beneficial as the operation select signal can be a decoded 2-bit signal. Higher powers of two also exhibit this beneficial characteristic.

The testbench supplies a sinusoidal waveform, while modifying which capacitor is currently activated. The output is a mostly undistorted version of the input. However, some distortion will be present during the charging phase of each capacitor.

11.3 Cell

The cell was modified to replace the switchable capacitor with the capacitor stack, as illustrated in Figure 11.4. The operation signal was also modified, to accommodate the extra bits required for operation selection.

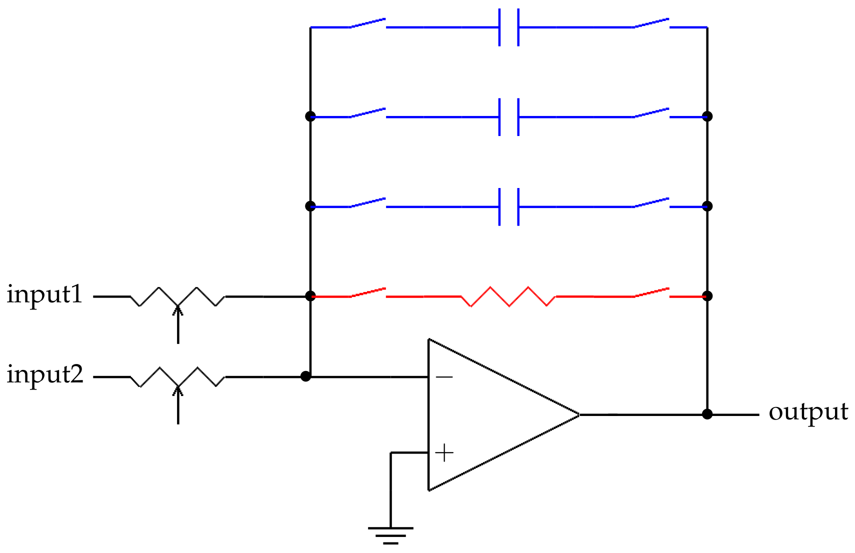


Figure 11.4: Schematic of multiplexed cell. Red indicates inversion; blue indicates integration.

The testbenches are similar to the ones used previously. However, instead of testing only inversion and integration, all three integrations are tested. For the sinusoidal waveform, the first three quarters of the waveform are phase shifted while the last quarter is inverted. For the square waveform, the first three quarters of the waveform are triangular while the last quarter is an inverted square waveform. In both cases, when switching between the different capacitors used for integration, there will be slight glitches.

11.4 Cell Router

The cell router was only modified to accommodate the larger operation signal, which passes through this entity.

The testbenches are similar to the ones used previously. However, instead of testing only inversion and integration, all three integrations are tested. The output appears virtually the same as that produced by the cell.

11.5 Computer and Digitised Computer

The computer and digitised computer were only modified to accommodate the larger operation signal, which passes through these entities.

The existing testbenches were also modified to accommodate the new size of the operation signal. In addition, new testbenches were constructed for the computer and digitised computer. These are based on the previously constructed simplified mass-spring-damper, mass-spring-damper and vehicle suspension system testbenches, but use multiplexing to perform three simulations in the timeframe of one. These are replicas of the testbenches created for the new ADE entity, so a more complete description is deferred until Section 11.9. In addition, another testbench was created for the purposes of evaluating the speed of the entity. This is described in greater detail in Section 15.4.3.

11.6 Operation Decoder

Currently, the operation signal is 4 bits and only one bit can be active at any given time. All of this information can be encoded in 2 bits. Therefore, it is better to provide a 2-bit interface to the user, and to provide an operation decoder that converts this to its native 4-bit format.

The project's operation decoder converts a 2-bit signal to a 4-bit signal. This is done synchronously with the rising edge of the clock. A reset signal converts the output to all zeroes.

Only a behavioural level description is provided for this entity. A structural level description is superfluous since the behavioural level may be synthesised. Nothing is gained by providing a list of the gates used to construct a decoder. This description assigns the decoded input to the output, synchronous to the clock.

The testbench supplies all possible input values to the decoder, so that the output may be observed.

11.7 Operation Decoders

Twenty nine operation decoders are required, since each cell's inputs must be decoded. To group these, this entity was created.

The behavioural description combines the descriptions of twenty nine decoders, for there is no simpler behavioural description. The structural description instantiates twenty nine operation decoders.

The testbench supplies all possible input values to each decoder, so that the output may be observed.

11.8 Control Unit

A control unit is used to send control signals to the datapath of a design. In this case, the datapath is the digitised computer. Currently, the control unit is nothing more than a wrapper for the operation decoders. It has been designed so that other control logic could be instantiated inside this unit, if it was required.

The behavioural description combines the description of twenty nine decoders, making it the same as the operation decoders entity. The structural description instantiates the operation decoders entity.

The testbench supplies all possible input values to each decoder, so that the output may be observed.

11.9 ADE

The top level component was modified to accommodate the new size of the operation signals. It also instantiates the control unit so that the operation signal inputs are correctly decoded. The new version of this component is illustrated in Figure 11.5.

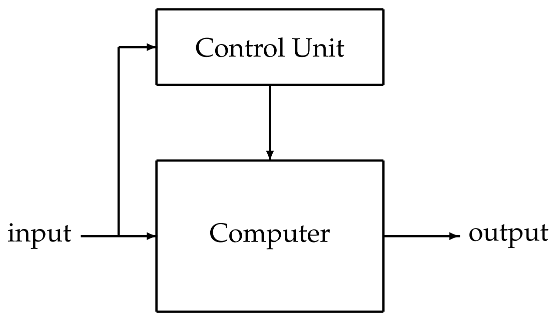


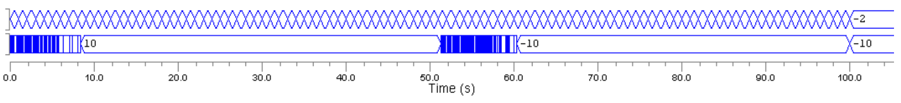
Figure 11.5: Schematic of multiplexed ADE

The previous testbenches have been modified to accommodate the new size of the operation signal. Since multiplexing is not used in these testbenches, the integration operation has been hardwired to `01`.

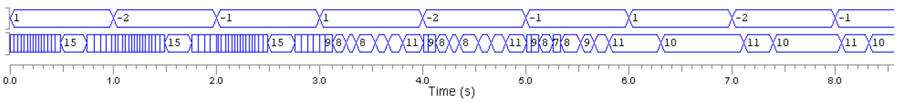
In addition, new testbenches have been created to test the multiplexed functionality of the physics engine. The capacitor switching algorithm shown in Code 11.1, changes the currently active capacitor every second. This switching could be performed more quickly if necessary, but this substantially increases the time the simulator takes to perform the simulation (Section 12.2). These testbenches also multiply the multipliers of their equations by three, since three simulations will be run concurrently and each must run at one third of real-time. The outputs are displayed in Figure 11.6 and Figure 11.7. As it is difficult to determine what is occurring during these simulations, Figure 11.8 and Figure 11.9 show the same simulation executed on the computer component. Examining these reveals that the same values are repeated at the outputs three times, corresponding to each of the three simulations. The outputs are not exactly the same but vary slightly in timing. This could be due to leakage of the switches or the finite precision provided by the simulator. However, it is of little consequence since such differences will be imperceptible when used in a computer game, as outlined in Section 2.4.2.

```vhdl
multiplexer: process
begin
  loop
    integrate <= "01";
    wait for 1 sec;
    integrate <= "10";
    wait for 1 sec;
    integrate <= "11";
    wait for 1 sec;
  end loop;
  wait;
end process;
```

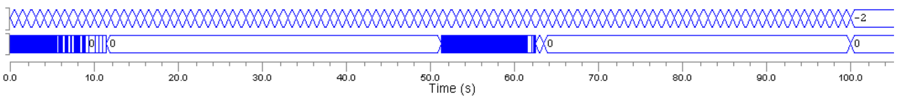
Code 11.1: Excerpt from `Structural/ade_testbench_mass_spring_damper_multiplexed.vhd`



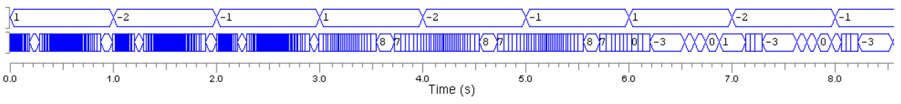
(a) Displacement out



(b) Magnified displacement out

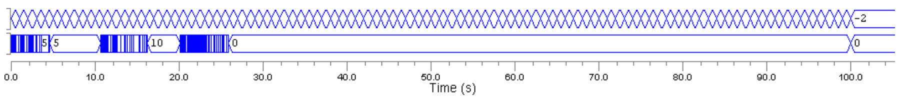


(c) Velocity

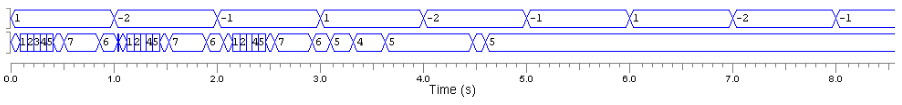


(d) Magnified velocity

Figure 11.6: ADE's structural description, using the mass-spring-damper testbench. The first waveform indicates the current simulation number.



