THE UNIVERSITY OF CALGARY

Hardware Acceleration of the Finite-Difference Time-Domain (FDTD) Method

by

Ryan N. Schneider

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CALGARY, ALBERTA

DECEMBER, 2002

THE UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled "Hardware Acceleration of the Finite-Difference Time-Domain (FDTD) Method", submitted by Ryan Schneider in partial fulfillment of the requirements for the degree of Master of Science.

Supervisor, Dr Michal Okoniewski
Dept. of Electrical and Computer Engineering

Dr Graham Jullien
Dept. of Electrical and Computer Engineering

Dr Laurence Turner
Dept. of Electrical and Computer Engineering

Dr Mike Potter
Dept. of Electrical and Computer Engineering

Dr Claudio Costi
Dept. of Computer Science

Jan 22nd, 2003
Date

Abstract

Electromagnetics are the foundation for many of the prolific technologies in modern society and very important to many branches of engineering. The finite-difference time-domain (FDTD) method has been successfully and very widely applied to the modelling of electromagnetic phenomena. The algorithm is computationally intensive, however, and simulations can run for hours to several days on multiprocessor supercomputers. Dramatically reducing the runtime of this method would greatly benefit FDTD users and open up new areas of research.

The goal of this research is to prove the concept of accelerating FDTD by using programmable hardware, integer arithmetic, and fine-grained parallelism. The concept is successfully proven using a pipelined bit-serial implementation of the FDTD algorithm on field-programmable gate-array (FPGA) hardware. The details of this implementation are described and the speed and accuracy are compared to software FDTD implementations. Finally, a resource-sharing approach for an FDTD hardware accelerator is outlined.

Acknowledgements

I would like to take the opportunity to recognize a number of people for their academic support of my Master's research and personal support of my life. First, I would like to thank Dr. Michal Okoniewski and Dr. Laurence Turner for providing tremendous support during the research and thesis writing process and for just being a couple of cool guys.

Second, my family and friends have also been very important in helping me to get this far. I would like to thank Rolf, Rose and Rachel Schneider for their ongoing support of my academic and life endeavors. I would also like to acknowledge Lorien Bouchard, Sarah Goard-Baker, all of my friends and my fellow students for putting up with me, helping me to have fun, helping me to relieve stress and supporting my Master's goals.

Finally, I would like to recognize the partial support of this research by the Natural Sciences and Engineering Research Council (NSERC) of Canada and the Alberta Informatics Circle of Research Excellence (iCORE).

Table of Contents

List of Tables

List of Figures

List of Symbols and Acronyms

Symbols

The following list contains the common symbols, in order of appearance, used by this thesis.

t	time $[s]$
$\vec{E}$	electric field $[V/m]$
$\vec{D}$	electric flux density $[C/m^2]$
$\vec{H}$	magnetic field $[A/m]$
$\vec{B}$	magnetic flux density $[Wb/m^2]$
$\vec{J}_e$	electric current conduction density $[A/m^2]$
ρ	volume charge density $[C/m^3]$
μ	magnetic permeability $[H/m]$
ε	electric permittivity $[F/m]$
σ	electric conductance (loss) $[\mho/m]$
ρ'	magnetic resistivity (loss) $[\Omega/m]$
$\Delta x, \Delta y, \Delta z$	spatial sampling interval(s) $[m]$
Δt	temporal sampling interval $[s]$
c	speed of light $[m/s]$
S_{factor}	stability factor (0.0 to 1.0) [dimensionless]
C	capacitance $[F]$
L	inductance $[H]$
ε_r	relative electric permittivity (to free space) [dimensionless]
μ_r	relative magnetic permeability (to free space) [dimensionless]

Acronyms

The following list contains the most frequently used acronyms, in order of appearance, in this thesis.

FDTD finite-difference time-domain
: A discrete (temporal and spatial) approximation to the integration of Maxwell's equations that is well suited to implementation on a computer. (See also Section 2.2.)

FPGA field-programmable gate-array
: A fixed array of programmable digital hardware resources implemented as an integrated circuit. (See also Section 2.1.)

PML perfectly matched layer
A type of absorbing boundary condition that is a very accurate approximation to a boundary at infinity.

HDL hardware description language
Similar to a programming language and used to specify the structure or behavior of a digital circuit.

VHDL VHSIC Hardware Description Language
An advanced form of a hardware description language.

VHSIC Very High Speed Integrated Circuit

PC personal computer

SDRAM synchronous DRAM

DRAM dynamic random access memory
A type of semiconductor memory in which the information is stored in capacitors on a MOS integrated circuit. Typically each bit is stored as an amount of electrical charge in a storage cell consisting of a capacitor and a transistor. Due to leakage the capacitor discharges gradually and the memory cell loses the information. Therefore, to preserve the information, the memory has to be refreshed periodically.

PCI Peripheral Connections Interface
A standardized hardware interface, commonly found in personal computers, that is used to connect generic "peripherals" to the computers processor and main memory.

AGP Accelerated Graphics Port
A standardized hardware interface, commonly found in personal computers, that is optimized for high throughput of video data in a personal computer.

LUT lookup table
For the Xilinx Virtex technology, a lookup table is a 16x1 table of programmable values, with four inputs (addressing) and one output. Any four-input, single output logic function can be implemented by appropriate choice of the table values.

LE logic element
A Xilinx Virtex logic element contains one D-type flip-flop, one lookup table and carry and control logic.

CLB configurable logic block
A Xilinx Virtex CLB contains four logic elements, as described for the previous acronym.

ASIC application-specific integrated-circuit

RF radio-frequency

VLSI very large scale integration

LDI lossless discrete integrator

ALU arithmetic logic unit

LSB least-significant bit

MSB most-significant bit

SWL system wordlength

PEC perfect electric conductor
A boundary, used in an FDTD simulation, which maintains a tangential electric field of zero.

PMC perfect magnetic conductor
A boundary, used in an FDTD simulation, which maintains a tangential magnetic field of zero.

BIBO bounded-input, bounded-output

Glossary of Terms

The following list defines several terms used, in order of appearance, in the thesis.

cache — A small fast memory holding recently accessed data, designed to speed up subsequent access to the same data. When data is read from, or written to, main memory a copy is also saved in the cache, along with the associated main memory address. The cache monitors addresses of subsequent reads to see if the required data is already in the cache. If it is (a cache hit) then it is returned immediately and the main memory read is aborted (or not started). If the data is not cached (a cache miss) then it is fetched from main memory and also saved in the cache. The cache is built from faster memory chips than main memory so a cache hit takes much less time to complete than a normal memory access.

dynamic range — The ratio of the smallest to the largest number that can be represented.

data bandwidth — ("data rate", "data transfer rate" or "transmission rate".) The amount of data transmitted per second by a communications channel or a computing or storage device.

memory bandwidth — Data transfer rate for a memory channel.

bitstream — A stream of serial bits which contain digital information.

interconnect — A programmable mesh of routing "wires" which can connect various internal input/output signals in a digital design together.

Virtex slice	A Virtex slice is a unit of hardware resources in the Xilinx Virtex family of devices and contains two D-type flip-flop's, two lookup tables, carry and control logic.
cache miss	(see also *cache.*) The data required by the processor is currently not stored in the cache so it must be fetched from main memory. Cache is typically much faster than main memory so it is desirable to have as many cache "hits" as possible, to improve performance.
cache lines	(Or cache block) The smallest unit of memory that can be transferred between the main memory and the cache.
Totem	Okoniewski's finite-difference time-domain (FDTD) research code, written in FORTRAN-90.
page fault	In a paged virtual memory system, an access to a page (block) of memory that is not currently mapped to physical memory. When a page fault occurs the operating system either fetches the page in from secondary storage (usually disk) if the access was legitimate or otherwise reports the access as illegal.
minor page fault	(see also *page fault.*) By definition, minor page faults do not require physical I/O. For example, reclaiming the page from the free list would avoid I/O and generate a minor page fault. More commonly, minor page faults occur during process startup as references to pages which are already in memory.
major page fault	(see also *page fault.*) By definition, major page faults require physical I/O, usually with secondary storage like a hard drive. In this instance, the desired "page" in virtual memory cannot be found or reclaimed in main memory so a fresh page must be loaded.
system wordlength	Describes the number of bits used to represent data in the digital system.

Chapter 1

Introduction

"I can see the time when every city will have one."

- American Mayor's reaction to the news of the invention of the telephone

1.1 Motivation

In today's technology-enabled society, electrical engineering and, specifically, electromagnetics play an increasingly important role. The continuing advances in areas such as cellular communications, fiber optics, smart antennas, mobile technologies and multi-gigahertz electronics have necessitated a computer-assisted design approach to model the complex electromagnetic interactions and problems that arise. The ability to understand and predict the behavior of complex electromagnetic structures is of great value to both academia and industry.

The finite-difference time-domain (FDTD) method [1] has been successfully and very widely applied to the modelling of electromagnetic phenomena [2]. The algorithm involves the computation of millions of three-dimensional (electric and magnetic) field components for thousands of discrete time steps. The time-domain results model the electromagnetic field behavior in a physical "volume of interest". The method is both flexible and accurate for a wide range of problems but is also computationally intensive. The past decade has seen a large increase in computational power at declining costs, but FDTD simulations can still run for several days on

multiprocessor supercomputers. Dramatically reducing the runtime of this method would greatly benefit FDTD users and open up new areas of research.

1.2 Thesis Objectives

The long-term goal of this research is to develop a hardware accelerator, capable of accelerating *existing* FDTD software implementations by an order of magnitude or more. The short-term goal, covered in this thesis, is to "prove the concept" of accelerating the FDTD algorithm using (i) programmable hardware, (ii) integer arithmetic, and (iii) fine-grained parallelism. The important performance metrics are: the computation speed, the amount of hardware required, and the simulation accuracy. The chosen approach involves mapping the computationally-intensive FDTD algorithm from the traditional sequential and multiprocessing computer environments onto custom or programmable hardware. The combination of knowledge from the digital design and microwave engineering fields provides a novel solution for accelerating the FDTD algorithm and represents a significant contribution to the FDTD area.

1.3 Accelerating FDTD

1.3.1 State of the Art

Traditionally, FDTD is accelerated by implementing parallel-processing techniques. Parallel-processing describes the act of performing a number of simultaneous computations in 'parallel'. In the computer/software domains this typically involves a complex interconnection of many (computer) processors and memory. Most parallel-processing implementations, in software, use the following two techniques: (i) shared

memory and (ii) distributed memory.

Shared-Memory

Shared memory implementations divide the computational "work" among multiple processors, which all access the same memory space. It is the responsibility of the programmer and compiler to ensure that calculations can be performed independently on several processors; in some cases, the algorithm needs to be re-formulated. The FDTD formulation is well-suited to a shared-memory implementation and can be easily parallelized; the reasons for this are discussed in greater detail in Chapter 2.

The shared memory method typically results in, at best, a linear speedup where acceleration is directly proportional to the number of processors used. Certain special cases may achieve greater acceleration but for most algorithms linear speedup is the theoretical limit. The achieved acceleration is usually lower than the theoretical maximum and tends to saturate because of the overhead needed for (i) coordination of a threaded or shared memory environment and (ii) maintaining consistency among cache memories for a large number of CPU's. While this approach is effective in reducing simulation runtime, shared memory computers with four processors or more can become prohibitively expensive. For example, near cutting edge UNIX-based, shared-memory computers may cost $50,000cdn or more [3].

Distributed Memory

Distributed memory implementations, for example Beowulf clusters [4], also divide the computational work among several processors. For this discussion, a cluster is defined as a number of computers with their own distinct memory and processor(s), connected by a communications network. The FDTD algorithm and data

are partitioned so that many (nearly) self-contained pieces are distributed among the individual computers. Process coordination and control is achieved by passing messages between computers. FDTD boundaries between computers are also updated using message-passing. This approach is more complicated, rendering it more difficult to implement than the shared memory approach.

Theoretically this method also results in linear speedup, if sufficient communication bandwidth is available. However, the communications overhead in current implementations causes any achieved acceleration to quickly saturate.

Performance of Parallel-Processing Implementations

Parallel-processing implementations of FDTD are an area of ongoing research. Tatalias and Bornholdt [5] report a speed gain of 80% of linear speedup using Taflove's FDTD code on the JPL Hypercube. However, they do not specify the number of processors. Okoniewski [3] reports nearly linear speed up for up to 12 processors, before saturating, using Silicon Graphics, Origin Class computers. Schiavone, *et al* [4], and Gillan and Fusco [6] describe distributed memory implementations of FDTD. Using a large FDTD mesh, Gillan and Fusco [7] report near linear speedup for ten processors before starting to saturate.

1.3.2 Literature Review

This work is a multi-disciplinary approach between digital/hardware design, microwave engineering and software; areas that might not typically be associated. While this overlapping of areas makes the research both interesting and novel, it also involves a larger domain of knowledge.

General References - FDTD

Yee [1], Kunz and Luebbers [8] and Taflove [9] provide the theoretical background and description for the FDTD algorithm. Furthermore, Taflove's second book [2] presents a literature survey describing the research advances made in the FDTD area in the past decade.

This research proposes two novel concepts in the FDTD area. A custom and/or programmable hardware implementation and integer arithmetic are introduced to provide the desired FDTD acceleration. The limited references that exist, for the preceding topics in the FDTD literature, are described in the following sections.

Integer FDTD

Because the FDTD algorithm is a numerical method, the numerical accuracy of the computations is very important and round-off errors are a concern. Some simulations also require a large dynamic range[1] ($>$ 100dB), in order to accurately represent broad magnitude variations that co-exist in the simulation. To meet the criteria of numerical accuracy and dynamic range floating-point arithmetic is typically used. Although a very-wide integer representation (60-bits) could be used, most hardware platforms and compilers either do not support this or the computations would be slower. All existing academic and commercial FDTD codes use single- or double-precision floating-point arithmetic.

One paper describes an implementation of the FDTD algorithm using integer computations. Grinin created a 16-bit integer FDTD code in order to take advantage of an integer-only or integer-optimized microprocessor [10].

[1]dynamic range: the ratio of the smallest to the largest number that can be represented.

Hardware FDTD

There are two hardware implementations of the FDTD algorithm described in the literature. In order to discuss the reported accelerations, two types of equations are introduced for the FDTD simulation: update and boundary equations.

The FDTD update equations form the kernel of the algorithm. An FDTD simulation, with simple boundaries, spends 95% or more of its runtime calculating these update equations (Section 3.3.2). Thus, accelerating the update equations by a certain amount will also reduce the total runtime by a very similar factor.

Specialized equations are used at the boundaries of the finite simulation volume. These boundary equations can be used to make the finite simulation space appear to extend to infinity. perfectly matched layer (PML) boundary equations [2] are a recent advance in this area and are a very accurate approximation to a boundary at infinity. PML's can add as much as 70% to the required number of computations, extending the runtime by an equal amount; in some cases, a large portion of the simulation runtime is attributed to boundary computations. To achieve worthwhile acceleration, it is then desirable to accelerate the boundary computations as well.

Marek, *et al* [11], describe a simulated hardware description language (HDL) design intended as a co-processor or accelerator for Sparc workstations. They predict a five-fold acceleration of the main FDTD update equations and a nine-fold acceleration of the PML equations, but they never attempted the actual implementation. Placidi, *et al* [12], describe a simulated VHDL[2] [13] design intended for the personal computer (PC) platform. This work was again limited to simulation of the hardware;

[2]VHDL stands for VHSIC Hardware Description Language, where VHSIC stands for Very High Speed Integrated Circuit

they predict a four-fold acceleration for the FDTD update equations.

No references were found in the literature where hardware has actually been constructed and the performance measured.

General References - Digital Design

Basic digital design techniques and knowledge are from three primary sources: Mano's text [14], Wakerly's text [15] and Andraka's website [16].

Information about the Xilinx Virtex family of field-programmable gate-arrays (described in Section 2.1) is from Xilinx [17] and the XESS product documentation [18].

Information about bit-serial, the specific arithmetic implementation technique, is from three main sources: a text by Hartley and Parhi [19], a text by Denyer and Renshaw [20], and a text by Oberman [21].

1.4 Thesis Outline

Chapter 2 presents the relevant background and theory for this research. Three main concepts are discussed: field-programmable gate-arrays FPGA's, the theoretical basis of the FDTD algorithm and an alternative FDTD representation using inductors and capacitors.

Chapters 3 to 5 investigate four avenues in order to achieve the thesis goals:

- Chapter 3 presents two of the research avenues namely computer technology and software benchmarks. The first part of this chapter describes the ex-

pected data bandwidth[3] of three main computer technologies: SDRAM memory, the Peripheral Connections Interface (PCI) hardware bus and the Accelerated Graphics Port (AGP) hardware bus. This information is included mainly: (i) to provide an expected/maximum data performance for the FDTD algorithm on the reference computer and (ii) to provide hardware bus information for the future hardware accelerator. The remainder of the chapter presents (i) the methodology used and (ii) the computation speed and memory bandwidth[4] benchmarks obtained for two software FDTD implementations on a reference computer. These benchmarks facilitate comparison of any achieved acceleration to a baseline computer/software implementation. Ultimately, this information is used to predict the computation speed and memory bandwidth required to achieve an order of magnitude acceleration.

- Chapter 4 describes the first approach to a hardware FDTD implementation, namely an "FDTD Computational Engine". In this approach, the entire simulation is implemented on programmable hardware. One- and two-dimensional hardware FDTD implementations are verified using microwave cavity resonators. The acceleration and simulation accuracy achieved for both implementations are discussed.

- Chapter 5 describes the proposed design for the future hardware accelerator as an "FDTD Co-Processor". It involves resource sharing, such that the same hardware resources are re-used for the FDTD algorithm.

[3]Data bandwidth ("data rate", "data transfer rate", "transmission rate") is the amount of data transmitted per second by a communications channel or a computing or storage device [22].

[4]Data transfer rate for a memory channel.

Chapter 6 provides a summary of the results and conclusions for the thesis, concluding with a discussion of future work and direction for the research.

Chapter 2

Background and Theory

"God runs electromagnetics by wave theory on Monday, Wednesday, and Friday, and the Devil runs them by quantum theory on Tuesday, Thursday, and Saturday. " – Sir William Bragg

This chapter is intended to provide a foundation for the main concepts used in subsequent chapters. It provides background information on three things: field-programmable gate-arrays, the theoretical basis of the FDTD algorithm and the theoretical basis of an inductor-capacitor FDTD representation. Subsequent chapters will make extensive use of the concepts described in the following sections.

2.1 Field-Programmable Gate-Array (FPGA) Technology

Field-programmable gate-arrays (FPGA's) offer an ideal platform for experimental digital designs, as they are easily configurable. Accordingly, the hardware platform chosen to achieve the thesis goals is an FPGA. Reasons for this choice are provided in Section 2.1.2. A specific definition of an FPGA is provided here to facilitate the discussion in subsequent sections and chapters. The definition's scope is confined to the particular product line used for this research, the Xilinx Virtex family of FPGA's.

In essence, an FPGA is a fixed array of programmable digital hardware resources implemented as an integrated circuit. How the resources are used and

the function that it performs is defined by a "bitstream"[1] when the FPGA is programmed/configured. Similar to computer memory, an FPGA can be programmed with new bitstreams many times.

The development of FPGA designs/configurations resembles software compilation, in that a desired behavior or structure is described using a hardware description language (HDL) and then synthesized using the equivalent of a hardware compiler. This process produces a configuration bitstream (the "executable") for downloading to the FPGA.

There are four important hardware resources for any synchronous digital design:

1. Synchronous Signal Storage - Flip-flops provide clock-edge triggered storage for one logic signal.

2. Combinational Logic - The Xilinx Virtex family uses 4-input look-up tables (LUT's) to implement combinational logic. For the Virtex technology, a lookup table is a 16x1 table of programmable values, with four inputs (addressing) and one output. Thus, any four-input, single output logic function can be implemented by appropriate choice of the table values. Logic functions with a larger number of inputs or outputs are implemented by combining multiple LUT's.

3. Wires / Interconnect - This is the fundamental infrastructure through which the digital signals are transmitted throughout the digital circuit.

4. Input / Output Pins - This is the means through which the digital device interacts with the environment.

[1]Bitstream: A stream of serial bits which contain digital information.

Recently, FPGA manufacturers increased functionality by adding dedicated integer multipliers and memory blocks to their devices. Multipliers and small internal memories are common elements in many digital designs. In the past, valuable FPGA resources were used to implement these elements.

With the Virtex family, Xilinx also began to support "runtime reconfiguration". This concept allows parts of an FPGA design to be modified in situ, while the rest of the circuit is still running. Reconfiguration can be used to change logic functions, redirect data paths or add/change/remove functionality to a specific part of an FPGA configuration while the remaining hardware continues to operate.

For the Xilinx Virtex products, hardware resources are grouped into common blocks called logic elements (LE's), configurable logic blocks (CLB's) or slices. Specifically for the Virtex family, two LE's are one slice. Two slices are one configurable logic block (CLB). A given FPGA implements an array of CLB's connected by a large mesh of configurable routing resources. These routing resources connect the individual hardware resources to each other on both a micro and macro scale. Information contained in the configuration bitstream is used to program the CLB's (internal CLB routing, LUT's, multiplexers and flip-flops) and (global) routing resources of the FPGA. The resulting logic and interconnect[2] embody the digital design created by the user.

[2]Interconnect: A programmable mesh of routing "wires" which can connect various internal input/output signals in a digital design together.

2.1.1 The Virtex Slice

A Virtex slice is a unit of hardware resources in the Xilinx Virtex family of devices. It provides a useful metric for the size of a particular design and the minimum size of device that would be required. In Chapter 4 Virtex slices are used to evaluate the hardware cost for various FDTD implementations.

A Virtex slice contains two D-type flip-flop's, two LUT's, carry and control logic. Figure 2.1 depicts a pair of Virtex slices [17].

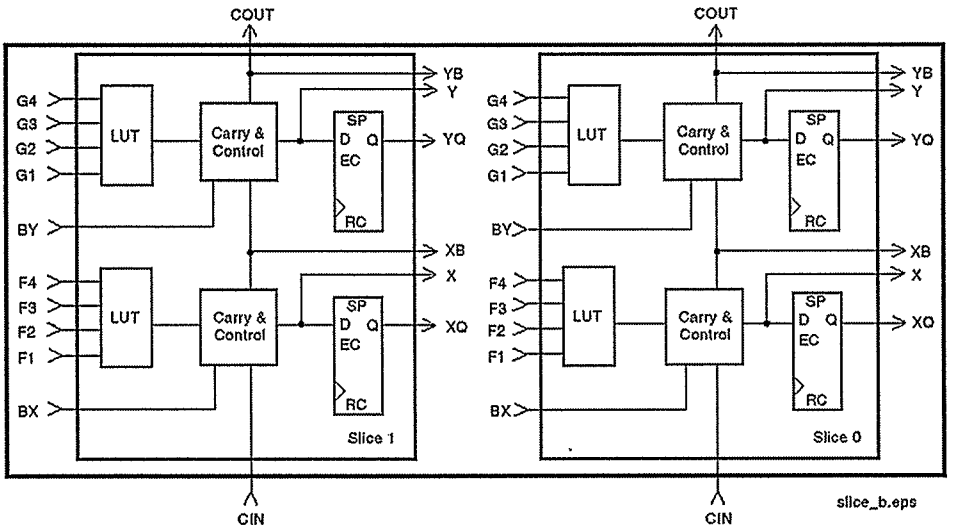


Figure 2.1: One Virtex CLB == Two Virtex Slices

2.1.2 Benefits of FPGA's

FPGA's are chosen for the hardware implementation for the following reasons:

- Flexibility - Unlike a dedicated application-specific integrated-circuit (ASIC) the hardware design can be changed and recompiled easily. As mentioned earlier, the FPGA configuration is produced using a process analogous to software

compilation. Many different designs can be explored using the same FPGA. The tradeoff is that FPGA designs, (i) generally do not represent the fastest speed or lowest number of transistors[3] for a given digital implementation and (ii) typically dissipate more power and are relatively expensive to distribute, compared to a mass-produced ASIC.

- Rapid Prototyping - Similar to the point above, FPGA's facilitate rapid prototyping. Designs can be implemented and verified within a matter of hours or days. Compiled designs are immediately available for testing on the FPGA. For custom integrated circuits, the design turnaround time can easily be on the order of months or longer.

- Transferability - FPGA designs can be translated to non-optimized, custom silicon implementations by software tools. The translated FPGA designs will not perform (speed, hardware size) as well as silicon designs produced by an ASIC designer, but FPGA's can be used as an intermediate step before moving to a custom ASIC design.

2.2 The Basic FDTD Algorithm

The finite-difference time-domain (FDTD) algorithm is the primary focus of this research. In all of the following chapters, the performance of the FDTD algorithm on software and hardware is discussed. The most important information to gain from this section is the formulation of the six FDTD update equations, described in

[3]The number of transistors will impact the power consumption/dissipation and silicon area required for a given digital design.

Section 2.2.2.

FDTD is a discrete (temporal and spatial) approximation to the integration of Maxwell's equations that is well suited to implementation on a computer. Maxwell's equations are the classical physics equations used to mathematically describe dynamic electromagnetic behavior. Engineers in numerous fields, working with frequencies ranging from transmitted power (50-60Hz) to radio-frequency (RF) to photonics (THz), use the FDTD algorithm to create computer simulation models of electromagnetic structures. After simulating, they can determine the structures' behaviors as a function of time, given some excitation. The FDTD method is both flexible and accurate for a wide range of problems, making the algorithm one of the most popular and successful in the area of electromagnetic modelling.

Kane Yee invented the FDTD algorithm in 1966 [1] but it was not used because it was too computationally intensive for the technology of that time. The past decades have seen a rapid increase in computational resources at declining costs, which has catalyzed the success of this technique in recent years.

In order to derive the FDTD algorithm, synonymous with Yee's Method, Maxwell's Equations are first described.

2.2.1 Maxwell's Equations in Three Dimensions

Most of the following material is adapted from Taflove [9]. A more detailed discussion of the following concepts is included in Appendix A.

Maxwell's equations mathematically describe the dynamic behavior of electric and magnetic fields. In differential form they are stated as:

$$\frac{\partial \vec{B}}{\partial t} = -\nabla \times \vec{E} \tag{2.1}$$

$$\frac{\partial \vec{D}}{\partial t} = \nabla \times \vec{H} - \vec{J}_e \tag{2.2}$$

$$\nabla \cdot \vec{D} = \rho \tag{2.3}$$

$$\nabla \cdot \vec{B} = 0 \tag{2.4}$$

Where t = time $[s]$

$\vec{E}$ = electric field $[V/m]$

$\vec{D}$ = electric flux density $[C/m^2]$

$\vec{H}$ = magnetic field $[A/m]$

$\vec{B}$ = magnetic flux density $[Wb/m^2]$

$\vec{J}_e$ = electric current conduction density $[A/m^2]$

ρ = volume charge density $[C/m^3]$

Maxwell's equations can also be expressed in integral form, see [8] for an example.

$\vec{B}$,$\vec{H}$ and $\vec{D}$,$\vec{J}_e$,$\vec{E}$ are simply related by the following constitutive equations:

$$\vec{B} = \mu \vec{H} \tag{2.5}$$

$$\vec{D} = \varepsilon \vec{E} \qquad\qquad \vec{J}_e = \sigma \vec{E} \tag{2.6}$$

Where μ = magnetic permeability $[H/m]$

ε = electric permittivity $[F/m]$

σ = electric conductance (loss) $[\mho/m]$

For linear, isotropic, and non-dispersive materials (whose properties are independent of field-intensity, direction or frequency) ε and μ are scalar quantities.

Substituting Equations 2.5 to 2.6 into Equations 2.1 to 2.2, under the assumptions of linear, isotropic and non-dispersive materials, yields:

$$\frac{\partial \vec{H}}{\partial t} = \frac{1}{\mu} \cdot \left(-\nabla \times \vec{E} \right) \tag{2.7}$$

$$\frac{\partial \vec{E}}{\partial t} = \frac{1}{\varepsilon} \cdot \left(\nabla \times \vec{H} - \sigma \vec{E} \right) \tag{2.8}$$

For the lossless case, Equations 2.7 to 2.8 further simplify to:

$$\frac{\partial \vec{H}}{\partial t} = \frac{1}{\mu} \cdot \left(-\nabla \times \vec{E} \right) \tag{2.9}$$

$$\frac{\partial \vec{E}}{\partial t} = \frac{1}{\varepsilon} \cdot \left(\nabla \times \vec{H} \right) \tag{2.10}$$

Equation 2.9 simply states that a circulating electric field creates a time-varying (perpendicular) magnetic field. Similarly from Equation 2.10, a circulating magnetic field creates a time-varying (perpendicular) electric field. In terms of dynamic electromagnetic behavior the pair of equations are coupled together by the spatial and temporal expression of the electric and magnetic fields.

If one considers the three-dimensional Cartesian coordinate system (x,y,z), decomposing the curl operator in Equations 2.7 and 2.8 yields:

$$\frac{\partial H_x}{\partial t} = \frac{1}{\mu} \cdot \left(\frac{\partial E_y}{\partial z} - \frac{\partial E_z}{\partial y} - \rho' H_x \right) \tag{2.11a}$$

$$\frac{\partial H_y}{\partial t} = \frac{1}{\mu} \cdot \left(\frac{\partial E_z}{\partial x} - \frac{\partial E_x}{\partial z} - \rho' H_y \right) \tag{2.11b}$$

$$\frac{\partial H_z}{\partial t} = \frac{1}{\mu} \cdot \left(\frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x} - \rho' H_z \right) \tag{2.11c}$$

$$\frac{\partial E_x}{\partial t} = \frac{1}{\varepsilon} \cdot \left(\frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} - \sigma E_x \right) \tag{2.12a}$$

$$\frac{\partial E_y}{\partial t} = \frac{1}{\varepsilon} \cdot \left(\frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} - \sigma E_y \right) \tag{2.12b}$$

$$\frac{\partial E_z}{\partial t} = \frac{1}{\varepsilon} \cdot \left(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} - \sigma E_z \right) \tag{2.12c}$$

It should be noted that Equation 2.8 is now generalized to include magnetic losses as well, which appear as $\rho' H$ in the magnetic field expressions.

This system of six coupled equations forms the foundation of the FDTD algorithm for modelling the interaction of electromagnetic waves with general three-dimensional objects. These equations are not suitable for computer implementation, because they contain continuous-time and continuous-space variables. Kane Yee's method provides an efficient way to transform this system of partial difference equations into a second-order accurate system of difference equations.

2.2.2 Yee's Method

Most of the following material in this section is adapted from Taflove [9] and Yee's original paper [1].

Yee used centered finite-difference operators to represent both the spatial and temporal partial derivatives in Equations 2.11a to 2.12c. These centered differences are easy to implement on a computer. Furthermore, they are second-order accurate.

The FDTD Update Equations

Yee's update equations are the discretized versions of Equations 2.11a to 2.12c. The notation used is: (i,j,k) represents a point in the lattice really $(i\Delta x, j\Delta y, k\Delta z)$ in space, and n represents a time step or $t = n\Delta t$. The FDTD update equations are:

For the magnetic fields:

$$H_x\Big|_{i,j,k}^{n+1/2} = D_{a,H_x}\Big|_{i,j,k} \cdot H_x\Big|_{i,j,k}^{n-1/2} + D_{b,H_x}\Big|_{i,j,k} \cdot \left[-\frac{E_y\Big|_{i,j,k+1/2}^{n} - E_y\Big|_{i,j,k-1/2}^{n}}{\Delta z} \; \frac{E_z\Big|_{i,j+1/2,k}^{n} - E_z\Big|_{i,j-1/2,k}^{n}}{\Delta y} \right] \tag{2.13a}$$

$$H_y\Big|_{i,j,k}^{n+1/2} = D_{a,H_y}\Big|_{i,j,k} \cdot H_y\Big|_{i,j,k}^{n-1/2} + D_{b,H_y}\Big|_{i,j,k} \cdot \left[-\frac{E_z\Big|_{i+1/2,j,k}^{n} - E_z\Big|_{i-1/2,j,k}^{n}}{\Delta x} \; \frac{E_x\Big|_{i,j,k+1/2}^{n} - E_x\Big|_{i,j,k-1/2}^{n}}{\Delta z} \right] \tag{2.13b}$$

$$H_z\Big|_{i,j,k}^{n+1/2} = D_{a,H_z}\Big|_{i,j,k} \cdot H_z\Big|_{i,j,k}^{n-1/2} + D_{b,H_z}\Big|_{i,j,k} \cdot \left[-\frac{E_x\Big|_{i,j+1/2,k}^{n} - E_x\Big|_{i,j-1/2,k}^{n}}{\Delta x} \; \frac{E_y\Big|_{i+1/2,j,k}^{n} - E_y\Big|_{i-1/2,j,k}^{n}}{\Delta x} \right] \tag{2.13c}$$

For the electric fields:

$$E_x\Big|_{i,j,k}^{n+1} = C_{a,E_x}\Big|_{i,j,k} \cdot E_x\Big|_{i,j,k}^{n} + C_{b,E_x}\Big|_{i,j,k} \cdot \left[\frac{H_z\Big|_{i,j+1/2,k}^{n} - H_z\Big|_{i,j-1/2,k}^{n}}{\Delta y} - \frac{H_y\Big|_{i,j,k+1/2}^{n} - H_y\Big|_{i,j,k-1/2}^{n}}{\Delta z} \right] \tag{2.14a}$$

$$E_y\Big|_{i,j,k}^{n+1} = C_{a,E_y}\Big|_{i,j,k} \cdot E_y\Big|_{i,j,k}^{n} + C_{b,E_y}\Big|_{i,j,k} \cdot \left[\frac{H_x\Big|_{i,j,k+1/2}^{n} - H_x\Big|_{i,j,k-1/2}^{n}}{\Delta z} - \frac{H_z\Big|_{i+1/2,j,k}^{n} - H_z\Big|_{i-1/2,j,k}^{n}}{\Delta x} \right] \tag{2.14b}$$

$$E_z\Big|_{i,j,k}^{n+1} = C_{a,E_z}\Big|_{i,j,k} \cdot E_z\Big|_{i,j,k}^{n} + C_{b,E_z}\Big|_{i,j,k} \cdot \left[\frac{H_y\Big|_{i+1/2,j,k}^{n} - H_y\Big|_{i-1/2,j,k}^{n}}{\Delta x} - \frac{H_x\Big|_{i,j+1/2,k}^{n} - H_x\Big|_{i,j-1/2,k}^{n}}{\Delta y} \right] \tag{2.14c}$$

To simplify the notation further, the terms C_a, C_b, D_a and D_b were introduced to represent the field vector coefficients (and material properties) at each point in space (i,j,k). For stationary media, whose properties do not vary with time, these coefficients can be pre-computed. These terms are as follows:

$$C_a\Big|_{i,j,k} = \left(\frac{1 - \dfrac{\sigma_{i,j,k}\Delta t}{2\varepsilon_{i,j,k}}}{1 + \dfrac{\sigma_{i,j,k}\Delta t}{2\varepsilon_{i,j,k}}} \right) \qquad C_b\Big|_{i,j,k} = \left(\frac{\dfrac{\Delta t}{\varepsilon_{i,j,k}}}{1 + \dfrac{\sigma_{i,j,k}\Delta t}{2\varepsilon_{i,j,k}}} \right) \tag{2.15}$$

$$D_a\,|_{i,j,k} = \left(\frac{1 - \dfrac{\rho'_{i,j,k}\Delta t}{2\mu_{i,j,k}}}{1 + \dfrac{\rho'_{i,j,k}}{2\mu_{i,j,k}}} \right) \qquad D_b\,|_{i,j,k} = \left(\frac{\dfrac{\Delta t}{\varepsilon_{i,j,k}}}{1 + \dfrac{\sigma_{i,j,k}\Delta t}{2\varepsilon_{i,j,k}}} \right) \tag{2.16}$$

To compute the new magnetic field value ($H_x\,|^{n+1/2}$) at a given point, in space and time, Equation 2.13a uses only the current fields ($E_y\,|^n$ and $E_z\,|^n$) and previous field ($H_x\,|^{n-1/2}$), all three of which are stored in computer memory.

Equations 2.13a to 2.14c comprise the simplest form of the FDTD update equations which are implemented as the core engine of the FDTD algorithm. For each time step, Δt, all of the magnetic fields are calculated followed by all of the electric fields, for the entire three-dimensional volume of Yee cubes.

The Yee Cube

The manner in which Equation 2.13a calculates the updated value of the field H_x is intuitively explained by the Yee cube, which is a graphical representation of Yee's discrete electric and magnetic fields in discrete space. The Yee cube is shown in Figure 2.2.

In this case, the new value of H_x, centered at (i,j,k), is determined by the weighted contour sum of the surrounding electric fields ($E_z\,\big|^n_{i,j+1/2,k}$, $-E_y\,\big|^n_{i,j,k+1/2}$, $-E_z\,\big|^n_{i,j-1/2,k}$, $E_y\,\big|^n_{i,j,k-1/2}$). For two of the fields, the negative signs are used to maintain sign convention.

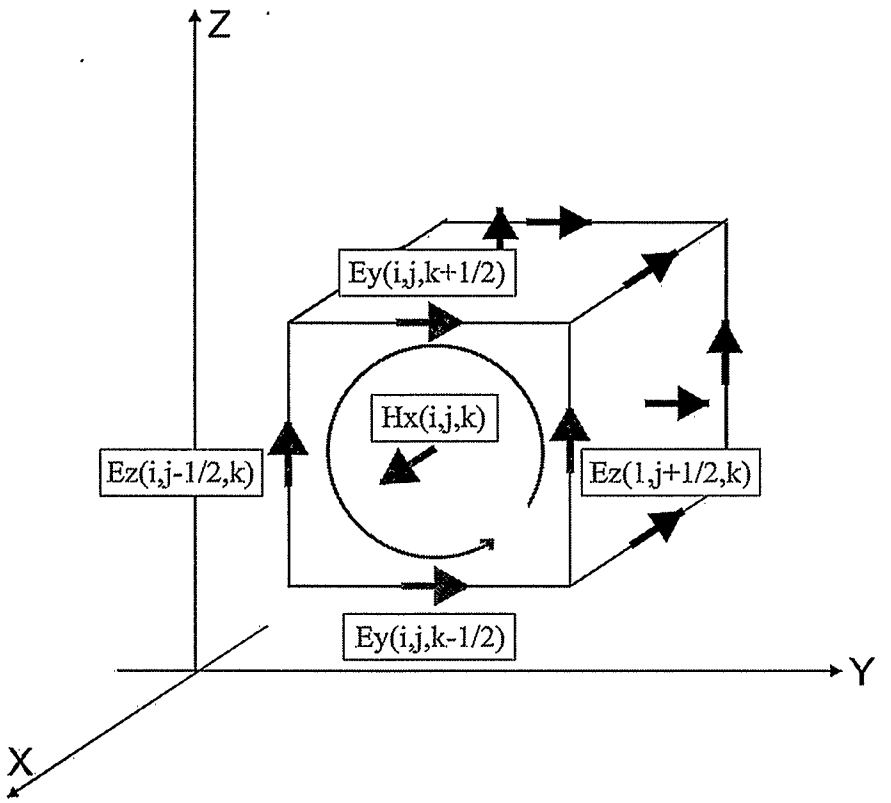


Figure 2.2: The Yee Cube

Relationship Between Two- and Three-Dimensional Computations

There is a relationship between the two- and three-dimensional implementations of the FDTD algorithm; the relationship is not immediately obvious but very useful for the purpose of this research. The three-dimensional field update equations are simply the computation of a series of two-dimensional planes. This is further illustrated by the Yee cube of Figure 2.2. All of the H_x field components will lie in the same plane for a given value of $i\Delta x$. The important observation is that once a hardware implementation of the two-dimensional FDTD update equation is solved, the three-dimensional case is solved as well. In other words, the FDTD update equations only require the field information from two dimensions, even for the three-dimensional case.

Summary of Yee's Method

The use of centered-difference equations has a few important effects on Maxwell's equations:

- The three-dimensional volume of interest is now divided into a mesh of discrete Yee cubes. Each cube has dimensions Δx, Δy and Δz. The example in Figure 2.3 is simplified for visual understanding. A much finer grid is needed in order to generate accurate results and represent the model's materials (a complex cellphone and human head, in this case) with sufficient resolution.