(a) Displacement



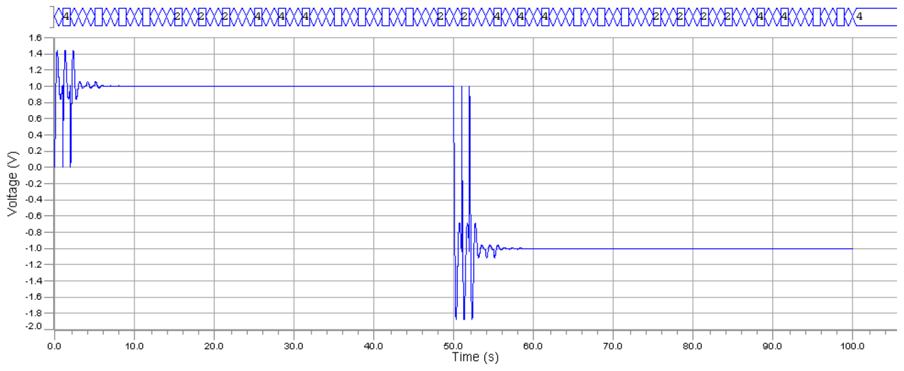
(b) Magnified displacement

Figure 11.7: ADE's structural description, using the vehicle suspension system testbench. The first waveform indicates the current simulation number.

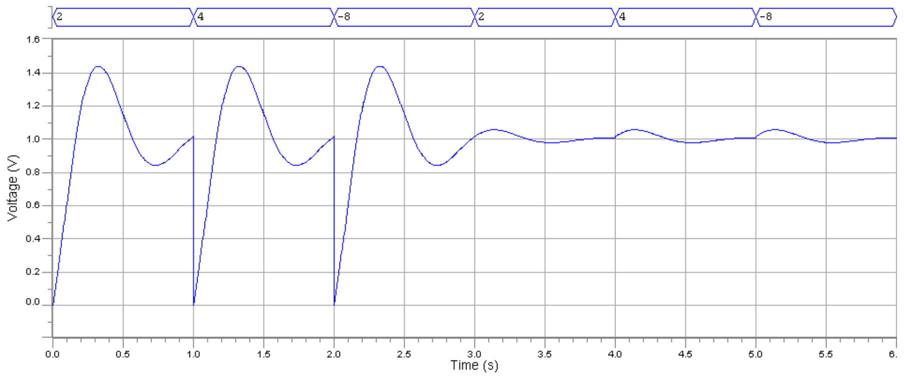
Finally, another testbench was created for the purposes of evaluating the speed of the entity. This is described in greater detail in Section 15.4.3.

11.10 Hierarchy

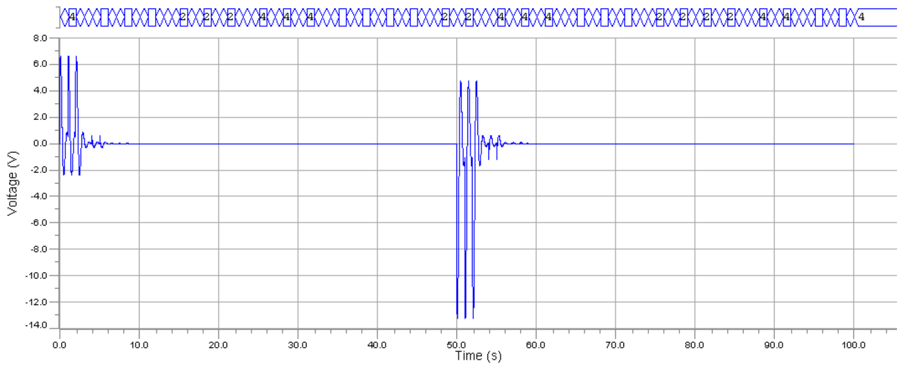
The final hierarchy of VHDL-AMS entities is displayed in Figure 11.10. Note that this hierarchy only specifies the types of entity and not the quantity instantiated.



(a) Displacement out

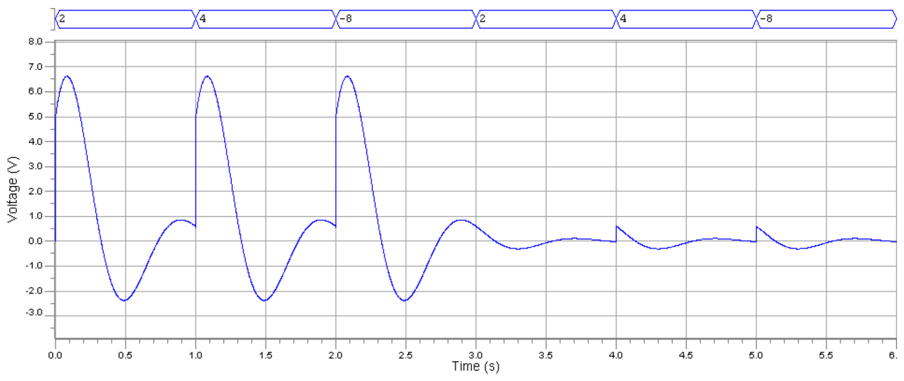


(b) Magnified displacement out



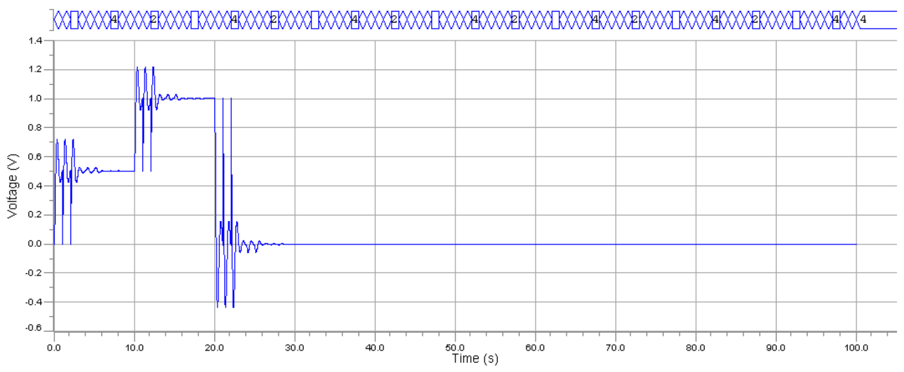
(c) Velocity

Figure 11.8: Multiplexed computer's structural description, using the mass-spring-damper testbench. The first waveform indicates the current simulation number.

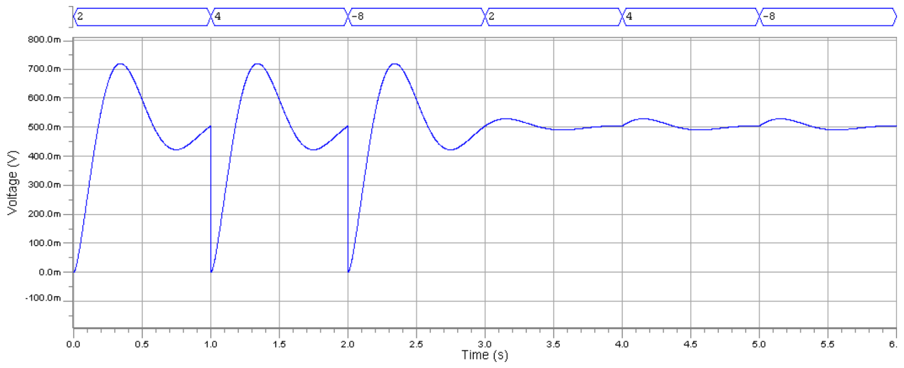


(d) Magnified velocity

Figure 11.8: Multiplexed computer's structural description, using the mass-spring-damper testbench. The first waveform indicates the current simulation number.