- The continuous electric and magnetic fields become discrete fields centered on the edges or faces of the Yee cubes, respectively.

- The continuous function of time is discretized into time steps of Δt.

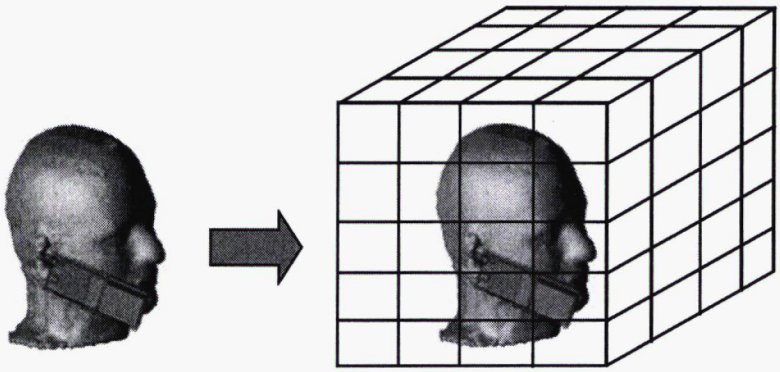


Figure 2.3: Model of a Three-Dimensional "Volume of Interest"

- At each time step, all of the magnetic fields are updated, followed by all of the electric fields, using Equations 2.13a to 2.14c. Over an entire simulation, this models the time-domain behavior of the electromagnetic fields in the "region of interest".

The preceding features are a result of the mathematics described in the previous sections, and in more detail in Appendix A and [1, 9]. The above effects/properties make the algorithm very suitable for a software implementation.

2.3 Properties of the FDTD Algorithm

Several important properties of the FDTD algorithm are described in this section. The discussion includes: the Courant condition, the runtime of FDTD, and the properties that are exploited for acceleration.

2.3.1 Courant Condition

The Courant condition [8] places an upper bound on the value of Δt in order for the FDTD simulation to remain stable. It ensures that fields do not propagate faster than the speed of light within the simulation.

The Courant condition for three-dimensional FDTD is:

$$\Delta t \leq \frac{S_{factor}}{\sqrt{\left(\frac{1}{\Delta x}\right)^2 + \left(\frac{1}{\Delta y}\right)^2 + \left(\frac{1}{\Delta z}\right)^2} \cdot c} \tag{2.17}$$

Where c = speed of light $[m/s]$

S_{factor} = stability factor, [0.0 to 1.0]

Δt is maximized when the left hand side of Equation 2.17 is equal to the right-hand side and $S_{factor} = 1.0$. This represents optimum FDTD performance (least numerical dispersion at higher frequencies) at the expense of operating at or close to instability. For practical applications, the "stability factor" is introduced to communicate the relationship between Δt and its maximum value. For this research, the stability factor is 0.95 unless otherwise noted. For one and two dimensions, which are also considered in this research, the denominator of the Courant condition is expressed with only a single dimension or two-dimensions respectively.

Finite-Precision Consideration

The sampling interval, Δt, and the corresponding stability factor are usually only considered or adjusted when they cause instability in the simulation. It will be shown

in Chapter 4 that finite-precision effects can contribute to an effective value of Δt, different than the calculated value, so the stability factor may need to be re-visited.

2.3.2 Run-time of the FDTD Algorithm

FDTD simulations can take a long time. In fact, simulations can run for several hours to several days on multi-processor supercomputers. In some cases, simulations are also run multiple times because of modelling difficulties or adjustments made to the model after previous runs. Reducing the runtime of this method would greatly increase the productivity of FDTD users in both academia and industry; this is the primary reason for research into hardware acceleration.

There are three factors which directly contribute to the runtime of a given simulation model:

1. To minimize numerical dispersion [9], the following rule of thumb is used: The mesh size is chosen such that the shortest wavelength of interest is over-sampled 10 to 20 times. In other words the width of the cubes (Δx or Δy or Δz) is 1/10 or 1/20 of a wavelength. While satisfying the Courant condition, this also yields a sample interval Δt which provides enough resolution at the highest frequency of interest. In fact, this criterion usually yields a sampling interval much shorter than required for the Nyquist rate. Thus the simulation is also highly over-sampled in time.

2. Some physical models may require extremely fine spatial resolution, because of small physical details, sharp edges or rapidly varying material properties. Two examples are a human body model and a photonic structure. The University

of Victoria has developed a model of the human body that represents material properties in 3mm cubes for the body and 1mm cubes for the head. Photonic structures may have very detailed structures with many edges and fins. Both of these examples produce simulation models with a very fine mesh and a correspondingly large number of cells.

3. As another rule of thumb, the simulation is run for three or four periods at the lowest frequency of interest[4]. A sufficiently long simulation time allows the excitation to propagate throughout the structure and for the time domain observations to converge or reach steady-state.

All of these factors lead to a large simulation model (large number of cells) or a simulation that needs to run for a large number of time steps or both. From results to be presented in Chapter 3, a simulation with 100x100x100 cells and 10,000 time steps will take just over one hour to run. This is considered to be a "medium-sized" simulation. For comparison, simulations by Fear [23] for a microwave detection method for breast cancer take more than 24 hours on multi-processor supercomputers. These simulations, with 250x250x170 cells for 6,000 time steps, are roughly an order of magnitude larger than the 100x100x100 cell simulation described previously.

2.3.3 Properties of the FDTD Algorithm that are Exploited for Acceleration

There are a number of properties, which make this algorithm well-suited for a hardware implementation.

[4] Alternatively, minimum simulation time $= 4 \times \frac{1}{f_{lowest}}$ where f_{lowest} is the lowest frequency of interest in the simulation.

1. Nearest Neighbor Data Dependence – A field update uses fields from adjacent cells only. It is theoretically possible to implement every single cube in the simulation as a separate piece of computational hardware with local connectivity. This widespread, low-level parallelism would yield the desired speed increase. In terms of a three-dimensional FDTD hardware implementation, only adjacent cells would need to be connected to each other. Local connectivity would not exist at the routing level on an FPGA because a three-dimensional volume of computations is mapped to a two-dimensional hardware array.

2. Leapfrog Time-Domain Calculations – Each magnetic field update is only dependent on its previous value and the value of four stored electric fields. Likewise, the electric field update is only dependent on itself and four stored magnetic fields. The key benefit is that all the magnetic fields or all electric fields could be computed in parallel. In other words, the electric and magnetic field calculations are "alternately clocked", which means the electric/magnetic field is stored while the magnetic/electric field is calculated.

3. Calculation In-Place – Each set of $\vec{E}$ or $\vec{H}$ field update values can be calculated in place and it is not necessary to store intermediate field values. Each update equation can also be implemented as a multiply and accumulate structure.

4. Six Similar Update Equations – Each field calculation, electric or magnetic, in any dimension has the same structure. This is advantageous for very large scale integration (VLSI) and FPGA platforms because the repetitive structure is easy to duplicate or reuse.

5. Very Regular Macro-Structure – Except for the multiplier coefficients, which determine local material properties, the computational structure is identical from simulation to simulation for a given volume. Thus, it is possible to reuse pre-compiled FPGA cores. Any modifications to the hardware simulation parameters could also be performed during runtime, using runtime reconfiguration.

6. Constant Coefficients – Material properties (omitting non-stationary media) and the sample rate remain constant throughout a simulation. Complex (non-linear, non-stationary or non-isotropic) materials require different update equations and are traditionally handled separately from the bulk of normal materials. Thus, for most typical simulations, coefficients remain fixed for a given field calculation for the entire simulation. This is also well-suited to an FPGA platform. Fixed coefficient multipliers can be configured during compile time or a fixed design reconfigured at run time. Custom fixed-coefficient multipliers also require less hardware than their generalized counterparts.

2.4 Inductor-Capacitor (LC) Implementation of FDTD

Gwarek [24, 25] observed that a two-dimensional FDTD structure can be represented as a network of inductors and capacitors (Figure 2.4). The capacitors represent electric field storage, the inductors represent the current and consequently the magnetic field storage. There are direct relationships between the inductor/capacitor values and the electromagnetic properties of the FDTD mesh.

This relationship between FDTD and an inductor-capacitor network is a useful

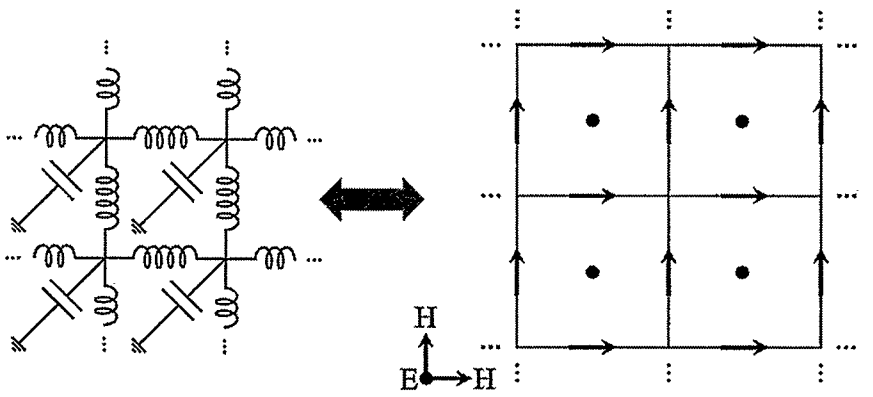


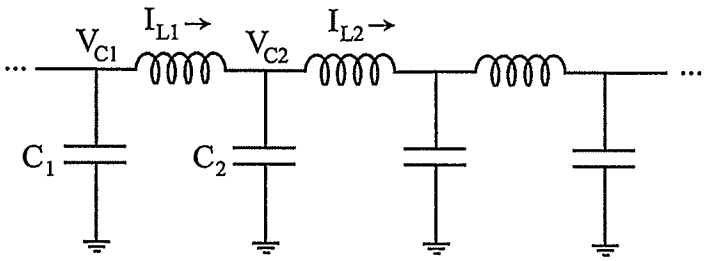
Figure 2.4: FDTD Mesh (*right*) Represented as an Inductor-Capacitor Mesh (*left*)

starting point for the hardware implementation. It provides an indirect path for implementing the FDTD algorithm. The LC network is very similar to an analog filter network, albeit somewhat unusual because of the two-dimensional nature. Filter designers have developed several techniques to both analyze and implement these networks in the digital domain. The following sections describe how the inductor-capacitor relationship is used to create one-dimensional and two-dimensional FDTD computational cells. The cells are constructed in Chapter 4 and interconnected to form FDTD simulations in hardware.

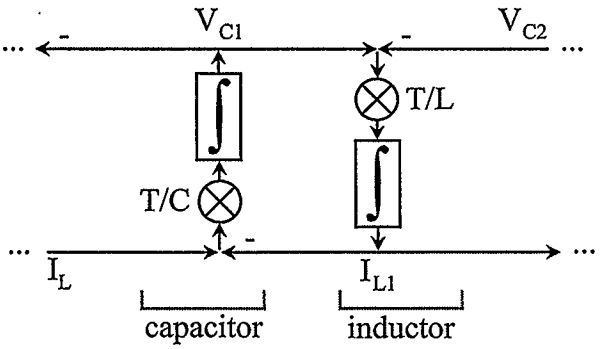
The one-dimensional case of FDTD is discussed and developed first, both for simplicity and the ease of graphical representation. The two-dimensional case is easily extended from the one-dimensional developments. With the solution of the two-dimensional computation, the following methodology can also be used to represent three-dimensional FDTD.

2.4.1 One Dimensional FDTD Cell

The one-dimensional FDTD case is as a special case of Figure 2.4, and is further explained in Figure 2.5.



(a) One-Dimensional Cells



(b) Voltage/Current Signal Flow Graph, Single Cell

Figure 2.5: One Dimensional FDTD Cell: Alternate Representation

To generate the signal flow graph representation of Figure 2.5(b), the capacitor is replaced by a current-integrator likewise the inductor is replaced by a voltage-integrator. Voltages are represented by the signals along the top of the graph and currents by the signals along the bottom.

Adapting Bruton's work [26], the integrators are replaced by lossless discrete integrators (LDI's) of the form depicted in Figure 2.6.

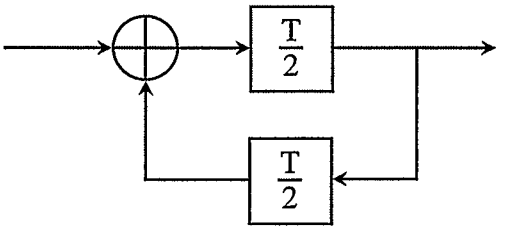


Figure 2.6: Lossless Discrete Integrator

With manipulation of the delays, as for the LDI structure, the circuit in Figure 2.7 is produced.

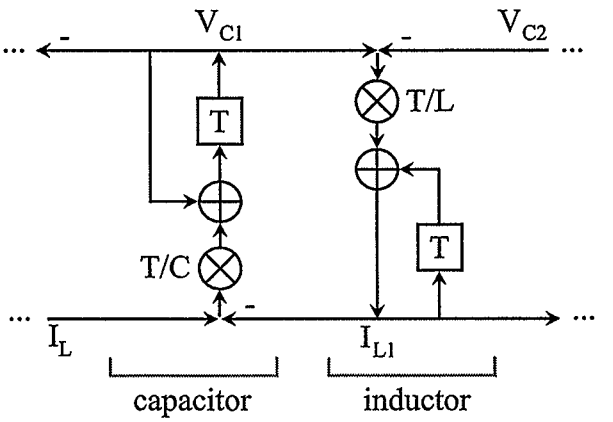


Figure 2.7: LDI Form of the One-Dimensional FDTD Computation

It should be noted that the value of current, I, is really for time k + 1. The value of voltage is at k, where k is one-half of a simulation time step. This is the same as leapfrog lossless discrete integrator (LDI) ladder structure and the classical FDTD algorithm.

2.4.2 Two Dimensional FDTD Cell

The two-dimensional FDTD mesh, using the LC representation, is just rows of the one-dimensional cells connected by inductors. The interpretation is that each inductor couples the capacitor to its adjacent cells, now in two dimensions. Figure 2.8 depicts this two-dimensional FDTD cell. In order to create a two-dimensional simulation mesh ($M \times N$ cells), this structure would be repeated in an $M \times N$ array.

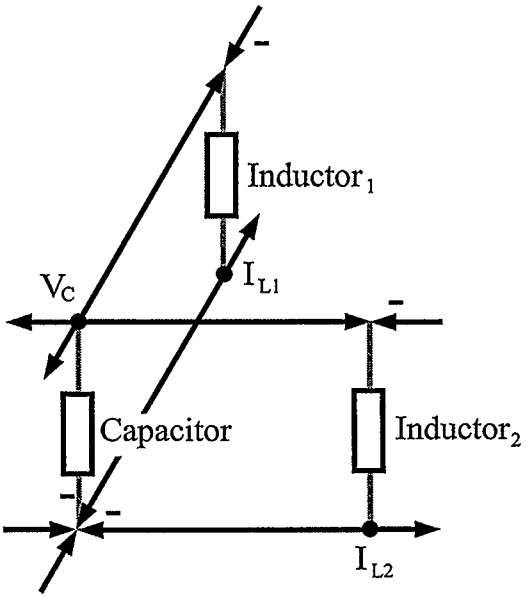


Figure 2.8: Two Dimensional FDTD Cell: Signal Flow Graph

2.4.3 Determination of the Inductor-Capacitor Coefficients

This section describes how the inductor and capacitor coefficient values are determined for each node in the FDTD hardware simulation. Following this, a method for scaling the coefficients with respect to each other is described and justified. Finally, the process for quantizing the infinite-precision coefficients into finite-precision

values is discussed.

The coefficient values for the capacitors and inductors in a particular cell represent the local material properties. For two-dimensional FDTD there is a dimension d introduced, which is the thickness of the structure. The only affect that d has is to scale the instantaneous voltage and current magnitudes. The V/I relationship, however, will remain the same.

The value of the capacitance is given by:

$$C = C_s \cdot a \tag{2.18}$$

$$C_s = \frac{\varepsilon}{d}$$

$$a = \Delta x \Delta y$$

Where C_s = capacitance per unit area $[F/m^2]$

a = area of the cell $[m^2]$

d = thickness of the cell $[m]$

$\Delta x, \Delta y$ = dimensions of the cell $[m]$

The value of the inductance is given by:

$$L = L_s \tag{2.19}$$

$$L_s = \mu \cdot d$$

Where L_s = inductance per unit length $[H/m]$

d = thickness of the cell $[m]$

Equations 2.18 and 2.19 can be extended for cell faces in any dimension (x,y,z). Given the mesh size $(\Delta x, \Delta y, \Delta z)$, ε and μ the inductor and capacitor values can be pre-determined for every cell in the FDTD mesh.

The multiplier coefficients are then determined by:

$$X_{capacitor} = \frac{\Delta t}{C} \tag{2.20}$$

$$X_{inductor} = \frac{\Delta t}{L} \tag{2.21}$$

Impedance Scaling

Impedance scaling describes the method of adjusting the inductor and capacitor coefficient values, with respect to each other. This is mathematically described as:

$$C_{new} = C \cdot S_{impedance} \tag{2.22}$$

$$L_{new} = L \cdot \frac{1}{S_{impedance}} \tag{2.23}$$

Where $S_{impedance}$ = scaling factor

This in turn leads to new values for $X_{capacitor}$ and $X_{inductor}$. The net effect is that the voltages and currents in the structure will be oppositely scaled by $S_{impedance}$. However, the V/I relationship is maintained. This is similar to the effect of the dimension d in the previous section.

This scaling is used to achieve the following:

1. The coefficients can be scaled to values within a given hardware multiplier's range.

2. The voltage and current amplitudes in the network can be adjusted to have similar amplitude ranges.

3. Coefficients can be scaled to a convenient finite-precision value. Generally, using fixed-precision integers, this will allow some coefficients to be represented exactly while others will have some quantization error.

For this research, there is only one inductor coefficient and one capacitor coefficient, which represent free space material properties throughout the entire mesh. In this case, one coefficient is normalized to 0.5. This fixes the other coefficient at some impedance-scaled value. When the coefficients are quantized, the coefficient of 0.5 is represented exactly while some error is introduced for the other.

For future designs and multiple coefficients, it would be possible to search the coefficient space and determine the optimum impedance scaling factor that would lead to the smallest quantization error.

Coefficient Quantization

For this work, the computations use fixed-precision arithmetic. Thus, the infinite-precision coefficients are truncated into fixed-precision values for use in the hardware.

The quantized coefficient value is determined by:

$$X_{multiplier} = round\left(X_{ideal} * 2^N\right) \tag{2.24}$$

$$X_{quant} = \frac{X_{multiplier}}{2^N} \tag{2.25}$$

Where X_{ideal} = infinite-precision multiplier coefficient

N = fixed-precision coefficient width

$X_{multiplier}$ = the integer coefficient applied to the hardware multiplier

X_{quant} = the finite-precision multiplier coefficient

In this instance, the rounding function chooses the integer result which minimizes the difference between X_{quant} and X_{ideal}. Thus, this function is entirely dependent on the coefficient value and the number of bits used.

This has an important impact on the results of the FDTD simulation. The quantization of the coefficients produces some error either in the value of Δt, C or L in Equations 2.20 and 2.21. This error will produce changes in the simulation results, when compared to the infinite-precision case. It is important to ensure that the quantization errors do not impact the accuracy of the FDTD simulation results. This can be controlled by adjusting the bit-width of the coefficients.

2.4.4 Useful Properties of the LC Representation

The lossless discrete integrator (LDI) digital ladder filter is known to exhibit low-valued transfer function sensitivity to the filter coefficient values [26]. This low sensitivity property allows high quality LDI lowpass digital filters to be constructed using very few bits to represent the filter coefficients and is also related to the very

desirable low noise characteristics of the filter [27]. As the structure of the LDI lowpass digital ladder filter and the one-dimensional FDTD cells can be the same, it is anticipated that this low sensitivity property can translate directly into hardware savings and low noise characteristics for the FDTD implementations.

Linear stability (using ideal infinite-precision arithmetic) of the FDTD structure is easily guaranteed. Stability of the FDTD structure when implemented using non-linear finite precision arithmetic requires further study. Initial evidence from simulations and implementations using finite-precision arithmetic do not produce instability. As long as the Courant condition is maintained, even after coefficient quantization, the FDTD simulation should remain stable. In order to encourage stability, small amounts of loss could be added to the FDTD structure, using the coefficients C_a and D_a of Equations 2.15 and 2.16. It should be noted that the primary concern might not be stability, rather, the insufficient quantization of field update coefficients may lead to unacceptable errors in the representation of the material properties and the simulation results.

Chapter 3

Computer Technology and Software Benchmarks

"Consistently separating words by spaces became a general custom about the tenth century A.D., and lasted until about 1957, when FORTRAN abandoned the practice." - Sun FORTRAN Reference Manual

This chapter describes three things: (i) the reference computer, (ii) the computer technology, that affects both the software FDTD implementation(s) and the envisioned hardware accelerator, and (iii) measurements (computation speed and memory bandwidth) of software FDTD implementations. Finally, this information is used to predict the memory bandwidth and computation speed that is required to achieve an order of magnitude acceleration.

The first section describes the reference computer for all software simulations. The computer's specifications provide a baseline for any achieved acceleration.

The second section describes the expected data bandwidth of three main computer technologies: SDRAM memory, the PCI hardware bus and the AGP hardware bus. This information is used mainly for two things: to provide an expected/maximum memory performance for the FDTD algorithm on the reference computer and to provide hardware bus information for the future hardware accelerator.

The third section presents the methodology used and benchmarks obtained (computation speed and memory bandwidth) for two software FDTD implementations on the reference computer. These benchmarks are used to compare any achieved accel-

eration to a baseline computer/software implementation.

3.1 Reference Computer

Although this work is primarily focused on a hardware implementation, in order to make valid comparisons between the hardware and software FDTD algorithms, software measurements are needed as well.

The computer which runs all of the software simulations is a PC type machine with the following characteristics:

- AMD (Advanced Micro Devices) Athlon 850MHz Processor
- 64KB of L1 data cache (64 bytes/line), 64KB of L1 instruction cache (64 bytes/line), 512KB of L2 cache(64 bytes/line)
- 2×256MB PC133 Crucial Micron (168-pin DIMM, CAS latency $= 2$) Memory
- Red Hat Linux v7.3 Operating System
- PCI (Peripheral Connection Interface) bus - 33MHz at 32-bits
- AGP (Accelerated Graphics Port) - 1X,2X support

The technologies for this computer were "cutting-edge" at the start of this project. When a hardware accelerator prototype is completed, it is suggested that the runtime of the FDTD software should be compared with and without acceleration enabled.

Over time, computer processors will increase in speed, due to advances in manufacturing. Those same advancements should also improve the speed of both FPGA's

and custom ASIC's. In fact, the Xilinx FPGA device used in future sections is older technology. Thus, newer products will provide even better performance than what is reported in the text.

Currently, the software benchmark collection is scripted, relying on very little intervention from the user. Measurements are typically collected and then averaged from one hundred instances (unless otherwise noted) of a particular FDTD simulation run.

With respect to the "reference computer", it is assumed that the processor speed and memory bandwidth will have the most impact on the runtime of the software algorithm. As long as there is sufficient physical memory to store the entire FDTD simulation, any additional RAM should not have much impact on the runtime. If the computation speed is limiting the performance, the processor speed will have the most impact on runtime until the memory bandwidth is fully utilized. On the other hand, if the memory bandwidth is fully utilized, then moving to a higher bandwidth memory technology will have the most impact on performance until the CPU resources are fully utilized.

3.1.1 Potential Interfaces for the Future Hardware Accelerator

The last two items in the "reference computer" list, the PCI and AGP details, are not important to the performance of the software. Rather, these two hardware interfaces are considered for the future hardware accelerator card. Thus, these specifications are listed to illustrate the available technology in this personal computer for the hardware accelerator interface/connection.

As a side note, initial research considered the memory bus as another hardware

interface for the accelerator. This avenue has not been fully pursued for two reasons:

First, there appears to be only one hardware-coupled memory product available in the market. SmartDIMM [28] provides for a PC100 memory card coupled with a Xilinx FPGA. It is only compatible with an Intel 440BX motherboard chipset. These requirements seem too restrictive for an accelerator that would compete with current technology. Thus, this would require the design of a new memory module that is compliant with the chosen memory standard and coupled to some digital hardware. This option was not pursued due to the perceived development time and the lack of a verified hardware FDTD implementation.

Second, the use of interleaved memory banks is an emerging trend in the PC industry. This means that a memory-based hardware accelerator would not be available on the memory bus all of the time and presumably this behavior would not be controlled by the user/programmer. This would increase the complexity of the acceleration solution.

It should be noted that there is another technology called Computational RAM (CRAM). While not considered in this work, it may provide another option for FDTD acceleration.

3.2 Data Bandwidth of Memory and Hardware Buses

This section provides information about the data bandwidth that can be expected from various memory and hardware bus technologies. The performance of memory is discussed in the following section, followed by the performance of the PCI (Peripheral Connections Interface) and AGP (Accelerated Graphics Port) hardware buses.

3.2.1 Memory Bandwidth: Theoretical vs. Sustainable

This section investigates the bandwidth of various memory technologies. This information is useful for a discussion of the software runtime performance and for future acceleration work. From Chapter 5 it is highly probable that the future hardware accelerator will be coupled to several megabytes of dynamic RAM. Table 3.1 depicts the theoretical, peak data bandwidths for several current memory technologies.

Table 3.1: Maximum (Read) Bandwidth for Various SDRAM Memory Technologies

Memory Technology	Peak Memory (Read) Bandwidth (MB/s)†
PC100 (64-bits × 100MHz)	762.9
PC133 (64-bits × 133MHz)	1014.7
PC2100 DDR‡ (64-bits × 266MHz)	2029.4
PC2700 DDR‡ (64-bits × 333MHz)	2540.6

†M = 1024^2, B = bytes
‡Double-Date Rate (DDR)

In general, dynamic random access memory (DRAM) is optimized for faster read operations than write operations. FDTD requires far more memory reads than writes, so this DRAM optimization is beneficial. This is discussed in greater detail in Section 3.3.

All of the above memory technologies access the data in bursts. Following a cache miss, a burst from memory would fill one or more cache lines in the CPU. In newer technologies, these bursts can be programmed to be 1, 2, 4, 8 values and more at

once. For the burst mode of 4, rather than one memory value being returned after a read request, 4 values are returned.

There are a number of factors which are not considered when the peak memory bandwidths are reported. There is a large latency for the first memory access, which is not accounted for in Table 3.1. The table value reports the rate at which the burst of the next three values (192-bits or 3×64-bits) is received, basically at one value per clock cycle (every 10 ns). For PC100, the first access latency is 50 ns [29]. This means that to read 4 values, it takes 80ns. The maximum read bandwidth, for PC100 memory with bursts of 4 values, is:

$$BW_{PC100} = \frac{256bits}{80 \times 10^{-}9s} \times \frac{MB}{8bits \times 1024^2} = 381.5MB/s \tag{3.1}$$

Because of the large latency for the first access, the sustainable memory bandwidth is nearly half of the peak reported. From [30], the initial latency for PC133 memory is 44 ns. Using a calculation similar to Equation 3.1 the sustainable memory bandwidth for PC133 with bursts of 4 values is 458.5 MB/s, once again less than half of the peak.

Memory manufacturers do not willingly publish this information and make it very difficult to obtain, because of the large discrepancy between sustainable and peak memory bandwidths. No suitable timing specifications could be found for Double Data Rate SDRAM. However, it is expected that sustainable bandwidths will be significantly lower than the reported peak.

In Section 3.3, these peak and sustained bandwidths are compared to the calculated memory bandwidth used by a software implementation of the six FDTD

update equations. The goal is to evaluate whether the computer's CPU or memory is the limiting factor in the FDTD computations.

3.2.2 PCI and AGP Bandwidth

This section applies to the next generation hardware accelerator. The possible bandwidths for the available PCI and AGP technologies are described. One of these two hardware buses will be selected for connecting the future hardware accelerator to the host computer.

For the current PCI standards, there are several different configurations supported by different motherboard chipsets and manufacturers. Traditionally, PCI has operated at 33MHz but can now operate at 66Mhz. In the same vein, traditionally, the PCI bus was 32-bits wide. Newer motherboards may also support a 64-bit PCI bus. Table 3.2 describes the bandwidths that the various configurations can achieve.

Table 3.2: Maximum Data Bandwidth for Various PCI Standards

	Bus Width	
Bus Speed	32-bits	64-bits
33 MHz	125.9 MB/s†	251.8 MB/s†
66 MHz	251.8 MB/s†	503.5 MB/s†

†M = 1024^2, B = bytes

Recently, manufacturers are also supporting a version of PCI called PCI-X. This is a 64-bit wide bus operating at 133 MHz. This results in a maximum bandwidth of 1014.7 MB/s.

One of the key assumptions in the above data is that the PCI bus is monopolized

by a single peripheral. Otherwise, the reported maximum bandwidths must be shared between many peripherals. Another observation is that the same bus must be used to transact both address and data information. Unless some form of bursting is used, half of the data bandwidth will be used for addressing.

AGP is designed specifically to provide higher data rates, between the CPU and a video card, than the PCI bus can achieve. Table 3.3 describes the bandwidths that the various AGP standards offer [31].

Table 3.3: Maximum Data Bandwidth for Various AGP Standards

AGP Standard	Peak Bandwidth† (MB/s)
1X	254.3
2X	508.6
4X	1017
8X	2034

†M = 1024^2, B = bytes

The AGP specification also provides for an additional 8 data lines for sideband addressing. This is explained in greater detail by [31]. Unlike PCI, addresses can be specified using these 8 lines while data transactions are still taking place on the main bus. Thus, AGP performance is expected to further improve if this modification is used.

3.2.3 Impact of Memory and Hardware Bus Technology on the Next Generation Hardware Accelerator

Table 3.1 provides the peak memory bandwidths offered by various technologies. It is then shown that sustainable bandwidths are only half of the peak values. Tables 3.2 and 3.3 provide a reference for the maximum data bandwidth that can be achieved between a PCI- or AGP-based hardware accelerator. Again, these rates will be limited by overhead and other factors.

The choice of technology used to connect the hardware accelerator to the host computer will also fix the available data bandwidth between the two devices. This will ultimately limit the amount of data that can be exchanged between the accelerator and the host computer. If the bandwidth is low or the amount of data exchanged is too high, the accelerator's performance will be limited.

The envisioned hardware accelerator will use some form of DRAM memory. Thus, the chosen memory technology will also determine the maximum sustainable memory bandwidth. The goal is to operate the memory bank(s) at maximum throughput. Nevertheless, the choice of technology will ultimately limit the available input bandwidth to the accelerator.

The next section provides performance benchmarks (run-time, computation speed and memory bandwidth) for several software FDTD simulations.

3.3 Benchmarks of FDTD Update Equations

The goal of this investigation is two-fold: (i) to determine the computation speed for a single field update, given that the six update equations are identical in structure, as

discussed in Chapter 2. And (ii) to determine the memory bandwidth utilized by the algorithm. Ultimately, this information is used to predict the required computation speed and memory bandwidth to accelerate the FDTD algorithm by a factor of 10 or more.

Later, in Section 3.4, Okoniewski's research FDTD code [32] is analyzed on the reference computer. However, for Section 3.3 the code is a simple FDTD program written in C++. For this analysis, the simulation model or results are not considered. The approach of this investigation is from a purely algorithmic perspective.

3.3.1 The Algorithmic Perspective

The goal is to measure the aggregate of the memory and arithmetic operations to determine the performance of the host computer/CPU.

For the purpose of this experiment, each update equation can be written as:

$$F_{new} = C_1 \cdot F_{old} + C_2 \cdot [C_3 (F_2 - F_1) + C_4 (F_4 - F_3)] \tag{3.2}$$

Where F_{old} = field to be updated (to F_{new})

C_1, C_2, C_3, C_4 = field and material property coefficients

F_1, F_2, F_3, F_4 = adjacent, circulating fields

This equation format is common to all six FDTD update equations. Thus, the FDTD algorithm is really the computation of Equation 3.2 for a three-dimensional array of six field components iterated for a given number of time steps.

Ignoring cache, the data requirements for Equation 3.2 is nine memory loads and

one memory store. This represents a somewhat pessimistic view of the implementation, because certain coefficients could be combined or even arranged as an array of constants. However, for the worst case scenario, the equation requires nine floating point values and writes back one floating-point value. If the three-dimensional array of fields is sufficiently large, the CPU will experience cache misses. The host computer will fetch the data from memory to cache on a regular basis as the computations proceed. Thus, any timing or memory bandwidth measurements will be from memory to the CPU because there is too much data to contain in the cache. Obviously, the code is structured to make the most effective use of cache lines[1]. However, the cache lines are still refilled on a regular basis, incurring the cost of a memory fetch each time.

An important observation can be made about the FDTD algorithm. The data transaction requirements are very asymmetrical. Only 10% of the data operations are writes, the rest are reads. This affects the approach to the future hardware accelerator. In order to compute an FDTD update equation, a great deal of input data is required. On the other hand, to return the completed calculation only requires about 10% of the total required bandwidth. In this way, it makes sense to push all of the simulation data as close to the hardware accelerator as possible, by using local memory (on the accelerator). Then, only updated fields are sent back to the host computer. These details are discussed further in Chapter 5.

The next section discusses the methodology used to explore the computation

[1](Or cache block) The smallest unit of memory that can be transferred between the main memory and the cache. [22] This takes advantage of (i) the principle of locality of reference, which implies that nearby memory locations are likely to be referenced in the future, and (ii) hardware memories are optimized for consecutive accesses.

speed for an individual field update. This is followed by the results of these methods, run on the reference computer.

3.3.2 Benchmark Methodology

As stated earlier, the goal of the investigation is to determine the computation speed for a single field update. Based on this speed, it should be possible to predict the runtime of an FDTD simulation with a certain three-dimensional mesh size and simulation length (number of time steps).

Experiments with the following characteristics are performed:

- The FDTD C++ code is compiled, using g++ (the Linux C++ compiler), with aggressive optimization enabled. This is representative of the real world situation for most software programs, where it is desirable that runtimes are as short as possible. Several experiments determined the best setting for the compiler optimization flags.

- A simulation model is chosen that is much larger than the cacheable memory. Later, for comparison, a simulation model is constructed that should fit entirely into cache.

- The simulation is run for a number of time steps, usually 100 or more.

- Each simulation is run 100 times and the measurements are collected for each iteration. Afterwards, the set of data is averaged to provide the typical performance of the code rather than a particular instance.

Two approaches are taken to measure the time spent by the algorithm on the FDTD update equations. The first method is to compile the FDTD program with profiling enabled. Then profilers, such as *gprof*, are used to analyze the profiling data that is generated when the program is run. Using specific identifiers or code markers, the FDTD update equations are separated from any other code. For the C++ code, without any enhancements to the FDTD algorithm, the update equations comprise 99% of the runtime. The second method uses the UNIX *time* function which provides CPU usage, page faults and runtime data for a program. Each method, (i) the *time* function applied to optimized, non-profiled code and (ii) the profiler data, reinforces the results of the other. Thus, either method can be used without a loss of accuracy.

This yields the runtime of the program, and more importantly the runtime of the FDTD update code, for a certain total number of field computations using calculations identical to Equation 3.2. The computation time for a single field update equation is given by:

$$t_{comp} = \frac{1}{f_{computation}} = \frac{t_{run}}{6 \times N_x N_y N_z \times N_{steps}} \tag{3.3}$$

Where t_{comp} = computation time for a single field update [s]

f_{comp} = computation speed for a single field update [Hz]

t_{run} = the (averaged) runtime reported by *gprof* or *time* [s]

N_x, N_y, N_z = the number of cells in each dimension

N_{steps} = the number of time steps in the simulation

The denominator of Equation 3.3 is really the total number of field updates

computed during the simulation. This number is also confirmed by a simple counter which is embedded within each update equation. For speed, it is only included in the code for verification, not during the benchmark runs.

It should be noted that t_{comp} represents the average computation time, for all fields, which will differ from the instantaneous value for each field update. This is due to the fact that, first, the run time is an average over several simulation runs. Second, some calculations will execute faster than others depending on the load on the system, whether a cache miss occurs and the order/scheduling and type of instructions in the CPU's pipeline at that instant.

Analysis of Equation 3.2 predicts that there are 10 values that are either loaded from or stored to memory in order to compute a field update. For this discussion, it is assumed that each (floating-point) value is 32-bits long. Thus, ten 32-bit values (40 bytes) of data are transacted per field update. Using the information from Equation 3.3 the average memory bandwidth (in MB/s) used during the FDTD computations is:

$$BW_{FDTD} = \frac{40bytes}{t_{computation}} \times \frac{MB}{1024^2 bytes} \tag{3.4}$$

With respect to Equation 3.2 it was originally stated that 10 values represented the worst case scenario for the number of values needed. For the case of the memory bandwidth, this number is in fact the most optimistic. Any reductions in the data requirements for a single update equation will further reduce the bandwidth used by the FDTD algorithm.

There are a few underlying assumptions, for this entire section, which should be

discussed before the results are presented. It is assumed that if the simulation model is sufficiently large (ie: too large to be contained in the cache) then the mesh size (N_x, N_y, N_z), the coefficient and field values, should not have an effect on the speed of the computation. Thus, it is assumed that the computation time for a single field update can be generalized and applied to all sufficiently large FDTD simulations. Initial research by the author indicates that this assumption is acceptable and that a calculated t_{comp} for a given memory and CPU configuration is valid for a wide range of FDTD mesh sizes.

3.3.3 Computation Speed and Memory Bandwidth for a Simple FDTD Code

Table 3.4 summarizes the simulation configurations used to determine the benchmarks for the FDTD update equations.

Table 3.4: Various Simulation Models Used to Measure Software FDTD

Simulation Name	Size	Number of Time Steps	Total Number of Fields
A	100x100x100	100	6×10^8
B	16x16x16	24,415	$600,023,040 \approx 6 \times 10^8$
C	3x3x3	4,146,042	$671,658,804 \approx 6.7 \times 10^8$

For each simulation A, B, and C (and later D, E, and F) the volume and simulation length are adjusted with respect to each other such that approximately 600 million field updates are performed by each simulation. This allows for comparison (of runtime and computation speed) between simulations. The FDTD simulations

are computed using the simplified C++ code described earlier.

Simulation A is designed so that the array of fields is much larger than the computer's cache. The goal is to force the CPU to fetch the data from physical memory. Simulation B is created after a large number of major and minor page faults are discovered (Table 3.5) in Simulation A. The hypothesis is that reducing the number of major page faults, which require the host computer to load a memory page from virtual memory (the hard drive), should greatly improve the speed. The last model, Simulation C, is designed to fit wholly inside the cache. The timing of this algorithm should reflect extremely high data rates from cache and the actual computation time of the arithmetic operations of Equation 3.2 in the CPU.

Table 3.5 describes the results that are obtained using the previous methodology, for the simulations described in Table 3.4.

Table 3.5: Computation Speed and Memory Bandwidth for FDTD Update Equations

Simulation	Total Number of Fields	t_{run} (s)	$t^{\dagger}_{comp}$ (ns)	$f^{\dagger}_{comp}$ (MHz)	BW_{FDTD} (MB/s)	Major, Minor Page Faults
A	6×10^8	64.35	107.2	9.32	355.69	35853, 11839
B	$\approx 6 \times 10^8$	38.60	64.32	15.55	593.10	295, 64
C	$\approx 6.7 \times 10^8$	33.64	50.08	20.0	761.7	167, 22

†Single Field Update

A few statements are necessary before the results of Table 3.5 are discussed. The computation time t_{comp} for a single field update contains timing information for both the CPU's arithmetic logic unit (ALU) and the memory. These two impacts, the ALU calculation time and the memory fetch time, are inseparable without detailed

knowledge of how the CPU executes the assembly instructions produced for a single FDTD update equation. The other thing to note is that a low value of memory bandwidth does not necessarily indicate that the data performance of the implementation is poor or could be improved. This value is an average bandwidth over the entire runtime of the simulation. The average may be comprised of very short bursts of peak bandwidth fetches from memory, which means that increasing the available memory bandwidth would improve the computation speed. On the other hand, the FDTD computation speed could be limited by the ALU. In this instance, the memory bandwidth will be under-utilized. Until the speed of the ALU or the CPU clock speed is increased, the data will not be processed/required any faster.