(a) Displacement



(b) Magnified displacement

Figure 11.9: Multiplexed computer's structural description, using the vehicle suspension system testbench. The first waveform indicates the current simulation number.

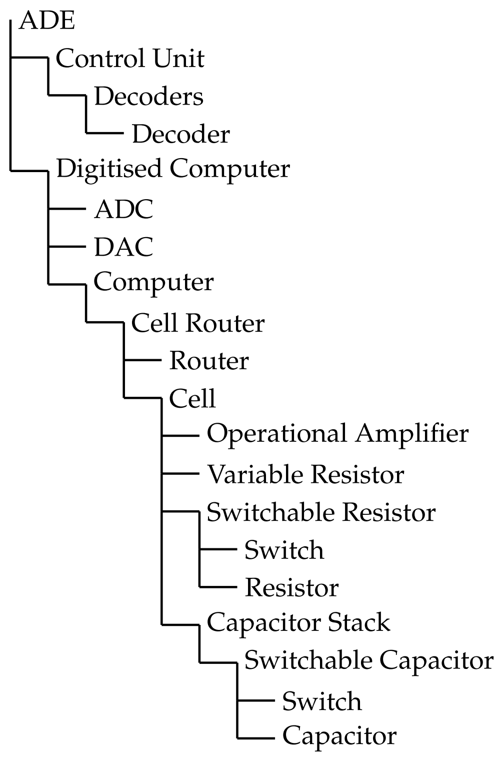


Figure 11.10: Hierarchy of VHDL-AMS entities

Chapter 12

Problems and Solutions

A number of problems were encountered during the implementation of the project, resulting from the simulator used. All VHDL-AMS compilers and simulators currently available are still under development, hence these problems. These are outlined in the following sections.

12.1 Size

The software used placed limits on the size of designs. This raised a number of issues throughout the project.

If the design were too large, the simulator would enter an infinite loop. During this infinite loop, 100% of the system's CPU was utilised by the simulator and no feedback was provided as to what was happening. To ensure that the software was not just sluggish, the simulator was left running for twenty four hours. After this time had elapsed, no feedback or results were available. Therefore, it was clear that the simulator could not simulate the system.

However, the manufacturer did not supply guidelines as to the limit on design sizes. It was often unclear when the simulator was entering the infinite loop as opposed to when it was only sluggish.

The problem had to be manually diagnosed and corrected. Correction was limited to reducing the complexity of the circuit either by using alternative designs or by removing entities. Obviously, these corrective methods were nonideal.

12.2 Speed

The dichotomy of analogue and digital means digital can never truly simulate analogue. The finite granularity of digital means that continuous analogue can only be approximated. This dichotomy resulted in simulations taking excessive amounts of time. Indeed, large designs could take longer than four hours to simulate.

In addition, the simulator provided options to control the accuracy of the approximation. The default accuracy of the software was sufficient for many simulations, but if precise timing was required, the accuracy had to be increased. This resulted in the simulator working even slower.

This was a great hindrance during the project, because more time was consumed simulating than coding. During simulation, the VHDL-AMS code could not be modified. This prevented the rapid testing of alternative ideas. Certainly, had the simulator been faster, more coding would have been achieved in the same timeframe.

One possible solution was to reduce the complexity or number of entities in the design, but clearly, this was not a good solution. Consequently, the only real solution was to wait.

12.3 VHDL-AMS Standard

As outlined in Section 7.2.1, SystemVision 2002 did not implement the entire VHDL-AMS specification.

In particular, some library functions were not implemented. Although this did not prevent the construction of any designed entities, it meant that workarounds had to be found in many cases. For example, the input/output (I/O) libraries were not provided. The voltage source (Section 8.2.1) used for this project read its values from an array. It would have been more natural for the source data to have been read from a text file, but this was impossible due to the lack of the I/O libraries. Other unimplemented library functions include conversion functions and functions for determining if there are unknown bits inside bit vectors. Where necessary, these functions were reimplemented inside custom-built packages.

In addition, not all VHDL-AMS constructs were supported. This meant certain that entities have been provided with somewhat unintuitive descriptions. For example, since aliasing was not implemented, `electrical_ref` had to be used in place of `ground`. Another example is that if a voltage or current was specified in one branch of a statement, it had to be specified in all branches. In certain cases, it was challenging to implement a description conforming to this criterion, while retaining the functionality of the design. Ultimately, a solution was found for every entity.

A final, minor issue is that the IEEE analogue libraries are in `ieee_proposed`, instead of the correct `ieee`, as the software was created before the entire VHDL-AMS specification had been ratified. The correct library names have been provided, but commented out, at the top of each file.

12.4 Looped Back Inputs

It is typical to construct analogue computers whose outputs are looped back to provide the inputs. The output is looped back and connected to the input. Therefore, the input is specified as the combination of the current state and the external signal. This provides a very natural way to design analogue computers.

However, these designs could not be used for the project, as the simulator cannot simulate multiple line drivers. To resolve this problem, modified versions of the systems were used so that the input and the output were regarded separately. This ultimately had no impact on the outcome of the project, but it removed the most obvious design path for much of the project.

Even if such designs are superior, the ideas cannot be applied to all situations. For example, it appears that the vehicle suspension system cannot be designed in this way, as it has two inputs but only one output.

12.5 Resistor and Capacitor Strengths

As outlined in Section 8.5.2, capacitors and resistors in the integrator must be balanced so that a multiplicative factor of one can be readily obtained from the entity. Although this does not sound difficult, the range of capacitor values that may be used is very limited. The range of resistor values is less limited. In essence, should the value of the capacitors have been too high, the output waveform would be completely distorted. Low capacitances worked well, but if the values were too low, the waveform would also be somewhat distorted. Moreover, it was unsatisfactory to fix the capacitors to some small capacitance and leave the resistors at 1 Ω because this meant a multiplicative factor of one could never be obtained from the circuit. Consequently, a high resistance had to be chosen to counteract the effects of the low capacitance. After repeated experimentation, it appeared that the values specified in the `constants` library, 100 kΩ resistance and 10 μF capacitance, provided the least distorted waveform possible.

However, the solution involved much trial and error as the correct values are dependent on the type of operational amplifier used, and cannot be determined by an equation. Moreover, due to the slow speed of the simulator, this experimentation proved extremely time consuming and challenging.

Part III

Conclusions

Chapter 13

Synthesis

One of the goals of the project was to find suitable hardware on which the physics engine could be implemented. This chapter analyses the viability of discrete components, application specific integrated circuits (ASICs), field programmable analogue arrays (FPAAs) and field programmable mixed arrays (FPMAs). A number of commercially available FPAAs are analysed and compared. Finally, a decision as to the most suitable hardware is made.

13.1 Discrete Components

The physics engine could be constructed as a set of discrete components interconnected on a printed circuit board (PCB). This solution would use readily available, off-the-shelf components, reducing initial assembly expenditure.

However, discrete component solutions provide numerous manufacturing challenges [36, p. 693]. The total cost of such a solution is likely to be greater than an equivalent ASIC due to the extra time required to assemble the system, coupled with the cost of each component. Moreover, the number of solder joints and connectors is increased, which results in decreased reliability, as there are more connections and components that may fail. This additionally increases the time and cost required for visual inspection. Finally, it subjects the devices to increased noise due to a longer and more exposed signal path. Based on these difficulties, discrete component solutions are rarely used.

However, the physics engine could be prototyped using discrete components connected on a breadboard. This would allow for rapid prototyping, prior to designing an ASIC solution.

13.2 ASICs

The project described in the previous chapters was designed so that it could be synthesised to an ASIC. Compared to other solutions, an ASIC offers a greater degree of flexibility in what can be created. Moreover, since the IC has only a single purpose, there are no unused components, which would be present in a reconfigurable device. This provides more die area for useful entities.

As outlined in Section 7.1, VHDL-AMS will be synthesisable to an ASIC in the future. When this is possible, the basic components used in the design such as the resistor, capacitor and operational amplifier will need to be substituted with the appropriate device from the manufacturer's device library. This should result in only minimal changes in the output generated by the design.

In addition, the design could be synthesised to hardware today. This would involve the reconstruction of the circuit in an application that could produce appropriate hardware.