For now, without detailed knowledge of the CPU's execution, the ALU calculation time and the memory fetch time must be discussed together rather than separately.

There are a few statements which can be made about the results summarized in Table 3.5. Simulation A does not appear to use all of the available bandwidth, calculated earlier (458.5 MB/s) for PC133 memory. However, there are a large number of page faults, which would require the computer to fetch a page from the hard drive, effectively slowing down the program significantly. Simulation B reduces the number of page faults and the performance also increases. It is expected that much of this data might reside in cache, however. Finally, Simulation C reflects the computation rate where all data is assumed to be in cache. In this case, the performance is improved again. Based on the results of Simulation C, it is argued that the FDTD computation is not entirely limited by the ALU. When the data is provided at a faster rate from cache, the computation speed increases. This means that for at least part of the update computation, the CPU is waiting on the memory.

The most interesting observation is that the rate at which updated fields are produced is at most 20 MHz, even if it is an artificial case (Simulation C). At this rate however, the algorithm uses a lot of memory bandwidth which could be a problem in a system (the hardware accelerator) without cache.

In the next section, a well-used FDTD research code is profiled. Runtime results from the research code are compared to those obtained in Table 3.5. Following this, the best numbers for the computation speed and memory bandwidth are extrapolated to requirements for the hardware accelerator.

3.4 Profiling Totem

Totem is a research FDTD code developed by Okoniewski [32]. This code is a fully-functional simulator that incorporates many advances in FDTD [2] (absorbing boundary conditions, complex materials, complex metal geometries and wires) that the simple C++ code of the previous section does not. Based on the long term goal of the project, this software is one of the *existing* FDTD software implementations for which acceleration would be desirable.

The code is primarily FORTRAN-90 with a few routines written in C. The Intel High-Performance FORTRAN Compiler (for Linux) [33] is used to generate an executable for the reference computer. To ensure a valid comparison, the simplified C++ code is also compiled using the Intel High-Performance C++ compiler. This gives identical results to the code compiled using *g++* with aggressive optimization enabled. It should be noted that, in general, it is not desirable to write the inner loops of the FDTD algorithm in assembler. First, the development of mature and in-

telligent compilers mitigates the need to generate hand-optimized assembly code. In most cases, the compiler produces results that perform identically to the optimized-assembler at a fraction of the effort. Second, custom assembler does not facilitate porting the software to multiple platforms easily.

In addition to the core loops for the FDTD update equations, Totem contains a significant amount of initialization and data pre/post-processing code. The simulation runtime is no longer just the core update equations; thus, the difference between two simulation runs of different length could be used to determine the actual computation time in the core update equations. The assumption is that the prologue and epilogue time are constant for the simulation model and do not change with the number of time steps. Conveniently, Totem computes the time for the core FDTD update equations only, as part of its functionality.

Identical simulation models as those in Table 3.4 are created and run using Totem. Simulation model D contains 100x100x100 cells. The runtime for 100 time steps, averaged over 1,000 runs, is 42.47 s. Similarly, the runtime for 200 time steps is 84.88 s. Subtracting the two values gives a runtime of 42.41 s for 100 time steps. Table 3.6 presents the computation rate and memory bandwidth for the three simulation models in Totem.

The performance of Simulation D is significantly better than the results of the simplified C++ code (Simulation A: $t_{run} = 64.35\ s$, $t_{comp} = 107.2\ ns$, $BW_{FDTD} = 355.69\ MB/s$) presented in the previous section. The performance of Totem is actually in line with Simulation B (of the previous section), which was designed to reduce page faults. Simulation D has significantly less page faults than Simulation A, even so the mesh for both is 100x100x100 cells. Similarly, Simulation E performs

Table 3.6: Computation Speed and Memory Bandwidth for Totem

Simulation	Total Number of Fields	t_{run} (s)	$t^{\dagger}_{comp}$ (ns)	$f^{\dagger}_{comp}$ (MHz)	BW_{FDTD} (MB/s)	Major, Minor Page Faults
D	6×10^8	42.41	70.68	14.15	539.68	583, 21480
E	$\approx 6 \times 10^8$	27.67	46.11	21.68	827.22	583, 624
F	$\approx 6.7 \times 10^8$	210.1	312.8	3.20	121.95	584, 32717

†Single Field Update

better than Simulation B. It would appear that the FORTRAN compiler produces faster code than the C++ compiler.

There is one anomalous result. The computation of the 3x3x3 mesh in Simulation F takes the longest time measured for any of the simulations. The suggested cause is that due to the large number of time steps, there is a disproportionate amount of output (to the screen and hard drive) for the simulation. Thus, the computation time is very short as expected but other factors utilize the runtime.

It is worth noting that the number of major page faults in the Totem runs are constant. Thus, it appears that these faults will occur regardless of the simulation size. This is unlike the simplified C++ code which has a large number of page faults for the larger simulation. Obviously, the memory allocation scheme used in the simplified C++ code could be improved. The C++ language does not provide for the dynamic declaration of three-dimensional arrays, without additional coding, so a less desirable implementation is used. This C++ implementation suffers from a large number of page faults while the FORTRAN code does not.

For sufficiently large FDTD simulation models, Totem provides the most compet-

itive computation speed and memory bandwidth results for the two codes measured. These best numbers are used to predict the speed and bandwidth requirements for the accelerator.

3.5 Extrapolation of Previous Results

In nearly all typical FDTD simulations, it is predicted that the model is larger than the computer's cache. Simulations smaller than this would generally have an insufficient mesh size to produce useful results. For sufficiently large simulations, the Totem code produced the fastest computation speed and highest bandwidth utilization. This means that the proposed hardware accelerator needs to have a computation speed and memory bandwidth (Table 3.7) which is an order of magnitude larger than the reported Totem results.

Table 3.7: Computation Speed and Memory Bandwidth Requirements for the Proposed Hardware Accelerator

Simulation	f_{comp} (MHz)	BW_{FDTD} (MB/s)
Hardware Accelerator	150	5397

The requirements of Table 3.7 can be satisfied in a number of ways. The available choices, for the hardware accelerator to meet these requirements, are described in greater detail in Chapter 5. Given the presented memory technologies and available custom and/or programmable hardware, these requirements are reasonable.

Chapter 4

The FDTD Computational Engine

"I think there's a world market for about 5 computers."

- Thomas J. Watson, Chairman of the Board, IBM, 1943

This chapter describes the first approach to a hardware FDTD implementation. For this approach, the entire simulation is implemented on programmable hardware. One- and two-dimensional hardware FDTD implementations are verified using microwave cavity resonators. The acceleration and simulation accuracy for both implementations are discussed.

4.1 Description of the Approach

The goal of the implementation approach described in this chapter is to build a self-contained FDTD simulation on the hardware. Each and every electric and magnetic field in the simulation is stored and calculated simultaneously. This yields the maximum possible parallelism for the FDTD computations. To achieve this high level of parallelism and thus acceleration, a large amount of digital hardware is required. A different approach, which is more suited to practical application and based on resource sharing, will be described in Chapter 5.

4.2 The Hardware Implementation of FDTD

In Chapter 2, a block diagram is derived (Figure 4.1) for the computation of voltages and currents. A direct correspondence between this diagram and the FDTD algorithm is also demonstrated.

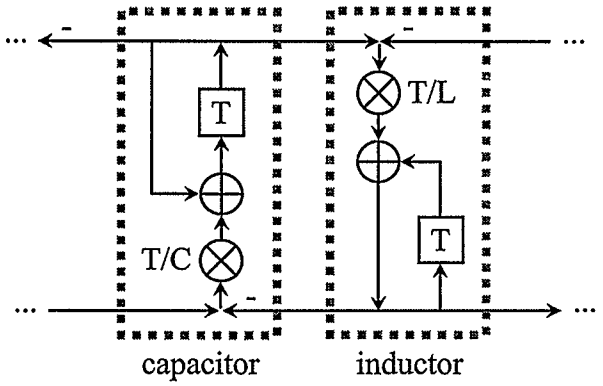


Figure 4.1: Block Diagram of an Inductor-Capacitor FDTD Computation

The arithmetic operators for the block diagram of Figure 4.1 could be implemented using a number of different digital implementation paradigms. The choice of paradigm recognizes the trade-offs between speed and hardware utilization (hardware cost). The spectrum of choices (Figure 4.2) can be arranged into bit-parallel, digit-serial and bit-serial computations.

At one end of the spectrum there is bit-parallel computation, where all the resultant bits of an arithmetic operation are generated at the same time. At the other end of the spectrum is bit-serial, where the result of an arithmetic operation is generated one bit at a time. In between is N-digit-serial, representing a mix of bit-parallel and bit-serial, where N-bits ($2 \leq N < N_{parallel}$) of the result are computed at the same

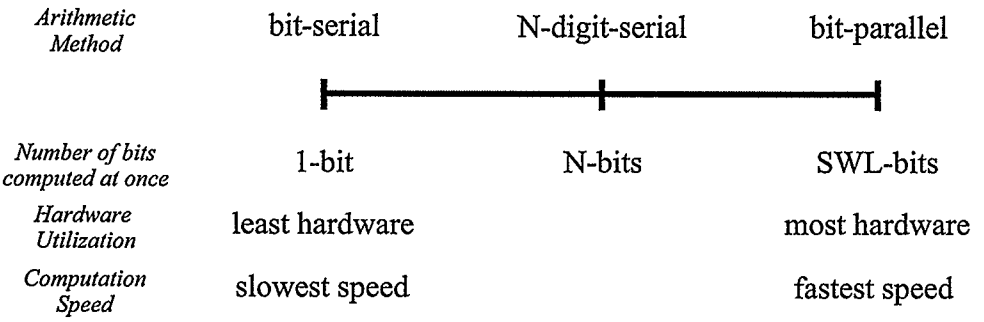


Figure 4.2: Hardware Utilization vs. Speed Trade-Offs for Different Arithmetic Implementation Methods

time. Each paradigm reflects different speed vs. hardware cost trade-offs.

Bit-parallel will typically result in the fastest computational speed, at the expense of using the most hardware. Moreover, the computation speed is ultimately limited by the carry-propagation path/delay between parallel elements. In an FPGA, the routing (resource) cost is also very high because of the parallel data path.

Bit-serial will typically result in a slower computation speed, but use the least hardware. For an M-bit system wordlength[1] (SWL), bit-serial will use approximately $1/M$ times the hardware (compared to bit-parallel) but it will take M times longer to compute the full SWL result. In essence, each bit-serial operator is re-used for each bit of the result.

Typically, fully pipelining a bit-parallel design requires a large amount of hardware; this is usually undesirable. On the other hand, bit-serial arithmetic is deliberately registered between each operator and, thus, fully pipelined. The short, serial propagation paths and short carry chains allow a bit-serial design to operate at higher clock frequencies than their bit-parallel counterparts.

[1]System Wordlength: describes the number of bits used to represent data in the digital system.

Early analysis by the author indicated that dedicated hardware would provide more than enough acceleration. The results presented later in this chapter (Section 4.5) reinforce this statement. It was predicted that data communication and hardware size, not computational speed, would become the limiting factors. With this in mind, emphasis is placed on minimal hardware utilization as opposed to computational speed. It is more desirable to have many, slower FDTD cells instead of a few extremely fast ones. The size of individual FDTD cells and the available FPGA resources will place an upper bound on the size of the FDTD simulation that can be fully implemented. Thus, the main objective of the following sections is to reduce the hardware size of the FDTD implementation. In summary, two key choices are made regarding the final implementation:

- A pipelined bit-serial architecture is chosen.
- Arithmetic operations are performed using fixed-point (integer) arithmetic.

These choices are described in the following sections.

4.2.1 Pipelined Bit-Serial Architecture

An example of a generic, bit-serial operator is depicted in Figure 4.3.

The inputs to this generic operator are: two data-inputs, a serial (bit) clock and a control signal. There is also one data output. For the purposes of this work, the data words are serial bitstreams arranged with their least-significant bit (LSB) first. There is the assumption that the serial bitstreams are aligned and the LSB's of the input data words arrive at the block at the same time. The control signal is used for LSB time framing, especially when multiple arithmetic operations are pipelined.

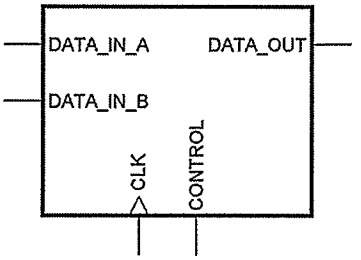


Figure 4.3: Generalized Bit-Serial Operator

For a fully synchronous design, the input and output data bits should be synchronous with the serial clock. This implies that the input data signals are registered/latched by the previous hardware (not shown), with respect to the serial clock. Likewise, the output data signal is registered/latched by the generic operator, again, with respect to the serial clock. This means that the result of the 1-bit arithmetic operation is not available until at least one serial clock cycle after the inputs have arrived. This single bit of delay (one serial clock) could be extended to an arbitrary amount of delay, depending on the arithmetic operation.

Control signals are used for system wordlength framing, to identify the arrival of the LSB at the inputs. Or, in other words, the control signal marks the boundary between adjacent words in the serial bitstream. As the serial bitstreams pass through the operators, the data path is delayed and this also requires that the control path be delayed.

Consider, for instance, two eight-bit words, 011100001_2 and 001111000_2, incident on a bit-serial adder. Figure 4.4 depicts the addition operation for this data.

If the LSB's of each word arrive at $T = 0$ then the LSB of the result appears at $T = 1$. As time continues to $T = 9$, the two input words are added bit by bit.

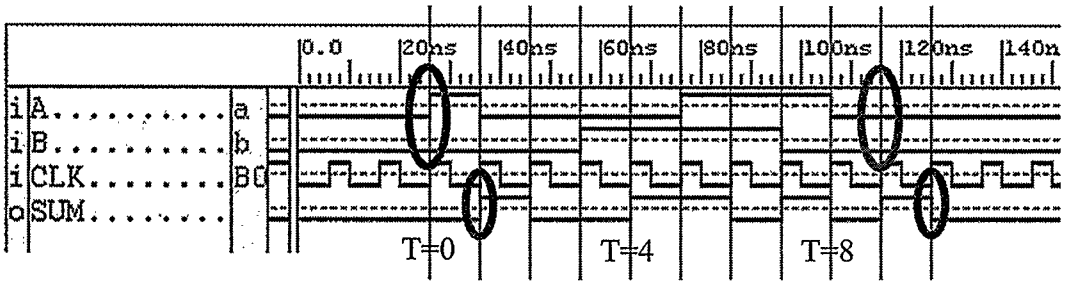


Figure 4.4: Example of Bit-Serial Addition

As explained previously, the generic bit-serial operator only operates on one bit from each of the input words per serial clock cycle. This implies, for the moment, that all the other bits of the input and output data words are just buffered or stored while the arithmetic operation is completed. This is inefficient, especially for long wordlengths.

In order to increase the number of arithmetic operations that are performed simultaneously, the serial computation path is pipelined by placing a register at the output of each bit-serial operator. By chaining together several bit-serial operators, the computation of one operator is overlapped with other operators' computations. Each operator will be processing different bits of the serial bitstream, for a given serial clock cycle. In order to achieve this pipelining, the only requirement is that the serial bitstreams are aligned throughout the structure. In simpler terms, the LSB's of the input data words must arrive at an operator at the same time. This implies that pipelined bit-serial needs to be scheduled. As future sections will show, the bit-serial operators are relatively simple. It is the scheduling of a complex computation that becomes the most difficult part of an implementation.

Consider the delay T in the block diagram for the one-dimensional FDTD computation in Figure 4.1. For a bit-parallel implementation this delay is, in fact, one

clock cycle. However for a bit-serial implementation, the delay T is in fact a number of serial clock cycles, equal to the system wordlength. To achieve proper alignment with the bit-serial operator following T, the LSB of the data word is delayed by SWL bits. This is very useful for the bit-serial implementation because the system wordlength amount of delay can be distributed throughout the pipeline. Again, proper scheduling becomes very important. These concepts are demonstrated in Section 4.4.2.

In summary, pipelined bit-serial arithmetic is chosen for the following reasons:

- The hardware utilization (cost) of pipelined bit-serial arithmetic units is low. Adders, subtractors and delays are reused for each bit of the system wordlength.
- The size of each computational unit is small, compared to parallel or digit-serial, allowing the most computational units to be implemented in parallel for a fixed amount of hardware.
- The bit-serial structure allows for very short routing lengths reducing hardware costs and simplifying routing. Low hardware costs should facilitate more FDTD cells on a given amount of hardware. Furthermore, short routing lengths equate to a faster operating frequency because the propagation/path delays are less.

4.2.2 Integer Arithmetic

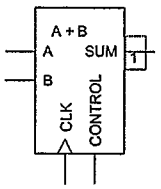
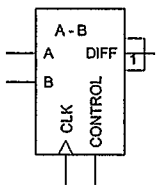
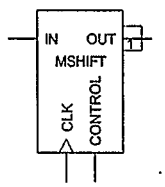
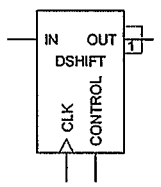
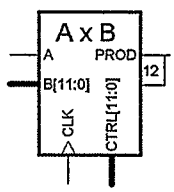
Integer arithmetic is chosen over floating-point arithmetic to further reduce hardware costs, and additionally to increase the computational speed. This is offset by the need for larger integer registers in order to maintain the dynamic range provided by a floating-point representation.

The next section describes the bit-serial, 2's-complement integer arithmetic implementation in greater detail.

4.3 Pipelined Bit-Serial Arithmetic: Basic Building Blocks

The pipelined bit-serial building blocks used to implement the FDTD algorithm are: bit-serial adders, bit-serial subtractors, arithmetic left/right shifters, arbitrary delays, and N-bit signed multipliers. Table 4.1 introduces these operators and their functions.

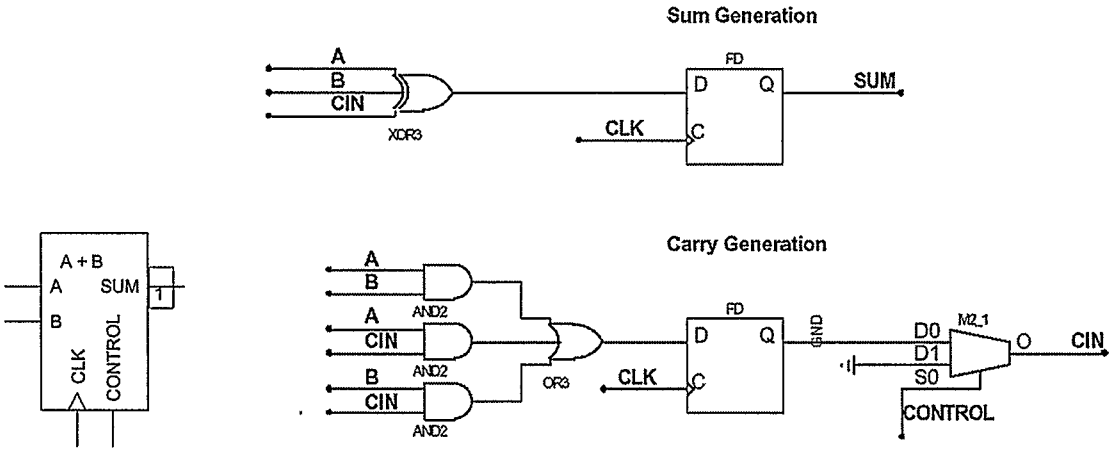
Table 4.1: Overview of the Bit-Serial Operators

Block Diagram	Bit-Serial Operator	Function	Output Delay ($N \times T_{serial}$)
	Adder	Adds the two input serial bitstreams, one bit at a time per serial clock cycle.	1
	Subtractor	Subtracts the two input serial bitstreams (A-B), one bit at a time per serial clock cycle.	1
	Arithmetic Left Shifter (MSHIFT)	Multiplies the input serial bit stream by 2, one bit at a time per serial clock cycle. It also inserts zeroes for the LSB†.	1
	Arithmetic Right Shifter (DSHIFT)	Divides the input serial bit stream by 2, one bit at a time per serial clock cycle. It also sign-extends the MSB‡ if necessary.	1
	Delay	Delays the input serial bitstream, by one to system-wordlength (SWL) bits.	1 to SWL
	N-Bit Coefficient Multiplier	Multiplies the input serial bit stream by an N-bit, parallel coefficient. It also truncates N-bits of the result.	N

†LSB: Least Significant Bit
‡MSB: Most Significant Bit

The bit-serial operators (Table 4.1) and the required control structures are described in the following sections. Designs for these building blocks are adapted from [19] and Denyer and Renshaw [20]. The hardware size of each unit is reported using 3 metrics, defined in Section 2.1: the number of (i) flip-flops, (ii) lookup tables (LUT's) and (iii) Virtex slices.