13.3 FPAAs

FPAAs are the analogue equivalent of field programmable gate arrays (FPGAs). They typically contain a number of operational amplifiers and passive components joined with programmable interconnect. Like an FPGA, a specific function may then be downloaded from a host PC to the FPAA. Typically, they may be reprogrammed potentially infinite times. However, some are write-once like an electronically programmable read only memory (EPROM). These have an advantage since the switches of the reprogrammable devices offer a resistance, thereby modifying the functionality of the device. The write-once fuses offer much less resistance.

FPAAs provide a natural match for the project outlined above since they consist of operational amplifiers and their functionality is instantly reprogrammable. Instead of the design outlined in previous chapters, a number of analogue computers could be designed. Each analogue computer would perform a specific task. For example, one analogue computer would implement the mass-spring-damper system while another would implement the vehicle suspension system. Then, the game programmer would select the required function and the host computer would download the appropriate design to the FPAA. The programmer would supply the inputs and read the outputs from the design. Of course, this does not offer the complete programmability offered by the design discussed above, but it is likely that this complete programmability is not required and would not be fully exploited. Moreover, this solution simplifies programming since the programmer no longer needs to understand the internal architecture of the physics engine hardware. In addition, software physics engines only offer functionality equivalent to the FPAA solution outlined here.

13.3.1 Zetex Semiconductors TRAC

The Totally Reconfigurable Analog Circuit (TRAC) [37], manufactured by Zetex Semiconductors, was the first FPAA commercially available. The most recent version, the TRAC020LHQ36, consists of twenty cells each containing a single operational amplifier. Therefore, the device is capable of performing twenty concurrent operations. The device can perform inversion, addition, logarithm calculation, antilogarithm calculation and rectification. By wiring additional passive components to the terminals of the device, it can be made to perform amplification, attenuation, differentiation and integration. However, the device is no longer available for purchase.

13.3.1.1 Analysis

The TRAC has enough operational amplifiers to simulate the vehicle suspension system. In addition, it performs all of the requisite functions: addition, inversion and integration.

However, integration may only be performed using external components. This places a limit on the number of concurrent integrations and therefore, on the flexibility of the device. Nevertheless, these limits are probably sufficiently high for this project.

The device is programmed on power-up. Therefore, dynamically reprogramming the device would require repeated stopping and starting. This would prove challenging to implement effectively and would result in long delays during which the device would be performing no useful task.

Coupled with these potential disadvantages, there is one serious problem: the device has very recently been discontinued. This clearly indicates that the device could not be used to implement the physics engine.

13.3.2 Lattice Semiconductor ispPAC

Lattice Semiconductor manufactures the In-System Programmable Analogue Circuits (ispPAC) family [38]. These devices are in-system programmable but not dynamically reprogrammable. Internal details are vague, but six programmable cells appear to be the maximum number available. Each cell contains three operational amplifiers, but only one is wired in a standard analogue computer configuration. This is because the devices are designed for signal conditioning and filtering applications. However, some of the devices have inbuilt DACs and up to three inputs. All of the devices have joint test action group (JTAG) or IEEE 1149.1 functionality to simplify programming.

13.3.2.1 Analysis

These FPAAs are the only ones to include DACs and JTAG, which are useful but not entirely necessary.

The biggest disadvantage associated with these FPAAs is the configuration of the operational amplifiers. While suitable for signal conditioning and filtering operations, it is likely that many operational amplifiers would not be utilised in the physics engine, leading to unnecessary hardware with reduced functionality. Certainly, six operational amplifiers are insufficient for even the mass-spring-damper system. Consequently, it appears that these FPAAs are not suitable for the physics engine.

13.3.3 Anadigm FPAAs

Motorola established Anadigm [39] in January 2000. It produces a range of FPAAs with differing numbers of inputs, outputs and "configurable analogue blocks (CABs)".

The most sophisticated device consists of two dedicated output ports and four ports that may be switched between input and output at will. In addition, it consists of four CABs each containing two operational amplifiers, providing eight operational amplifiers in total. Some configurations will allow for more than eight concurrent operations, but certain configurations will only allow for less, due to the internal layout of the device.

The device can perform a wide range of functions and the user may define additional functions. Moreover, integration and differentiation may be performed without the use of external components. It also includes an ADC, which simplifies integration with digital components.

Finally, these devices are the first FPAAs to be completely dynamically reconfigurable. This reconfiguration may be performed either from the host computer using an API or from an on-board microcontroller.

13.3.3.1 Analysis

The dynamic reprogrammability of the device is the most desirable feature for the project, making it the most suitable of the currently available FPAAs. Both methods of reprogramming the device, from the host computer or from an on-board microprocessor, would be suitable for implementing this project.

Furthermore, it includes an ADC, which makes an external ADC unnecessary. A DAC would still be required. It includes sufficient inputs and outputs for the vehicle suspension system. It also performs a satisfactorily wide variety of functions with which to implement all of the necessary physics systems.

The glaring problem is that the device is quite limited in the number of concurrent operations it can perform. This is sufficient for simple problems like the mass-spring-damper, but not for more complex problems like the vehicle suspension system, which requires twenty concurrent operations.

Nevertheless, the dynamic reprogrammability of the device offers a solution to this problem. The system being simulated could be divided into a number of constituent parts with each part being simulated sequentially, possibly using input values from previous simulations. For example, the vehicle suspension system could be first modelled as two mass-spring-damper systems. The two outputs could then be supplied to a third mass-spring-damper system with an additional halving operation, to obtain the final output. The obvious disadvantage with this solution is that it takes longer to perform the required operation, since in the example, three simulations were required instead of one. Therefore, some of the advantages associated with the inherent concurrency of analogue computers have been lost. The simulation could then be accelerated using the approach taken by the multiplexed physics engine, described in Chapter 11. However, this acceleration leads to loss of accuracy and the reconfiguration between simulations would still lead to some time being unutilised.

Another solution to this problem would be to use multiple FPAAs but this increases the cost of the solution and the complexity of programming the devices.

In summation, these devices would be suitable for implementing the physics engine. However, there are some disadvantages associated with the lack of operational amplifiers. This problem will be solved as technology progresses and more operational amplifiers may be placed on a single device.

13.3.4 Other FPAAs

In addition, a number of other FPAAs have been available in the past. For example, Motorola were one of the first companies involved in the field and successfully designed and manufactured FPAAs for a number of years. These products were discontinued when the company established Anadigm. Since Anadigm's product range is an evolution of Motorola's, these FPAAs offered no additional functionality and warranted no further consideration.

13.3.5 Disadvantages

Of course, any FPAA solution has one major disadvantage: only the analogue part of the design may be implemented on the device.

To rectify this problem, discrete DACs could be placed at the inputs to the FPAA and discrete ADCs could be placed at the outputs. The digital part of the design could then be implemented in software on the host computer. This means that part of the design is no longer hardware-based but software-based, potentially reducing its overall speed. Since only a small part of the design would need to be placed in software, this is likely to have only minimal impact.

Another solution would be to use a combination of an FPGA and an FPAA, with the digital part implemented on the FPGA and the analogue part implemented on the FPAA. Again, discrete DACs and ADCs would be needed. However, this solution is likely to be of considerable expense, due to the amount of hardware required.

The ideal solution would be to place both the analogue and digital parts of the design on the same device, with the DACs and ADCs integrated into the device. This is the approach taken by FPMAs.

13.4 FPMAs

FPMAs are an amalgamation of FPAAs and FPGAs, consisting of programmable analogue elements and programmable digital elements with the DACs and ADCs necessary to unite the two divisions.

However, FPMAs are currently not commercially available. They only exist as research prototypes. Additionally, it is questionable as to whether such devices will ever be commercially viable, as many potential manufacturers cite a lack of consumer interest.

13.5 Decision

Ultimately, it was decided not to create a hardware implementation of the physics engine. The only viable hardware for the physics engine was an FPAA, but all of the commercially available FPAAs had some problems associated with them. The Anadigm FPAAs appeared to be the most appropriate. Moreover, FPMAs were more suitable, but unavailable. Besides, any solution involving FPAAs would have required redesigning the analogue computer for the differing architecture, using different software. Consequently, it was decided to concentrate on furthering the VHDL-AMS solution, rather than create a similar implementation for an alternative architecture.

Chapter 14

PC Interfaces

Ultimately, it would be necessary to connect the physics engine to a conventional PC. This chapter analyses some potential solutions.

14.1 Peripheral Buses

ICs similar to the physics engine have traditionally been placed on cards, which interface with the CPU through a peripheral bus.