4.3.1 Bit-Serial Adder



(a) Block Diagram

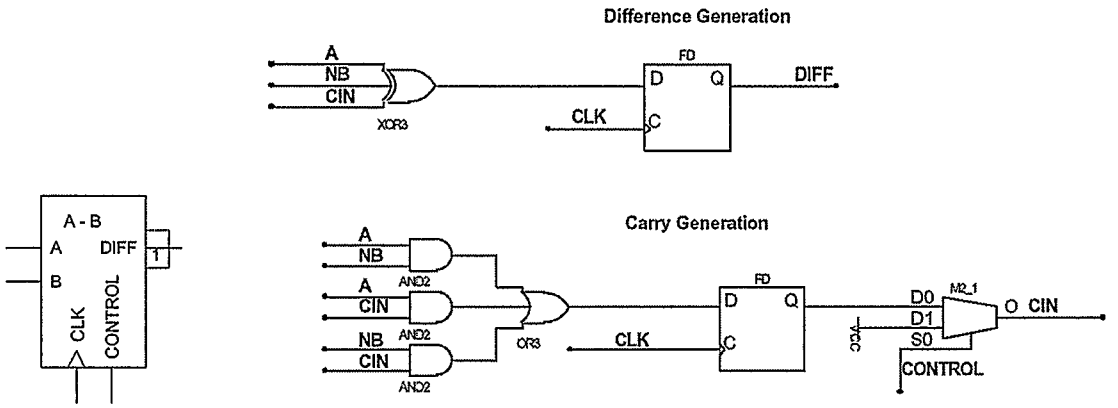
(b) Logic Gate Implementation

Figure 4.5: Bit-Serial Adder

The bit-serial adder, depicted in Figure 4.5, is a 1-bit, carry-save adder. It does not generate the result of the addition of individual bits A and B until one clock cycle later. The carry is delayed by a clock cycle as well so that it can be applied to the following, next significant bit in the serial word. When the control/framing signal is Active High, identifying the arrival of a LSB, the carry is zeroed.

This adder occupies one Virtex slice, specifically two flip-flops and two four-input lookup tables (LUT's).

4.3.2 Bit-Serial Subtractor



(a) Block Diagram

(b) Logic Gate Implementation

Figure 4.6: Bit-Serial Subtractor

The bit-serial subtractor is depicted in Figure 4.6. Again, the result is delayed by a clock cycle. For subtraction, the B-input is inverted (denoted NB) and the carry is set to be '1' when the LSB enters the block. This performs an 'invert and add 1' operation so that when addition takes place the B input is subtracted from the A input.

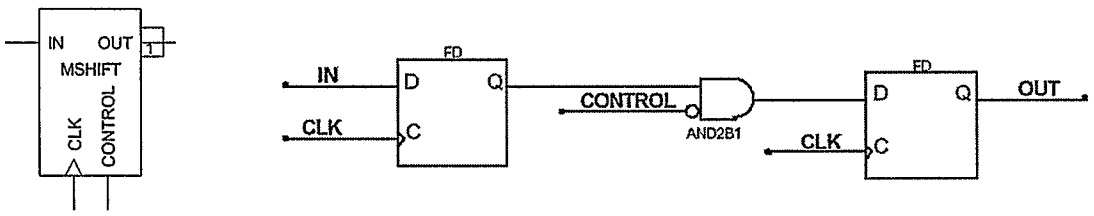
This subtractor occupies one Virtex slice, in particular two flip-flops and two LUT's.

It should be noted that for both the bit-serial adder and bit-serial subtractor designs the "carry" signal (CIN) is used as an internal signal only. Any carry is not

considered as an input or output in these blocks because the carry is zeroed at the arrival of the LSB and then stored internally for application to the next bit in the serial bitstream.

4.3.3 Arithmetic Left Shifter

The left-shift operator (MSHIFT [20]), depicted in Figure 4.7, performs a multiply by two on the signed serial bitstream.



(a) Block Diagram

(b) Logic Gate Implementation

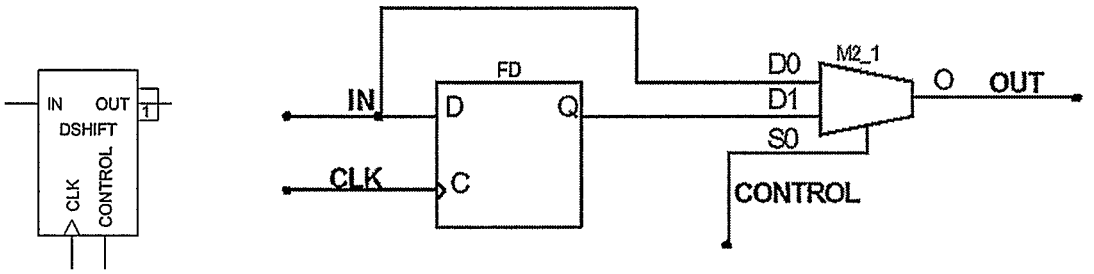
Figure 4.7: MSHIFT

Externally, this operator is assigned a delay of one bit time. Internally, there are two bit times of delay in the data path. This has the effect of delaying the input bitstream by an additional clock cycle, effectively (shifting the word left and) multiplying by two. The control signal is used to insert zeroes at the output when the LSB is expected.

This operator occupies 1 Virtex slice, using two flip-flops and one LUT respectively.

4.3.4 Arithmetic Right Shifter

The right-shift operator (DSHIFT [20]), depicted in Figure 4.8 performs a divide by two on the serial bitstream.



(a) Block Diagram

(b) Logic Gate Implementation

Figure 4.8: DSHIFT

Externally, this operator is assigned a delay of one bit time. Internally, there is no delay in the data path. The LSB arrives one clock cycle early, effectively dividing by two. For a truly synchronous design, this part should actually contain a single flip-flop in the primary data path and be assigned two delays externally. Our implementation requires less hardware, but does not latch/register the output. As long as the propagation delay through the part is negligible this should not affect its function; this component appears to be pipelined even so it is not.

The control signal is used to sign-extend the data value if necessary. This operator occupies one half of a Virtex slice, using one flip-flop and one LUT respectively.

It should be noted that it is possible to create MSHIFT and DSHIFT components that shift the input data value by more than one bit. This can be achieved by using two methods: (i) simply by chaining several MSHIFT or DSHIFT operators (as

described in the previous sections) to produce larger shifts or (ii) by combining the logic level circuits. The second option would allow some of the overlapping output registers and combinational logic to be absorbed between adjacent shifts.

4.3.5 Delays: One Bit to System-Wordlength Bits

In a complex bit-serial system, the data path lengths of different bitstreams will vary when measured relative to the inputs of a given operator. In order to ensure that the LSB's of different input bitstreams arrive at the same time, it is necessary to delay the data path. Delays from one bit time to system wordlength bits may be required. Delays larger than two to three bit times in length can be constructed efficiently using linear-feedback shift-registers (address generation) and LUT's (dual port RAM). The reader is referred to the Xilinx Application note [34] for more information. The designer has control over using only flip-flops or a combination of flip-flops and LUT's to implement delays, depending on resource availability.

For this design, delays of 3 to 16 bit times occupy 2.5 Virtex slices. Delays of 17 to 32 bit times occupy 3.5 Virtex slices.

4.3.6 N-bit Multiplier

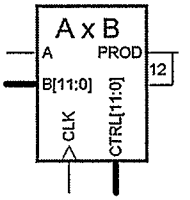


Figure 4.9: Bit-Serial Multiplier (with 12-bit Parallel Coefficient)

This operator, adapted from [19], multiplies the serial bitstream by an N-bit

(parallel) coefficient. The multiplier is signed-number capable and the range of coefficients (X) is:

$$-0.5 = \frac{-2^{N-1}}{2^N} \leq X \leq \frac{2^{N-1}-1}{2^N} \approx 0.5 \tag{4.1}$$

In general, when two integers are multiplied, an M-bit word by an N-bit coefficient, this produces an M + N bit result. This multiplier truncates the lower N-bits of the result automatically, otherwise the system wordlength would increase after every multiply operator.

The multiplier operator consists of three main parts, each end slice and the middle slice(s), as depicted in Figure 4.10.

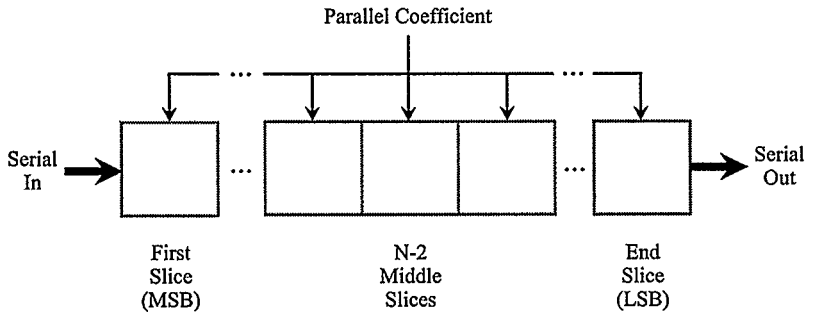


Figure 4.10: Construction of an N-bit Multiplier

The multiplicand, A, is slid past the N-bit coefficient for system wordlength clock cycles. When the LSB arrives at the input to the multiplier (and for N-1 additional clock cycles after this) the output of the previous word is still being generated. From Figure 4.11 it can be seen that a sum of products (SOPO) is generated by a full adder (inputs: SI - sum in, CI - carry in, outputs: SO - sum out, CO - carry out). The sum of products is then passed to the next slice as an input. In fact, the sum of

products for the entire N-bit column is computed and the resultant bit output from the block. Once again, the carry is delayed by one time unit to affect the generation of the next sum of products, and the carry is zeroed when the LSB is in the slice.

This path through the N-full adders is the longest path of the bit-serial implementation and will have the greatest impact on the final computation speed of the entire circuit.

The slice associated with the most-significant bit of the coefficient, depicted in Figure 4.12, is very similar to the middle slices except that the sum of products input is zeroed. There is one bit time of delay for each coefficient bit.

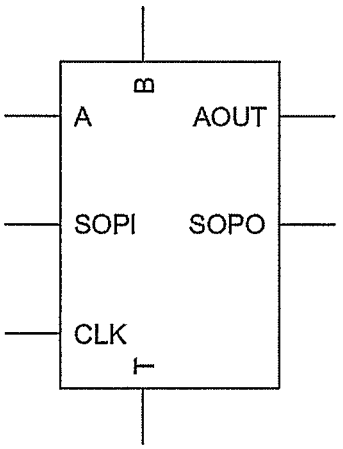
For the first N-1 slices, the control signal is used to clear the carry bit between the MSB and LSB of two adjacent words travelling through the multiplier.

The slice associated with the least-significant bit of the coefficient, depicted in Figure 4.13, is slightly different in structure compared to the other slices. When a LSB is not incident on this part, and thus a control signal is not applied, its operation is identical to that of the middle slice. The operation of this block differs when the second control signal is applied, for the LSB. The second control signal (T_{FB}) is used to sign extend the MSB of the result, if necessary. As well, it restarts the carry chain for the next serial input.

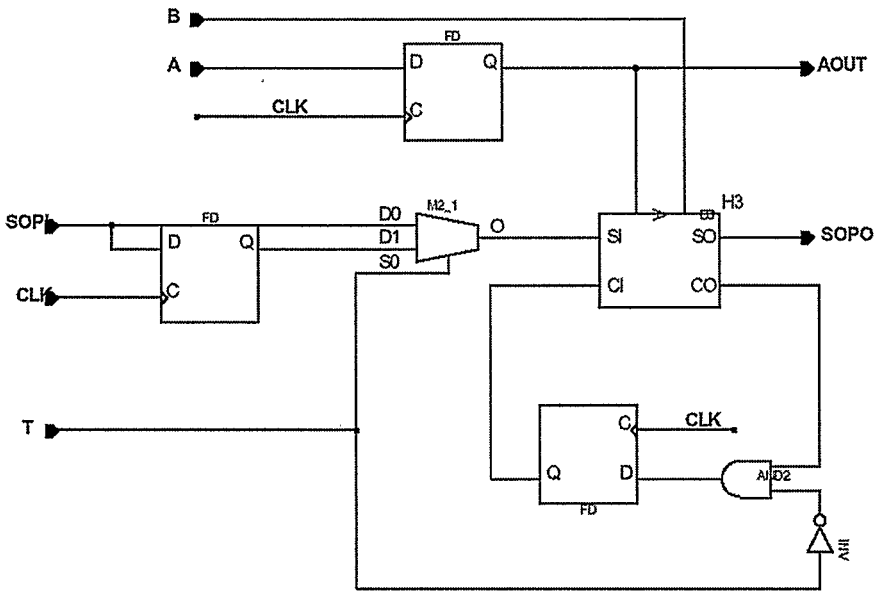
The cost of a 12-bit multiplier is 29.5 slices, with 35 flip-flop's and 37 LUT's.

4.3.7 Control structure

As indicated previously, a control signal is required when the LSB arrives at the input to an operator. Because of the delay associated with each operator and the pipelined nature, the LSB of a bitstream may arrive at different clock cycles for

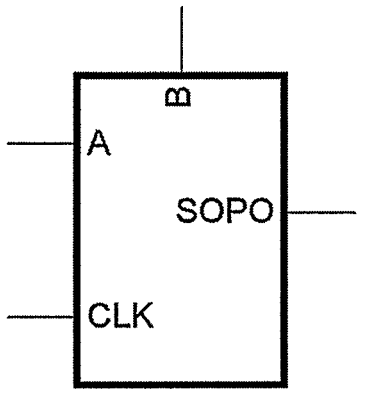


(a) Block Diagram

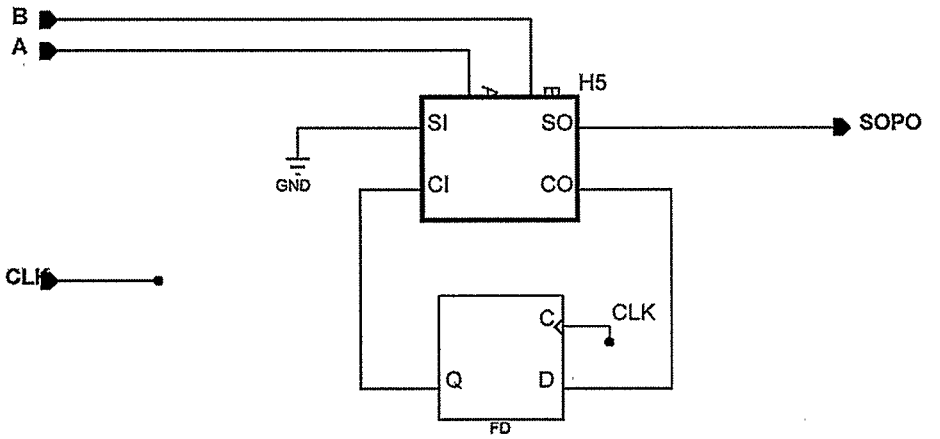


(b) Logic Gate Implementation

Figure 4.11: The Bit-Serial Multiplier: Middle Slice

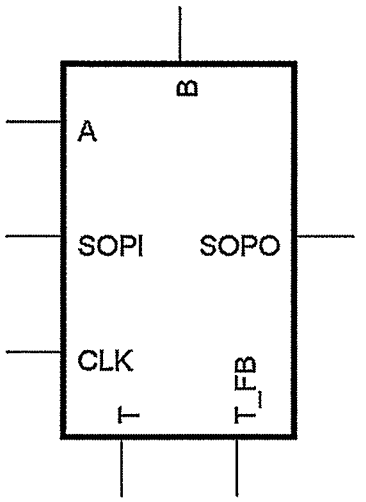


(a) Block Diagram

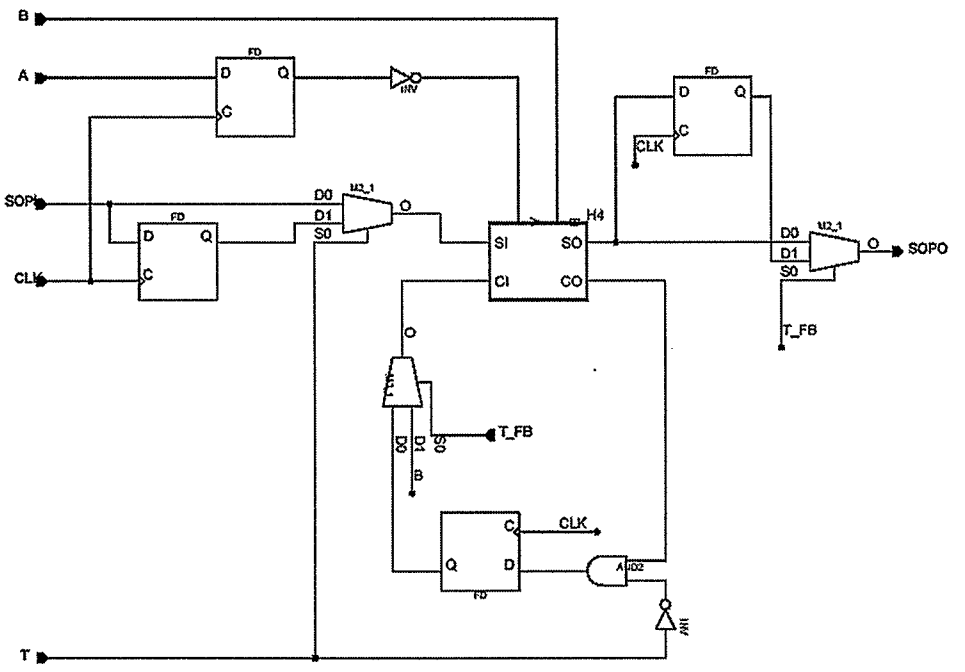


(b) Logic Gate Implementation

Figure 4.12: The Bit-Serial Multiplier: First (Most-Significant Bit) Slice



(a) Block Diagram



(b) Logic Gate Implementation

Figure 4.13: The Bit-Serial Multiplier: End (Least-Significant Bit) Slice

different operators in the circuit.

It is necessary to generate a control structure, which can output a control signal in all possible bit periods of the system wordlength. The simplest solution is to use a "one-hot ring" counter with the number of states equal to the system wordlength. Such a control structure for a system wordlength of 32-bits costs 16 Virtex Slices.

4.3.8 Summary

Table 4.2 summarizes the cost, in terms of Xilinx Virtex-family slices, for the various units described in the previous sections.

Table 4.2: Hardware Cost of Various Pipelined Bit-Serial Arithmetic Units

Arithmetic Block	Virtex Slices	Flip-flops	LUT's
Bit-Serial Adder	1	2	2
Bit-Serial Subtractor	1	2	2
Left Shift(MSHIFT)	1	2	1
Right Shift (DSHIFT)	0.5	1	1
Delay (3-16 bits)	2.5	4	3
Delay (17-32 bits)	3.5	5	4
12-bit Multiplier (per bit)	29.5(2.5)	35(2.9)	37(3.1)
32-bit Control Structure (per bit)	16(0.5)	32(1)	0

4.4 Experimental Verification of Hardware FDTD

The goal of this part of the project is to investigate the feasibility of implementing FDTD cells in hardware and to determine the resulting speed and size of such cells. Two simulation models, representing one-dimensional and two-dimensional microwave resonators, are constructed for both hardware and software platforms.