Many buses have existed since the introduction of PCs, but the two viable options are currently Peripheral Component Interface (PCI) and PCI Express (PCIe). The other connection, Accelerated Graphics Port (AGP), is used only by graphics cards and consequently, is not considered here.

14.1.1 PCI

PCI is the current standard peripheral bus in desktop PCs, having superseded the slower Industry Standard Architecture (ISA). It allows for data transfer at a rate of 133 Mbps. PCI evolved into PCI-X, which used a faster clock to achieve a data transfer rate of $2,133$ Mbps.

PCI offers one possible solution for allowing the CPU and physics engine to communicate. Its ubiquity makes PCI solutions both low cost and commercially viable. There are, however, faster peripheral buses currently available.

14.1.2 PCIe

PCIe is the next generation of the PCI bus. Although backwards compatible with PCI, it uses a redesigned architecture to achieve a data transfer rate of 2.5 Gbps. It is expected that faster versions will be available in the future.

Although the majority of PCs do not yet support PCIe, it is available in many new PCs, having gained widespread support from companies such as Intel. GPU manufacturers have also supported the technology and are gradually switching to this technology from AGP, due to its increased bandwidth and speed.

The increased bandwidth and speed is likely to be of benefit to the physics engine, particularly if it needs to operate at 72 fps (Section 15.4.3). Therefore, this would be the most desirable way to interconnect the CPU and physics engine. However, since it is currently widely undeployed, it makes little sense to use PCIe only solutions. Consequently, this solution coupled with the PCI solution (Section 14.1.1) appears to be the best option.

14.1.3 Motherboard

The device could also be integrated onto the motherboard. In order to reduce the manufacturing cost of PCs, many devices formerly provided only as expansion cards are now integrated onto the motherboard. Examples include GPUs, audio ICs and network ICs. This allows for sharing of certain resources, such as main memory, which removes the need to place memory on each card.

Lower end desktop PCs usually use an integrated GPU, but higher end desktop PCs usually use a graphics card. The graphics card offers higher performance since it is not sharing main memory with the CPU. The same arguments would apply to a physics engine. Since the physics engine would be primarily aimed at high end users, it makes little sense to manufacture a motherboard integrated physics engine for desktop PCs.

Laptops also use motherboard integrated GPUs in order to reduce space. They typically do not feature high end graphics cards as they are rarely used for game playing. Accordingly, it makes little sense to make a motherboard integrated physics engine for laptops either.

14.2 Graphics Cards

High end graphics cards are almost exclusively purchased for games. Since games are also the target application of this physics engine, it may be beneficial to place the physics engine IC on high end graphics cards. Therefore, a computer game player would only need to purchase one expansion card, reducing the cost involved. Moreover, most purchasers would utilise the physics functionality, so that it would not be wasted expenditure.

The two major graphics companies, ATI and NVIDIA, sell their ICs to card manufacturers, who then design a card onto which the IC may be placed. Therefore, it would practical for a company to design a card that uses one of these GPUs and the physics engine.

14.3 External Device

The physics engine could also be manufactured as an external device, which would be connected to the PC using Universal Serial Bus (USB) or IEEE 1394 (FireWire). This is an unusual solution and few devices interface a PC in this way, other than for peripherals such as keyboards and mice. However, USB “soundcards” may be purchased for PCs with no available expansion slots and no built-in “soundcard”. Such a solution is convenient for the user since they do not have to open the case of the PC to install the device. Moreover, it may be the only solution for making the device available to laptops. Such a device should work with heterogeneous hardware, including the PC, Macintosh and Sun workstations, due to the universal nature of USB and IEEE 1394.

However, such devices are typically unsuccessful since they consume desk space. The advantages gained by an external device offer little more than niches for the physics engine. Therefore, such a solution is not entirely appropriate.

14.4 Conclusions

The optimal solution is to place the physics engine IC along with a GPU on graphics cards (Section 14.2). This allows high end users to purchase a card that performs two functions they are likely to use, saving PC expansion slots and expense.

The other viable solution is a custom built PCI (Section 14.1.1) or PCIe (Section 14.1.2) card. PCIe is the preferable solution due to its higher bandwidth, but a PCI solution would still be necessary, as PCIe is currently rare.

Chapter 15

Analysis

As outlined in Chapter 1, the primary purpose for implementing this project was to determine whether it is viable to construct such a system for use within a computer. The following sections analyse some of the potential difficulties associated with this goal.

15.1 Interface

The top-level ADE component has an interface that consists of a large number of bits. As the calculations in Table 15.1 indicate, it utilises 846 bits or 105.75 bytes of data. The majority of ICs do not have 846 pins. Therefore, a way of reducing this number is desirable.

Quantity	Vector Width	Bits	Bytes
Input			
2	1	2	0.25
29	2	58	7.25
58	5	290	36.25
58	8	464	58
2	8	16	2
	Total	830	103.75
Output			
2	8	16	2
	Total	16	2
Input & Output			
	Total	846	105.75

Table 15.1: Pins used by the interface physics engine

The most obvious solution is to input the bits serially and convert them to parallel on the IC, using a standard serial-to-parallel converter.

Alternatively, the description of a number of physics systems could be stored on the device. Afterwards, the programmer would only select the required system, instead of specifying a layout. However, the flexibility of the device would be reduced. This is unlikely to be problematic since it is the approach taken by software physics engines. Besides, the majority of programmers typically use only a subset of the systems provided. Additionally, this solution has the advantage of simplifying the programming of the device, which could potentially make it more viable. There is a great deal of interest in reducing development cycles, which would be facilitated by this solution. Further, many hardware devices have a finite number of configurations and do not allow for complete flexibility. Complete flexibility is of questionable benefit because it exposes the internal architecture, making its later redevelopment difficult due to the need to maintain backwards compatibility.

Therefore, hardwiring a limited number of systems into the device is probably the best solution. It simplifies programming, allows later redevelopment of the device and the reduced flexibility would likely have minimal impact on developers.

15.2 Die Area

The project's physics engine consumes a large quantity of hardware resources. In order to determine if the current design fits on a single ASIC, an estimate of the die area is calculated below.

The design consists of twenty nine cells, each using one operational amplifier.

Each cell uses a fixed resistor of 100 kΩ and three fixed capacitors of 10 μF. These components are switched using eight switches. The two inputs are multiplied by variable resistors whose values may vary from 50 kΩ to $12,750$ kΩ, with a granularity of 50 kΩ. This results in 256 possible resistor values for each input. Therefore, such a resistor consists of 256 resistor slices interconnected with 256 switches. In other words, the system consists of 87 capacitors of 10 μF and 7,453 resistors of 100 kΩ.

External to the cell are two routers, which allow any of thirty one possible inputs or ground to be routed to each output. Each of these would be constructed from thirty two transistor switches.

At the perimeters of the digitised computer are two 8-bit DACs and two 8-bit ADCs.

The control unit of the ADE consists of twenty nine 2-to-4 decoders, featuring registered outputs.

The resistors and capacitors are constructed using special fabrication techniques. Recent IC fabrication data was used to calculate the area of these components [40, pp. 142–144]. The typical resistance that can be achieved from CMOS is $300 - 1000$ Ω per square. For the calculations, it is assumed that the maximum resistance, 1000 Ω, is the default and that a square is 0.35 μm^2. The typical capacitance that can be achieved from CMOS is $1 - 4$ fF/μm^2. For the calculations, it is assumed that the maximum capacitance, 4 fF/μm^2, is the default. In addition, the transistor switches would be fabricated using standard techniques.

The resistor and capacitor values used were chosen for convenience and not for minimum area. Using these values inside a real IC would waste a large quantity of area. To use the minimal area, the area consumed by all of the resistors and the area consumed by all of the capacitors should be equal. This can be achieved by setting the resistors to 31,959,361.354853709459277903125423 Ω and setting the capacitors to 0.000000031289736640752012689654937536124 F. These values are obviously unsuitable for fabrication but serve to illustrate the likely area of the physics engine. The calculated area uses these values.

The operational amplifiers, ADCs and DACs would typically be purchased as intellectual property (IP) blocks. Therefore, to calculate the area of these components, some recently IP blocks, designed by austriamicrosystems, were used [41, 42, 43].

The digital control unit would be constructed from standard logic games. A 2-to-4 decoder consists of two NOT gates and four NAND gates, or six NAND gates. A NAND IP block was used to determine the area [44]. Registers could be constructed from standard logic gates, but they are typically constructed using some optimisations. Consequently, an IP block for a D-type flip-flop was used to determine the size of the registers [45].

In addition, routing typically adds 10% to the size of the IC.

These data are displayed in Table 15.2.

Component	Quantity	Die Area per Component (μm^2)	Total Die Area (μm^2)
Fabricated			
Resistors	7,453	91,312.4610	680,551,771.9364
Capacitor	87	7,822,434.1602	680,551,771.9364
Switch	15,080	0.35	5,278
		Subtotal	1,361,108,821.8727
IP Blocks			
Operational Amplifier	29	12	348
ADC	2	17	34
DAC	2	39.9	79.8
		Subtotal	461.8
Digital Logic			
NAND	174	55	9,570
D Flip-Flops	116	310	35,960
		Subtotal	45,530
		Subtotal	1,361,154,813.6727
Routing			
		Subtotal	136,115,481.3673
		Total	1,497,270,295.04

Table 15.2: Die area consumed by the physics engine

This design is too large to be fabricated. However, there are solutions whereby the die area may be reduced.

The resistors and capacitors consume the greatest quantity of area. The Zetex TRAC FPAA (Section 13.3.1) combats this problem by including few internal resistors and capacitors, requiring external ones to be connected to the pins of the IC as necessary. This is not a perfect solution for the physics engine, as it greatly restricts the reconfigurability of the device.

If the capacitors, resistors and variable resistors were placed off the IC, more pins would be required. Eight pins would be needed for the devices parallel to each cell. In addition, four pins would be required for the variable resistors at each cell. This would require an extra 124 pins. This figure is well within the limits of current design techniques, so this solution will work, although it would be somewhat ungainly.

Other FPAAs eliminate resistors, replacing them with switched capacitors. This is the most viable solution for the physics engine. A switched capacitor consists of a capacitor that may be switched on and off by a clock [24, pp. 492–495]. By controlling the clock, the effective resistance of the circuit may be varied. This switching results in a discrete time analogue signal as opposed to a continuous analogue signal, but this is unimportant since the digital conversion process ultimately results in a discrete time signal. This would reduce the area consumed since the resistor and capacitor values no longer need to be balanced. Instead, low capacitance capacitors, typically in the picofarad range, would be used. High resistance switched capacitors may then be constructed from these components. Another advantage obtained is that the variable resistors do not need resistor slices but a single capacitor with a variable clock, whose period determines the resistance. A final advantage is that the circuit is less likely to be affected by process variations, since the resistor and capacitor do not need to be balanced. It is easy to fabricate many capacitors with the same capacitance. For this reason, switched capacitors have almost entirely replaced resistors in analogue ICs.

To convert the physics engine to a switched capacitor based solution would require the resistor parallel to each operational amplifier to be substituted with a capacitor. The switches already surrounding it may be used to perform the switching. The variable resistor would be substituted with a capacitor and two switches. It is assumed that capacitors of 4 fF would be sufficient. The size of the clock generation logic cannot be calculated precisely, so a value of $10,000\ \mu\text{m}^2$ was chosen, as it is assumed that this logic would be similar in size to the decoder logic. Finally, the registers used by the decoder logic may be removed as these are not entirely necessary.

The new die area calculations are displayed in Table 15.3.

The size of this design is within the limits of current fabrication technology.

This project implementation did not use switched capacitors, since the main goal of the project was to design and implement a physics engine. Therefore, the design of the system had precedence over the fabrication of the system. Moreover, the simulator would almost certainly be unable to simulate a system using switched capacitors due to the design size limits outlined in Section 12.1.

Therefore, although the physics engine could not be currently fabricated as an IC, it could be easily modified to make it suitable for fabrication. Moreover, the current design of the system is suitable for construction using discrete components.

Component	Quantity	Die Area per Component (μm^2)	Total Die Area (μm^2)
Fabricated			
Capacitor	174	1	174
Switch	348	0.35	121.8
		Subtotal	295.8
IP Blocks			
Operational Amplifier	29	12	348
ADC	2	17	34
DAC	2	39.9	79.8
		Subtotal	461.8
Digital Logic			
NAND	174	55	9,570
Clock Logic	?	?	10,000
		Subtotal	19,570
		Subtotal	20,327.6
Routing			
		Subtotal	2,032.76
		Total	22,360.36

Table 15.3: Hypothetical die area consumed by a switched capacitor implementation of the physics engine

15.3 Design

One of the disadvantages associated with analogue design is that it is very labour intensive, in comparison to digital design.

The current analogue circuit sizing process is labour intensive. Experienced designers regularly take weeks to size complex cells. In addition, the layout process is similarly labour intensive. In contrast, the digital circuit sizing and layout processes have been highly automated with only manual fine-tuning required today. This leads to reduced time-to-market, making digital designs more competitive than analogue designs.

However, due to the current resurgence of analogue circuits for use in applications such as wireless communications (Chapter 5), there has been renewed interest in automating these techniques for analogue designs. Moreover, the rapid increase in computing power predicted by Moore's Law makes automation continually more viable, as the once extremely long execution times have now been substantially reduced. In addition, distributed computing techniques have been employed so that analogue back-end engineering runs at comparable speeds to its digital equivalent. Consequently, analogue designs now have virtually the same time-to-market as digital designs. The technology is likely to improve as time progresses.

15.4 Timing

This section analyses how fast the physics engine could be multiplexed. The relevant issues, ADC conversion rate and operation amplifier bandwidth, are first considered. Then, a sample problem is designed and analysed. The results are compared against a software implementation of the problem.

15.4.1 ADC Conversion Rate

Physics engines provide the data enabling the graphics engine to render the necessary objects. Since the graphics engine must execute at a certain minimum speed, this determines how quickly the physics engine must generate data in order to be viable.

The maximum number of frames per second that can be detected by the human eye is approximately seventy two [46, pp. 82–83]. In other words, the human eye detects changes only every $0.013\dot{8}$ s. The rate of change of the outputs in the physics engine is determined by the conversion rate of the ADCs used for digitising the computer's analogue outputs. The converters currently in place may not be used for an analysis, as they are ideal ADCs. Current practical ADCs range from a few thousand to a few million samples per second [47]. Further, the ADC IP block used in Section 15.2 has a conversion rate of $111.\dot{1}$ kHz [42]. Therefore, ADCs that have more than adequate performance are widely available. This will not be a problem for the physics engine.

15.4.2 Operational Amplifier Bandwidth

Another potential problem caused by running the analogue computer too fast is that problems may occur based on the nonideality of the operational amplifiers. It was mentioned in Section 6.4 that ideal operational amplifiers have infinite bandwidth. In other words, they can process an infinite range of input frequencies. In contrast, practical operational amplifiers have a finite, but usually very large, bandwidth. If the analogue computer is executing faster than real-time, this may lead to higher frequencies and the possibility of exceeding the operational amplifier's bandwidth. Current operational amplifiers have bandwidths in the range of a few kilohertz to a few hundred megahertz [48]. Moreover, high bandwidth operational amplifiers are not of substantially greater cost than their lower bandwidth counterparts. The operational amplifier IP block used in Section 15.2 has a bandwidth of 2.57 MHz [41].

Whether the operational amplifier's bandwidth limit will be reached depends on the system in question. In the case of the mass-spring-damper example developed in Chapter 8, the maximum frequency is $1.\dot{1}$ Hz. This example should be somewhat representative of the type of problems for which the engine would be used. If fifty of these simulations were to be multiplexed, the maximum frequency would not exceed $55.\dot{5}$ Hz. These figures are well within the limits of today's operational amplifiers. Consequently, it appears that the bandwidth of the operational amplifiers will also be unproblematic.

15.4.3 Execution Speed

Another figure that could be calculated is the maximum speed at which the system will execute simulations. Of course, the system would usually be executed in real-time, but here the goal is to find the maximum possible speed.

The analysis was performed using the vehicle suspension system, which is a relatively complex example. The analysis simulated fifty vehicles. It ran for a length of 10 m and the vehicles moved at a speed of 5 m/s. A wedge shaped bump of maximum height 1 m and depth 2 m was placed 1 m from the starting position. The rear wheels were offset 0.7 m from the front wheels, so that objects detected at the front of the vehicle were detected 0.14 s later at the rear.

To simulate fifty vehicles, one hundred vehicle suspension system models must be multiplexed, since each model only accounts for a single axle. The provided simulation uses only one axle since, due to limitations imposed by the simulator (Section 12.1), one hundred axles cannot currently be simulated. If the ADC IP block described in Section 15.2 was used, the system could be run so that the maximum frequency is 110 kHz. If 72 fps (72 Hz) were desired, the system could be executed at $15.2\dot{7}$ times faster than real-time. Therefore, these two seconds would only require $0.13\dot{0}\dot{9}$ seconds to simulate.