For this discussion, a microwave resonator is a lossless cavity bounded by reflective walls. These walls are implemented using perfect electric conductors (PEC's), also known as electric walls (which maintain $E_{tangential} = 0$), or perfect magnetic conductors (PMC's), also known as magnetic walls (which maintain $H_{tangential} = 0$). Some initial disturbance or energy is excited inside the cavity. When an electromagnetic wave is incident upon one of the boundaries, all of the energy is reflected back into the cavity. Constructive and destructive interference occurs and the fields inside the cavity will resonate. The frequencies of resonation are determined by the material properties, physical dimensions of the cavity and the spectral components of the initial excitation.

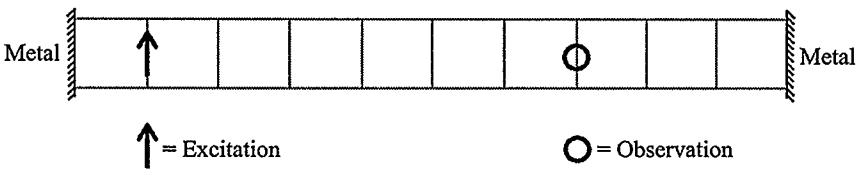
For each resonator, the hardware and software simulations use the same stimulus and observation points so that the results from either platform can be compared. Figure 4.14 shows the two resonators that are used.

The primary measure for accuracy of the various simulations is the location of the resonant frequencies. These values can be analytically predicted as well as computed, using the Fourier Transform, from the FDTD simulation results.

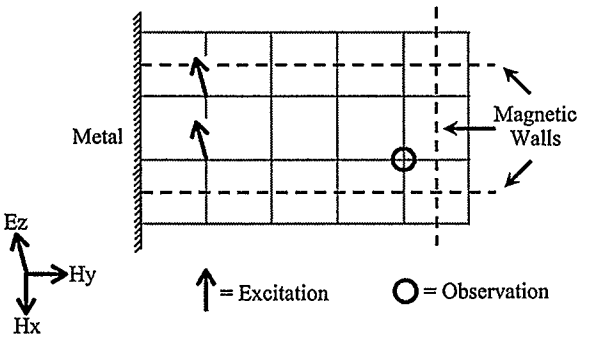
The results of the hardware and software simulations can also be compared against each other. This generates some useful insights, including the fact that results may need to be adjusted for the quantization of the coefficients. These are discussed fully in the results sections (4.4.4 and 4.4.7).

4.4.1 The Hardware Platform

The FPGA used to implement the hardware FDTD simulations is the Xilinx Virtex Family FPGA, XCV300, PQ240 package, speed grade 4, and it offers 3,072 slices.



(a) One-Dimensional Resonator



(b) Two-Dimensional Resonator

Figure 4.14: Excitation and Observation Points for the Two Resonators

The FPGA is situated on an XESS Development board [18]. This board is used because it offered the latest available Virtex part, at the time, and is large enough to implement "proof of concept" designs.

4.4.2 One-Dimensional Resonator

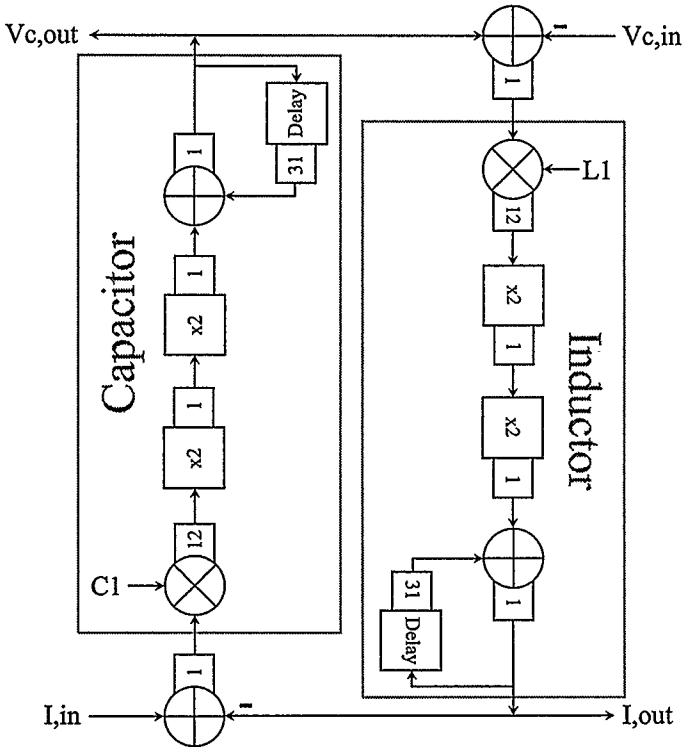
One-Dimensional FDTD Cell Implementation

The circuit in Figure 4.1 can be implemented using the pipelined bit-serial technology described in the previous sections. The resulting cell is given in Figure 4.15.

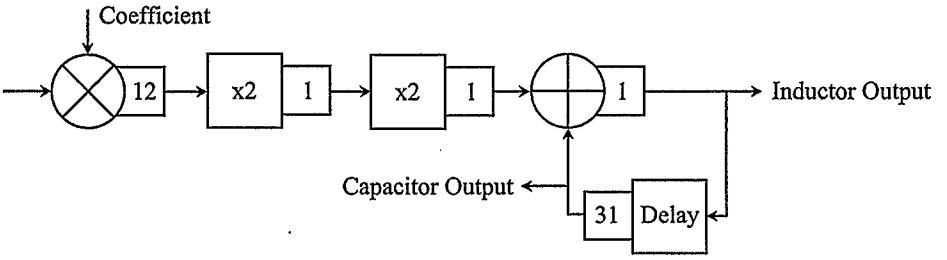
The design in Figure 4.15 uses a system wordlength of 32-bits and 12-bit coefficients. The boxed numbers at each operator output represent the delay through the block. Control signals are distributed around the circuit to mark the arrival of the LSB at each operator in the loop. Each delay from Figure 4.1 is 32-bits (system wordlength) long. The capacitor's delay is distributed between its adder and the rest of the inductor/capacitor loop, requiring 31-bits of delay in the feedback path. The inductor's delay represents the desired system wordlength delay before it is added back into the data path. The multipliers are followed by a multiply by four, which is used to change the range of coefficients (magnitude larger than one) that can be represented.

It is worth noting that, due to the symmetrical nature of the design and calculation of the two 'fields', the inductor and capacitor structures are identical. It is expected that this will not always be the case.

For more accuracy in the computations, the two left shift operators should actually be placed before the multiplier. It would be the hardware designer's and FDTD programmer's responsibility to ensure that no overflow would occur following this



(a) Bit-Serial Implementation (32-bit System Wordlength)



(b) Structure for Capacitor and Inductor

Figure 4.15: One-Dimensional FDTD Cell Implementation

multiply by four. The current implementation is safer because the chance of an overflow is reduced by multiplying by a coefficient less than one and then multiplying by four. However, this is achieved at the expense of 2-bits of accuracy. In the current case, the multiplier discards 12-bits of the full result and then this truncated value is again shifted (up) left by 2-bits. Two of the originally discarded bits will now be substituted as zeroes by the left shift operators.

Additional circuitry, not shown, is used to reset the fields in the cells to zero or initialize the fields values to an excitation value. In general, control structures are shared among a maximum of five one-dimensional, computational cells. After this, a new control structure is added. The intention is to localize the control signals and avoid the effects of clock skew.

Resonator Description

A one-dimensional resonator, terminated in perfect electric conductors (PEC's) is constructed. The resonator is depicted in Figure 4.16. The PEC causes the inbound wave to be reflected back into the resonant structure and can be represented using the one-dimensional cell without significant modification.

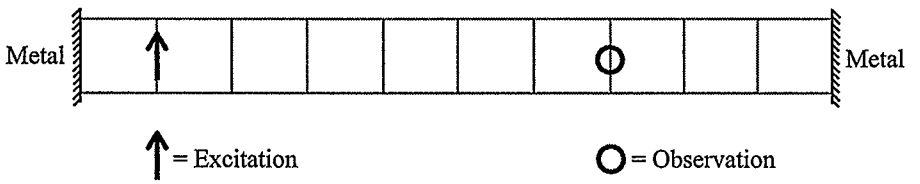


Figure 4.16: One-Dimensional Resonator

A resonator represents a trivial example, nevertheless, it is very useful for veri-

fication of the algorithmic implementation. Errors in the calculations quickly accumulate and the output may become unbounded. The resonant frequency amplitudes are several orders of magnitude above the noise floor and narrowband. The coefficients directly relate to the location of the resonances, further verifying the multiplier structure.

The excitation is a short, time-domain impulse to create a wideband excitation within the resonator. The impulse is realized by biasing one of the capacitors with a non-zero value at the start of the simulation. Due to the electromagnetic behavior, the spatial location of the impulse will also affect which resonant frequencies appear in the structure and their strength.

Coefficients are chosen such that $\Delta x = 1.0cm$ and $\varepsilon_r = \mu_r = 1.0$ (related to the inductor and capacitor values, respectively), which signifies free space. By experiment, it was found that 8-bit coefficients did not result in bounded-input, bounded-output (BIBO) stability. Increasing the coefficient accuracy to 12-bits provides stability. Further research is necessary to determine (and predict) the necessary register width (i.e. number of bits) for the coefficient and field values to achieve a desired simulation accuracy and computation speed. The necessary register widths to achieve a desired FDTD performance (accuracy and speed) is implementation dependent so it is desirable to perform this research once a final implementation has been chosen. Furthermore, the stability of the finite-precision FDTD algorithm will need to be analyzed fully.

Using 10 cells, this yields a resonator 10.0 cm in length. As mentioned before, the rule of thumb in FDTD is to use a minimum of 10-20 samples per wavelength. This means that the fundamental resonant frequency would be sampled accurately

but the second harmonic would be under-sampled.

Choice of Coefficients

The coefficient quantization has an important impact on the simulation's time-domain results. This effect is discussed in Section 4.4.4.

Given that $\varepsilon_r = \mu_r = 1.0$ and $\Delta x = 1.0cm$ the value of the capacitance is $C = 8.854 \times 10^{-13}F$ and the inductance is $L = 1.257 \times 10^{-9}H$. Finally, setting $\Delta x = 1.0cm$ and using a stability factor of 95% in the Courant condition yields a sampling interval of $\Delta t = 3.169 \times 10^{-11}s$.

The values of the coefficients are calculated as $\Delta t/C = 35.79s/F$ and $\Delta t/L = 0.0252s/H$. By performing impedance scaling (using 71.579)[2] the capacitor coefficient, $\Delta t/C$, is normalized to 0.5 and the inductor coefficient, $\Delta t/L$, to 1.805.

The capacitor coefficient (0.5) quantizes without error, but the inductor coefficient is represented as

$$\frac{1848}{2^{12}} \times 4 = 1.8047 \tag{4.2}$$

which includes a 12-bit multiply and the two left shift operators.

4.4.3 One-Dimensional FDTD Speed and Hardware Utilization

The bit-serial design on the XCV300-4 FPGA runs with a maximum bit-clock of 37.7 MHz, as reported by the Xilinx tools. A new result is available every system wordlength clock cycles or 849 ns (f_{op} = 1.18 MHz). Each one-dimensional computational cell utilizes 86.5 Virtex slices. The resonator, 10 cells in length, uses 917 (30%) of the available slices. 52 slices are used to gather the data from the

[2] $\Delta t/C$ is divided by 71.579 and $\Delta t/L$ is multiplied by 71.579.

simulation, yielding 865 slices for the computation and control structure.

As mentioned earlier, the pipelined bit-serial structure yields very short routing lengths. The average connection delay is 1.771 ns and average delay for the worst 10 nets is 4.208 ns.

Comparison of Hardware versus Software Runtime

Of most interest is the comparison of the simulation runtime between the hardware and the software. After all, the goal of the hardware implementation is to achieve an acceleration of an order of magnitude or more compared to the software. Table 4.3 compares the hardware and software runtime. The software runtime is measured using a modified version of the simple C++ code (for one-dimensional FDTD) that was previously described in Chapter 3.

Table 4.3: Runtime for the One-Dimensional Resonator Simulations

Simulation Method	Runtime† (ms)	Acceleration
Software	71	
Hardware	8.49	8.4X

†10,000 time steps

These acceleration results are already quite exciting because they are inline with the thesis objectives. The results confirm the hypothesis that mapping the FDTD algorithm to hardware would produce the desired acceleration.

It is worth noting that the Xilinx Virtex series part used in the experiments is not the latest technology. Using current FPGA technology, it is expected that the serial clock of 37.7 MHz could be increased to a few hundreds of MHz. Any serial clock increase translates directly into more acceleration.

While the hardware implementation is fast, it is also important that it is accurate. The next section discusses the simulation results for hardware FDTD.

4.4.4 One-Dimensional FDTD Simulation Results

Three different simulations are run for the one-dimensional resonator case: a hardware simulation, a standard software simulation and then a software simulation with an adjusted time step. The reasons for the latter simulation are discussed in later sections.

All three simulation methods provided nearly identical accuracy in determining the resonant frequencies. Figure 4.17 depicts the resonant frequencies calculated via simulation. The difference between the hardware and software simulations is so negligible that this plot could be attributed to either a hardware or software implementation. The first three resonant frequencies were analytically predicted to be 1.4990, 2.9980 and 2.4969 GHz.

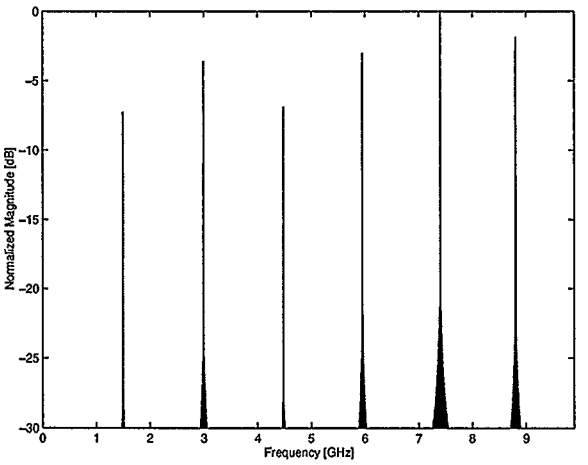


Figure 4.17: Resonant Frequencies of the One-Dimensional Resonator

The hardware-computed, one-dimensional resonator run successfully predicts the

first three resonant frequencies to within 0.024%, 0.164% and 0.396% of theoretical values. These results are very similar to the results produced using a traditional FDTD simulation programmed in C++, using 32-bit floating-point numbers, on an Intel-Linux computer. The accuracy of all three simulation methods are summarized in Table 4.4.

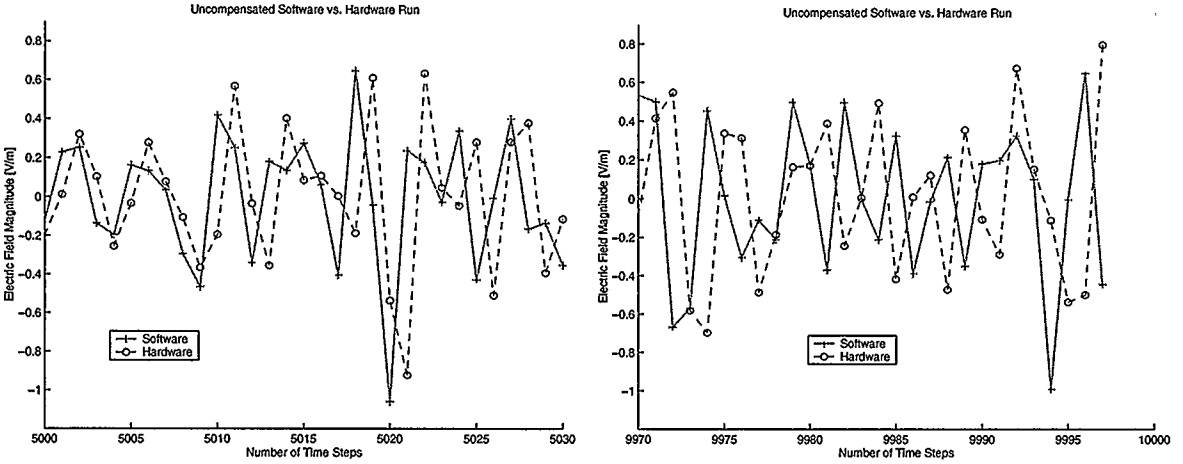
Table 4.4: Accuracy of the One-Dimensional Resonator Frequencies

	Resonant Frequency		
Simulation Method	Predicted (GHz)	Calculated (GHz)	% error
Software	1.4990	1.4985	-0.040
	2.9980	2.9932	-0.169
	4.4969	4.4796	-0.393
Hardware	1.4990	1.4986	-0.024
	2.9980	2.9930	-0.164
	4.4969	4.4791	-0.396
Software (adjusted Δt)	1.4990	1.4984	-0.040
	2.9980	2.9929	-0.168
	4.4969	4.4793	-0.393

If one compares the time domain data from the hardware and software platforms, segments of which are shown in Figure 4.18, some interesting results emerge.

Viewing the two curves after 5,000 time steps, two important observations are made. The behavior of the curves over time appear to be nearly identical, except that the hardware simulation curve leads the software simulation curve. By 10,000 time steps, the two curves appear to be quite different in behavior.

If one considers the difference between the two curves as a function of time, as in Figure 4.19, it is apparent that the error between the curves is very small at $t = 0$ but increases with time. By qualitative analysis of the data in Figure 4.18 it appears



(a) Simulation Data Near 5,000 Time Steps (b) Simulation Data Near 10,000 Time Steps

Figure 4.18: Comparison of Hardware vs. Software Simulation Data

that the two simulations lose time synchronization with respect to each other.

The loss of time synchronization between the hardware and software data points can be attributed to the quantization of the inductor and capacitor multiplier coefficients. In this specific case, the capacitor coefficient was normalized to 0.5 whereas the ideal inductor coefficient was determined, using a priori floating point calculations, to be 1.805. With a 12-bit multiplier coefficient, the best representation of this value is 1.8047 as shown in Equation 4.2. If the inductance remains the same, then the quantized coefficient actually has a different Δt than what was originally calculated. In fact, the new value of Δt should be

$$\Delta t_2 = \frac{X_{quant}}{X_{ideal}} \times \Delta t_1 \tag{4.3}$$

where X is the coefficient value. Although the difference between the ideal and quantized coefficients is relatively small, it is significant enough to represent two

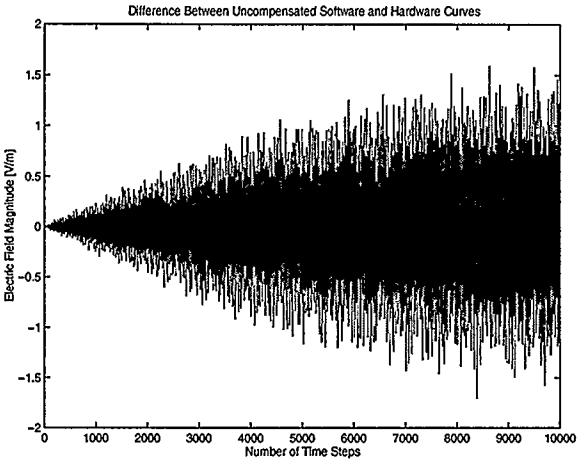


Figure 4.19: Difference Between Hardware and Software Simulation Curves

or more samples by the end of 10,000 time steps. This could be why the data in Figure 4.18 loses synchronization and the comparative error increases over time.