However, the system may be optimised. As observed in Section 11.1.2.2, the outputs only change once the inputs have changed. Most of the simulation involves the vehicle driving over a level surface, so that the suspension system remains in equilibrium. In fact, the output is only nonzero from 0.2 to 2.0 seconds. These 1.8 seconds of simulation could be performed $15.2\dot{7}$ times faster than real-time, so that the entire system would finish execution in $0.117\dot{8}\dot{1}$ seconds. Other simulations could obtain a much larger gain from the same optimisation. Therefore, exceptional performance may be obtained from the system.

For comparison purposes, a similar model was built using the ODE software physics engine, based on provided examples. A graphical version (Figure 15.1) was created for reference purposes, but the graphical component was disabled for the analysis. The system used for testing had an AMD Athlon 64 3000+ (2 GHz) CPU with 512 MB of RAM, making it a relatively powerful system. As its operating system, it used the 32 bit version of Fedora Core 3, which used GNU/Linux kernel 2.6.9-1.667. The 64 bit version was not used since ODE failed to compile with the 64 bit X11 libraries. The average CPU time taken to execute the simulation over 100 runs was used. The calculated figure was 4.053 s, much slower than that obtained from the project's physics engine and slower than real-time.

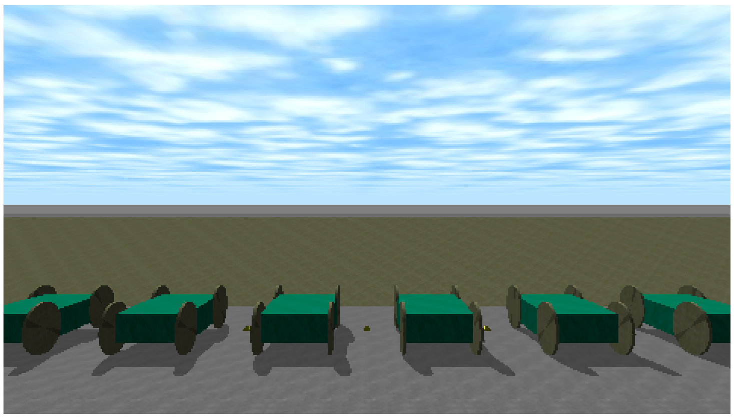


Figure 15.1: Screenshot of vehicle simulation application

Note, however, that the two systems cannot be compared precisely. For example, the software physics engine is modelling more than just the vehicle suspension systems. It is also modelling collision detection and the effects of gravity, for instance. More importantly, the software engine simulates the flight of the vehicles through the air, whereas the hardware engine does not. However, the comparison of the two figures does provide a rough estimate into the relative performance of the two systems.

Chapter 16

Conclusions

This chapter provides some conclusions to the project, outlining the knowledge acquired and potential future work. The chapter concludes with a summary of what was achieved during the project.

16.1 Knowledge Acquired

Before starting this project, I had no knowledge of operational amplifiers and therefore, of analogue and hybrid computers. Since these systems formed the foundation upon which the project was constructed, I now believe I am adept at the design and implementation of such systems.

In addition, although I had some knowledge and experience of analogue components such as resistors and capacitors, my knowledge was greatly expanded during the implementation of this project. Moreover, I acquired techniques for the design of analogue of which I had been previously unaware. I also realised that there are still a great number of applications in which analogue pervades, even in a world that many claim is digital.

Building on my newfound knowledge of analogue devices, I learned how to design and implement dynamically reconfigurable analogue systems. Additionally, I learned how to integrate digital components into such a system, so that reconfiguration may be performed automatically.

For this project, I had to determine how state was stored in an analogue computer. I subsequently utilised this knowledge to multiplex simulations.

My knowledge of VHDL was adequate but not complete. During the course of this project, I furthered my knowledge of the language. I also learned a new language, VHDL-AMS, the analogue and mixed signal extensions to the existing language. Analogue HDLs are a recent development. Indeed, this language was only standardised by the IEEE in 1999 and although still widely unused, it is rapidly gaining in usage and support. Through using VHDL-AMS, I became skilful in using the industry-standard Mentor Graphics software.

I also attained an insight into both FPAAs and FPMAs. I acquired information about the internal architectures of these devices and became *au fait* with their construction and reprogramming. In addition, I was introduced to the process whereby analogue circuits are synthesised to an IC.

Finally, this project contributed to my research skills since, as outlined above, many of the project's concepts were alien to me. This resulted in a considerable amount of research, in both published and electronic resources. The cross-subject nature of this project resulted in the need to acquire information in the fields of both computer science and electronic engineering. Many of the most important analogue computer books were published in the 1960s, while many of the most important digital computer books were published more recently. Therefore, my research had to span decades, since the project involved a fusion of analogue and digital computers.

16.2 Future Work

Firstly, the construction of additional physics systems that are relevant to computer games would be advantageous. Examples of relevant systems includes trajectory, collision response and recoil calculations. In addition, the vehicle suspension system could be expanded to model other physics relevant for vehicle simulation. Other complex systems such as aircraft and ships could also be constructed. These systems could be used to further test and analyse the system.

Next, the existing physics engine could be optimised. Based on data obtained from the constructed physics systems, patterns could be observed in either the behaviour or construction of such systems. These patterns could be used to remove redundant entities from the physics engine, resulting in a potentially smaller die area.

The physics engine could also be modified so that it could potentially be placed on hardware. This would require a reduction in the number of bits used by the interface, which could be achieved by using a serial-to-parallel interface or, preferably, by building certain physics systems into the device, as outlined in Section 15.1. Additionally, the resistors in the design could be replaced by switched capacitors, for reasons outlined in Section 15.2.

Finally, the project could be implemented on an FPAA. This would require the project to be rewritten using the FPAA manufacturer's custom software. This would allow for a hardware demonstration of the physics engine, proving that the concepts can be translated into hardware.

16.3 Summary

The ultimate result of this project is a reconfigurable hybrid computer with a purely digital interface. This computer has the capability to be used as a physics engine because it is adroit at performing sophisticated simulations that would be of use in a physics engine, such as the vehicle suspension system. Moreover, the computer executes in real-time, which is necessary for computer games. Accuracy is high albeit with some minor deviations, which are unimportant to computer games.

In addition, the physics engine is capable of multiplexing three simulations. This allows multiple simulations to run concurrently, which is pivotal for games, as they often contain a multitude of objects that need to be simulated concurrently.

Based on these points, the constructed physics engine would be suitable for use as a hardware equivalent to today's software physics engines, if it were to be synthesised and connected to a PC. Although some minor issues remain, the project forms a foundation for future work and proves the viability of the original concept. Therefore, this project was successful in achieving its goals.

Bibliography

[1] D. H. Eberly, *Game Physics*, ser. The Morgan Kaufmann Series in Interactive 3D Technology, D. H. Eberly, Ed. San Francisco, California, USA: Morgan Kaufmann, 2004.

[2] (2005) PhysX. AGEIA. Mountain View, California, USA. [Online]. Available: http://www.ageia.com/technology.html

[3] (2005) Havok. Havok.com. Dublin, Ireland. [Online]. Available: http://www.havok.com/

[4] (2005) Meqon. Meqon Research. Linköping, Sweden. [Online]. Available: http://www.meqon.com/

[5] (2005) RenderWare Physics. Criterion Software. Guildford, Surrey, UK. [Online]. Available: http://www.renderware.com/physics.asp

[6] (2005) NovodeX. AGEIA. Mountain View, California, USA. [Online]. Available: http://www.ageia.com/novodex.html

[7] (2005) SD/FAST. PTC. Needham, Massachusetts, USA. [Online]. Available: http://www.sdfast.com/

[8] R. Smith. (2005, Feb.) Open Dynamics Engine. ODE. Mountain View, California, USA. [Online]. Available: http://ode.org/

[9] S. McMillan. (2001, July) DynaMechs. DynaMechs. Columbus, Ohio, USA. [Online]. Available: http://dynamechs.sourceforge.net/

[10] H. Keller. (2001, Feb.) AERO. AERO. Stuttgart, Germany. [Online]. Available: http://www.aero-simulation.de/

[11] D. M. Bourg, *Physics for Game Developers*, R. Denn, Ed. Sebastopol, California, USA: O'Reilly, 2002.

[12] R. Smith. (2002, Nov.) IGC 2002 slides. ODE. Mountain View, California, USA. [Online]. Available: http://ode.org/slides/igc02/index.html

[13] "Analogue," in *The Oxford English Dictionary*, 2nd ed., J. A. Simpson and E. S. C. Weiner, Eds. Oxford, UK: Clarendon Press, 1989, vol. I, pp. 431–432. [Online]. Available: http://dictionary.oed.com/cgi/entry/50007887

[14] P. A. Holst, "A note of history," *Simulation*, pp. 131–135, Sept. 1971.

[15] J. S. Small, “General-purpose electronic analog computing: 1945-1965,” *IEEE Annals of the History of Computing*, vol. 15, no. 2, pp. 8–18, 1993. [Online]. Available: http://ieeexplore.ieee.org/iel4/85/5317/00207740.pdf

[16] C. Bissell, “A great disappearing act: the electronic analogue computer,” in *IEEE Conference on the History of Electronics*, June 2004. [Online]. Available: http://telematics.open.ac.uk/people/bissell/chris/bletchley_paper.pdf

[17] J. S. Small, *The Analogue Alternative: The Electronic Analogue Computer in Britain and the USA, 1930–1975*, ser. Studies in the History of Science, Technology and Medicine. London, UK: Routledge, 2001.

[18] P. A. Holst, “Analog computer,” in *Encyclopedia of Computer Science*, 4th ed., A. Ralston, E. D. Reilly, and D. Hemmendinger, Eds. London, UK: Nature Publishing Group, 2000, pp. 53–59.

[19] D. Welbourne, *Analogue Computing Methods*. Oxford, UK: Pergamon Press, 1965.

[20] R. Gayakwad and L. Sokoloff, *Analog and Digital Control Systems*. Englewood Cliffs, New Jersey, USA: Prentice Hall, 1988.

[21] M. Brand and T. Brand, *Analogue Computers*, ser. Contemporary Mathematics, A. Sherlock, Ed. London, UK: Edward Arnold, 1970.

[22] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis*, 2nd ed., ser. Texts in Applied Mathematics, F. John, J. E. Marsden, L. Sirovich, M. Golubitsky, and W. Jäger, Eds. New York, USA: Springer-Verlag, 1993.

[23] D. Greco and K. Kuehl, “Hybrid computer lab: A new tool for science,” *The Wisconsin Engineer*, vol. 77, no. 2, pp. 10–12, Nov. 1972. [Online]. Available: http://digital.library.wisc.edu/1711.dl/UW.WIEv77no2

[24] P. E. Allen and D. R. Holberg, *CMOS Analog Circuit Design*, 2nd ed., ser. The Oxford Series in Electrical and Computer Engineering, A. S. Sedra, Ed. New York, USA: Oxford University Press, 2002.

[25] B. Razavi, *Design of Analog CMOS Integrated Circuits*, ser. McGraw-Hill Series in Electrical and Computer Engineering. New York, USA: McGraw-Hill, 2001.

[26] D. Báez-López, F. S. Sánchez, and G. E. F. Verdad, “Multimedia tutorial on operational amplifiers, fundamentals, and applications,” in *28th Annual Frontiers in Education Conference, 1998 (FIE ’98)*, vol. 2, Nov. 1998, pp. 541–546. [Online]. Available: http://ieeexplore.ieee.org/iel4/5943/15948/00738730.pdf

[27] (2004, Oct.) LM741 operational amplifier. National Semiconductor. Santa Clara, California, USA. [Online]. Available: http://www.national.com/ds/LM/LM741.pdf

[28] (2000, Sept.) μA741, μA741Y general-purpose operational amplifiers. Texas Instruments. Dallas, Texas, USA. [Online]. Available: http://focus.ti.com/lit/ds/symlink/ua741.pdf

[29] G. B. Clayton, *Operational Amplifiers*, 2nd ed. London, UK: Butterworths, 1980.

[30] *IEEE Std 1076.1-1999: IEEE Standard VHDL Analog and Mixed-Signal Extensions*, IEEE Computer Society Std., Mar. 1999. [Online]. Available: http://ieeexplore.ieee.org/iel5/6556/17511/00808837.pdf

[31] U. Heinkel, M. Padeffke, W. Haas, T. Buerner, H. Braisz, T. Gentner, and A. Grassmann, *The VHDL Reference: A Practical Guide to Computer-Aided Integrated Circuit Design Including VHDL-AMS*. Chichester, West Sussex, UK: John Wiley & Sons, 2000.

[32] A. Doboli, A. Nunez-Aldana, N. Dhanwada, S. Ganesan, and R. Vemuri, "Behavioral synthesis of analog systems using two-layered design space exploration," in *DAC '99: Proceedings of the 36th ACM/IEEE conference on Design automation*, M. J. Irwin, Ed. New York, NY, USA: ACM Press, June 1999, pp. 951–957. [Online]. Available: http://ieeexplore.ieee.org/iel5/6338/16938/00782234.pdf

[33] T. J. Kazmierski and F. A. Hamid, "Analogue integrated circuit synthesis from VHDL-AMS behavioural specifications," in *Proceedings of the 23rd International Conference on Microelectronics (MIEL 2002)*, vol. 2, May 2002, pp. 585–588. [Online]. Available: http://ieeexplore.ieee.org/iel5/7858/21637/01003325.pdf

[34] J. Hargrave and P. Zorkoczy, *Analogue Computing/Control*, ser. The Man-made World: Technology Foundation Course. Milton Keynes, UK: The Open University Press, 1974.

[35] D. Cebon, *Handbook of vehicle-road interaction: vehicle dynamics, suspension design, and road damage*, ser. Advances in engineering, D. Cebon, F. Ma, and E. Kreuzer, Eds. Lisse, Netherlands: Swets & Zeitlinger, 1999.

[36] D. O. Nessman and P. P. Rodriguez, "Using ASICs for component integration," in *Proceedings of the 40th Electronic Components and Technology Conference, 1990*, vol. 1, May 1990, pp. 693–699. [Online]. Available: http://ieeexplore.ieee.org/iel5/157/3478/00122265.pdf

[37] (2005) TRAC. Zetex Semiconductors. Manchester, UK. [Online]. Available: http://www.zetex.com/3.0/a5-6.asp

[38] (2005) ispPAC devices. Lattice Semiconductor. Hillsboro, Oregon, USA. [Online]. Available: http://www.latticesemi.com/products/ispPAC/

[39] (2005) Anadigm. Anadigm. Crewe, UK. [Online]. Available: http://www.anadigm.com/

[40] N. H. E. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 3rd ed. Boston, Massachusetts, USA: Pearson, 2005.

[41] (2004, Oct.) OP05B CMOS operational amplifier. austriamicrosystems. Schloss Premstätten, Austria. [Online]. Available: http://asic.austriamicrosystems.com/databooks/c35_a/op05b_c35_reva.pdf

[42] (2004, July) ADC8 CMOS 8-bit ADC. austriamicrosystems. Schloss Premstätten, Austria. [Online]. Available: http://asic.austriamicrosystems.com/databooks/c35_a/adc8_c35_reva.pdf

[43] (2004, Oct.) DAC8 8-bit digital to analog converter. austriamicrosystems. Schloss Premstätten, Austria. [Online]. Available: http://asic.austriamicrosystems.com/databooks/c35_a/dac8_c35_reva.pdf

[44] (2003, May) NAND21. austriamicrosystems. Schloss Premstätten, Austria. [Online]. Available: http://asic.austriamicrosystems.com/databooks/c35/databook_c35_33/NAND21.html

[45] (2003, May) DFC1. austriamicrosystems. Schloss Premstätten, Austria. [Online]. Available: http://asic.austriamicrosystems.com/databooks/c35/databook_c35_33/DFC1.html

[46] B. G. Haskell, A. Puri, and A. N. Netravali, *Digital Video: An Introduction to MPEG-2*, ser. Digital Multimedia Standards. Norwell, Massachusetts, USA: Kluwer Academic, 1997.

[47] (2005, Apr.) ADC selection table. Analog Devices. Norwood, Massachusetts, USA. [Online]. Available: http://www.analog.com/IST/SelectionTable/?selection_table_id=124

[48] (2005, Apr.) Operational amplifiers. Analog Devices. Norwood, Massachusetts, USA. [Online]. Available: http://www.analog.com/en/subCat/0,2879,759%255F786%255F0%255F%255F0%255F,00.html