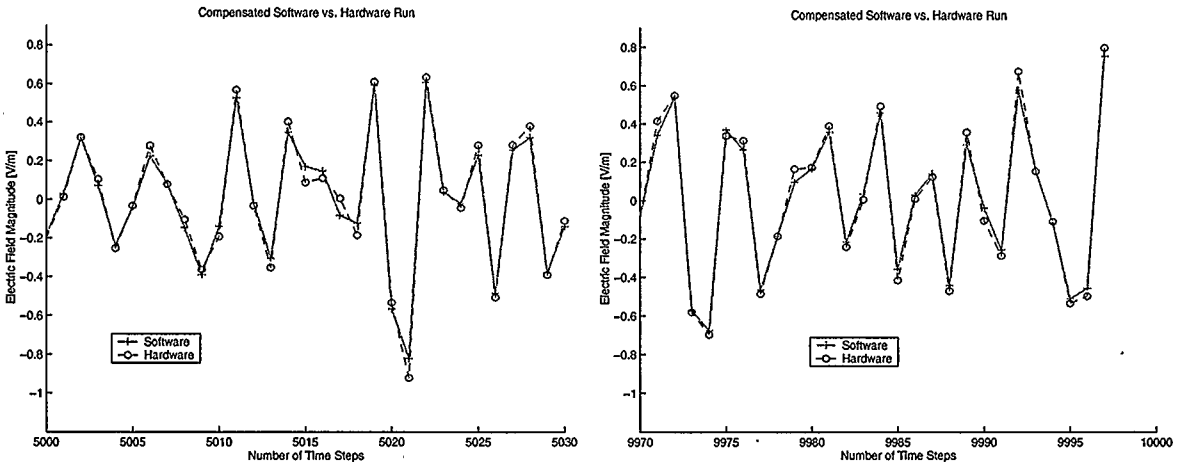
For this case, the value of X_{quant} is smaller than X_{ideal}, leading to a Δt that is smaller than what was originally calculated. This corresponds to the use of a larger stability margin (or smaller than maximum Δt). The only detrimental effect of such an approach is increased numerical dispersion at higher frequencies [9]. In some cases, however, the value of X_{quant} could be larger than X_{ideal}, resulting in a smaller stability margin.

Experimentally, the adjusted or compensated value of Δt is approximated to be

$$\Delta t_{adjusted} = \frac{\frac{X_{ideal} - X_{quant}}{2} + X_{quant}}{X_{ideal}} \times \Delta t_1 \tag{4.4}$$

which is the arithmetic average of Δt_1 and Δt_2. Although the hardware simulation data is fixed for a given resonator model, Δt and quantization, the software simulation is run with the value of $\Delta t_{adjusted}$. The results are depicted in Figure 4.20.

The new value of Δt is not that originally suggested in Equation 4.3 because it only considers one of the coefficients. In fact, because the capacitor coefficient is exact it still has the original calculated value of Δt_1. For the simulation model, half the coefficients use Δt_1 and the other half use Δt_2.



(a) Simulation Data Near 5,000 Time Steps (b) Simulation Data Near 10,000 Time Steps

Figure 4.20: Comparison of Hardware vs. Compensated Software Simulation Data

As can be seen from the plots, the hardware and software curves behave very similarly to each other. The difference between the curves for the uncompensated vs. compensated software runs is depicted in Figure 4.21. If the difference between the hardware and software platforms is now compared, the error magnitude no longer increases as a function of time. Some of the differences between the software and the hardware curves can now be attributed to the performance of the integer arithmetic and round-off error versus software floating-point. If more precise computations are performed by using larger integer registers and larger coefficient widths (or float-

ing point arithmetic), it is predicted that the difference between the hardware and software simulations will diminish altogether.

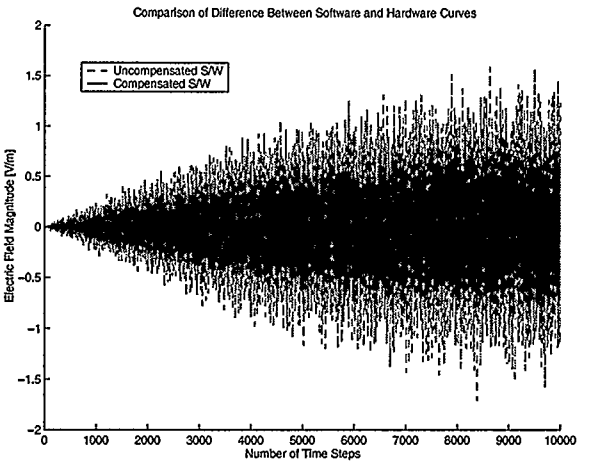


Figure 4.21: Difference Between Hardware and Software Simulation Curves with Compensated and Uncompensated Δt

The previous discussion illustrates two key points.

1. For the one-dimensional case, so far, it is possible to create a hardware simulation which is nearly identical to the software, floating-point simulation.

2. Coefficient quantization does affect the simulation data. In fact, the coefficient quantization has an effect not only on the magnitude of the results but also on the sampling interval. For this implementation, the quantized coefficients created an effective Δt which was different than the calculated value. It should be noted that this discrepancy does not affect the accuracy of the results, as shown in Table 4.4, only the comparison between the hardware and software implementations.

The simulation model represents a relatively simple case, with only two coeffi-

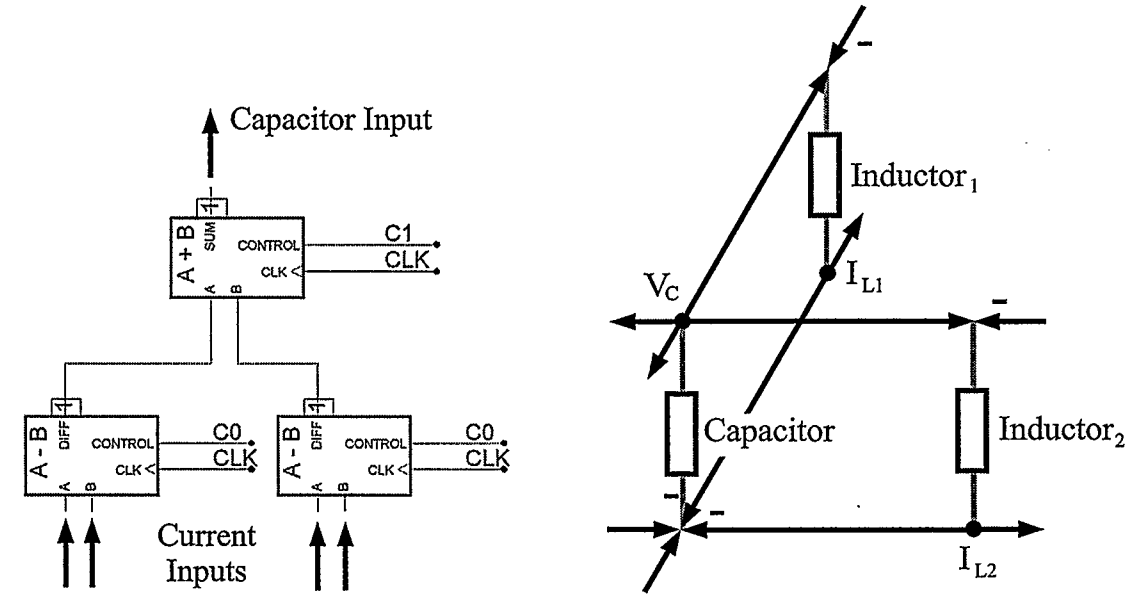
cients implemented on the hardware, 0.5 and 1.8047 respectively. It is predicted that with the use of more than one dielectric ($\varepsilon \geq 1.0$) or other electromagnetic material there will be several differently quantized coefficients, that will result in several different values of Δt in the hardware structure. For this case there appears to be no obvious solution, at this time, to analytically determine the actual Δt for the entire hardware simulation.

4.4.5 Two-Dimensional Resonator

Two-Dimensional FDTD Cell

For this case, an additional field is represented within the computational structure. A cell similar to the one-dimensional FDTD cell is constructed, with additional adder and subtractors circuits at the input of the capacitor (Figure 4.22(a)). Furthermore, an inductor is added to represent an additional magnetic field. With two magnetic fields (inductors) and an electric field (capacitor), this case represents the transverse-electric (TE) mode of propagation. The block diagram for the new cell is shown in Figure 4.22(b). This cell can be repeated in an $M \times N$ array to form a two-dimensional simulation space with N and M cells per side.

Originally, there was only one subtractor acting as an input to the capacitor structure, which coupled an inductor current from the adjacent cell. The new cell has two subtractors followed by an addition as the input to the capacitor. This allows for the input of two additional inductor currents, with the subtraction for sign convention, to be included in the capacitor voltage calculation. This extends the cell to two dimensions. It should be noted that there is also a complementary two-dimensional cell which would have one inductor and two capacitors to represent



(a) Modifications to Capacitor Input Chain

(b) Block Diagram

Figure 4.22: Two-Dimensional, Bit-Serial FDTD Cell

the two-dimensional FDTD. This complementary cell would represent the transverse-magnetic (TM) mode of propagation.

The system wordlength had to be increased in order to accommodate the additional computation elements in the data path. Referring to Figure 4.15 there are no spare delay elements in the primary data path. Adding computational elements in the primary data path causes the loop to have a delay larger than 32 bit-clock delays. To maintain byte-wise boundaries, which are mostly important for data collection and later processing, the system wordlength is increased to 40 bits.

Resonator Description

A two-dimensional resonator is constructed and terminated on one side with a perfect electric conductor (PEC) and perfect magnetic conductors (PMC's), also known as magnetic walls, on the remaining three sides. Figure 4.23 depicts the resulting structure.

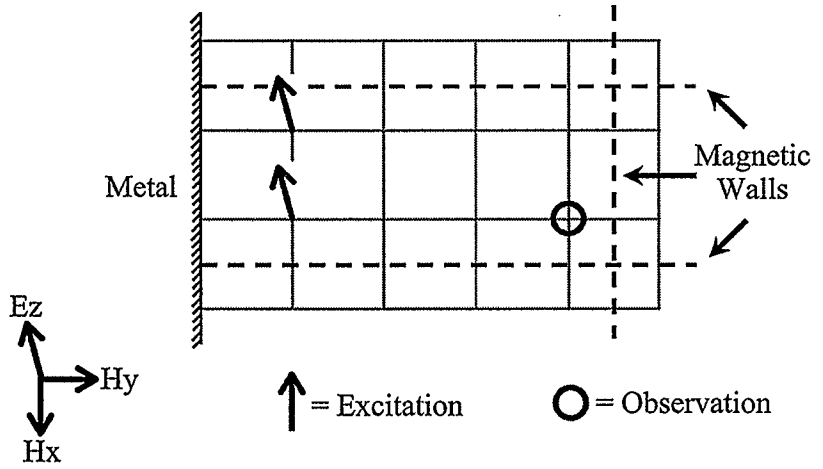


Figure 4.23: The Two-Dimensional Resonator

There was some concern about the amount of available hardware, so this structure is relatively small while still achieving useful results. The perfect magnetic conductor (PMC) (where $H_{tangential} = 0$) causes anti-symmetric behavior of the tangential magnetic fields adjacent to the boundary. Similarly, the PEC (where $E_{tangential} = 0$) causes anti-symmetric behavior of the tangential electric fields adjacent to the boundary. In order for the structure to appear longer, the magnetic wall (PMC) opposite the metal boundary (PEC) acts to reflect (copy) the entire structure. This provides more samples per wavelength, while retaining the compact size of the structure. Comparing it to the equivalent structure without the magnetic wall, this resonator can only support odd-order resonant frequencies.

Again, the excitation is a short, time-domain impulse to create a wideband excitation within the resonator. To discourage higher modes, the excitation is placed as a non-zero bias on a column of capacitors. This is designed to excite resonant energy only in the longer axis of the resonator.

Using 4.5 cells and then reflecting them yields a resonator 9.0 cm in length. The resonator is also 2.0 cm wide.

Choice of Coefficients

Given that $\varepsilon_r = \mu_r = 1.0$ and $\Delta x = \Delta y = 1.0cm$ the value of the capacitance is $C = 8.854 \times 10^{-13}F$ and the inductance is $L = 1.257 \times 10^{-9}H$. Finally, setting $\Delta x = \Delta y = 1.0cm$ and using a stability factor of 95% in the Courant condition yields a sampling interval of $\Delta t = 2.401 \times 10^{-11}s$.

The values of the coefficients are calculated as $\Delta t/C = 25.31s/F$ and $\Delta t/L = 0.0178s/H$. By performing impedance scaling (50.6138) the capacitor coefficient,

$\Delta t/C$, is normalized to 0.5 and the inductor coefficient, $\Delta t/L$, to 0.9025.

The capacitor coefficient (0.5) quantizes without error, but the inductor coefficient is represented as

$$\frac{924}{2^{12}} \times 4 = 0.90234375 \tag{4.5}$$

which includes a 12-bit multiply and the two left shift operators.

4.4.6 Two-Dimensional FDTD Speed and Hardware Utilization

Each two-dimensional FDTD cell requires 120 Virtex slices. Operating at a serial clock of 32 MHz and 40-bit SWL, new results are available every 1.25 microseconds (f_{op} = 0.8 MHz, 10,000 iterations = 12.5 milliseconds).

Comparison of Hardware versus Software Runtime

Again the goal of the hardware implementation is to achieve useful acceleration compared to the software implementation. Table 4.5 compares the hardware and software runtime. The software runtime is measured using a modified version of the simple C++ code (for two-dimensional FDTD) that was previously described in Chapter 3.

Table 4.5: Runtime for the Two-Dimensional Resonator Simulations

Simulation Method	Runtime† (ms)	Acceleration
Software	230	
Hardware	12.5	18.4X

†10,000 time steps

In this case, the acceleration is larger than reported for the one-dimensional resonator despite the penalty for the increased system wordlength. This is due to

processing gain achieved by the computation of more field components in parallel. Meanwhile, the software deals with the additional field calculations in a sequential manner.

As mentioned for the one-dimensional case, these results are exciting. Furthermore, using current FPGA technology the serial clock can be increased. In this case, the reported acceleration would be increased significantly as well.

There is a very important point that should be made with respect to this method of computation. Unlike the software, the runtime for 10,000 time steps is fixed even for larger simulations. For a fixed system wordlength, the hardware runtime remains constant regardless of simulation size; larger simulations require more hardware. The achievable acceleration is actually directly proportional to the number of cells implemented in the hardware, which in turn is directly related to the amount of hardware required.

4.4.7 Two-Dimensional FDTD Simulation Results

As for the one-dimensional resonator, three different simulations are run for the two-dimensional resonator case: a hardware simulation, a standard software simulation and then a software simulation with an adjusted time step.

All three simulation methods provided nearly identical accuracy in determining the resonant frequencies. Figure 4.24 depicts the resonant frequencies calculated via simulation. Once again, the difference between the hardware and software simulations is so negligible that this plot could be attributed to either a hardware or software implementation. The first two resonant frequencies were analytically predicted to be 1.6655 GHz and 4.9966 GHz.

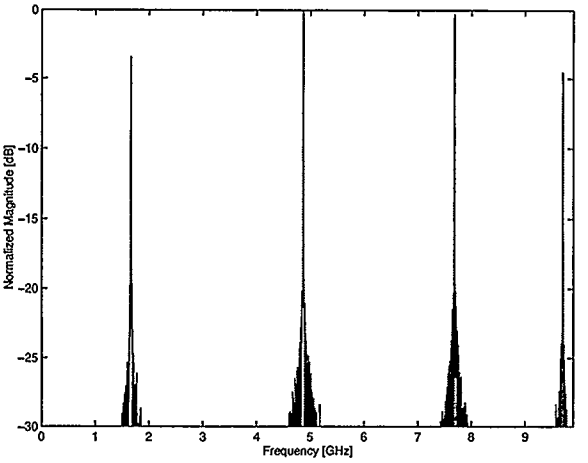


Figure 4.24: Resonant Frequencies of the Two-Dimensional Resonator

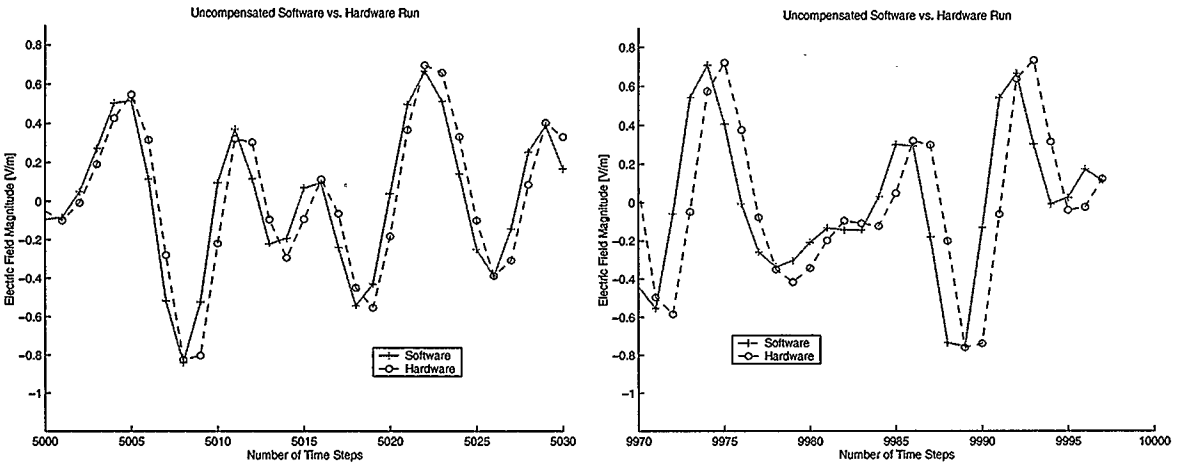
The hardware-computed, two-dimensional resonator run successfully predicts the first two resonant frequencies to within 0.269% and 2.61% of theoretical values. These results are very similar to the results produced using a traditional FDTD simulation programmed in C++, using 32-bit floating-point numbers, on an Intel-Linux computer. The accuracy of all three simulation methods are summarized in Table 4.6.

Table 4.6: Accuracy of the Two-Dimensional Resonator Frequencies

	Resonant Frequency		
Simulation Method	Predicted (GHz)	Calculated (GHz)	% error
Software	1.6655	1.6609	-0.278
	4.9966	4.8659	-2.615
Hardware	1.6655	1.6610	-0.269
	4.9966	4.8660	-2.614
Software (adjusted Δt)	1.6655	1.6609	-0.280
	4.9966	4.8660	-2.614

If one compares the time domain data from the hardware and software platforms,

segments of which are shown in Figure 4.25, the sampling interval skew (discussed earlier) is evident once again. In this case, the magnitude of the inductor's coefficient quantization error is less, so the resulting time skew errors are also less. Even after 10,000 time steps, the behavior of the curves is still quite similar.



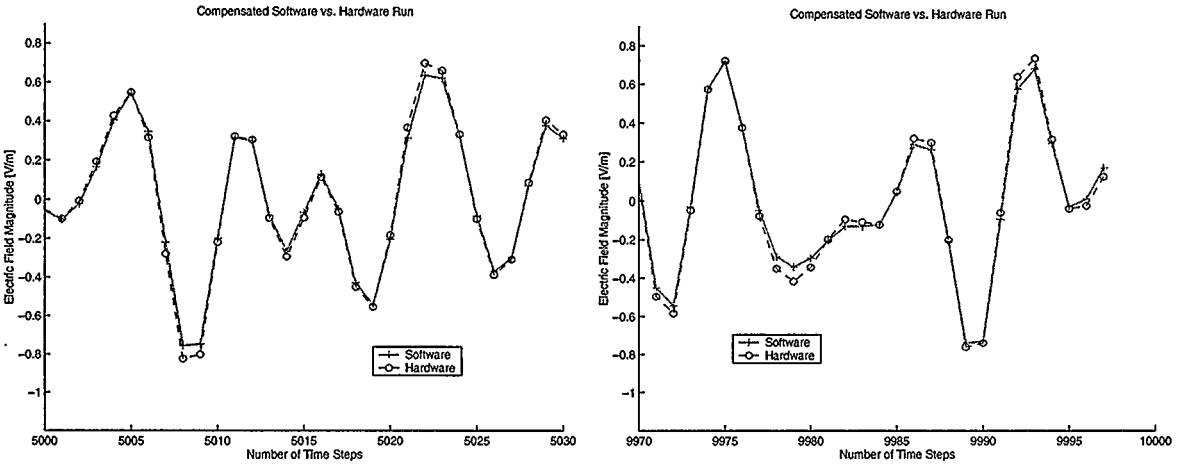
(a) Simulation Data Near 5,000 Time Steps (b) Simulation Data Near 10,000 Time Steps

Figure 4.25: Comparison of Hardware vs. Software (2D) Simulation Data

Using the same formula, Equation 4.4 from the one-dimensional case, to calculate $\Delta t_{adjusted}$ the software simulation is run again. Figure 4.26 depicts the new time domain data for the software and hardware methods.

As can be seen from the plots, the hardware and software curves behave in a very similar manner. The difference between the curves for the uncompensated vs. compensated software runs is depicted in Figure 4.27.

Once again, two observations can be made. First, this method is successful at implementing a two-dimensional, hardware-based FDTD cell, which produces



(a) Simulation Data Near 5,000 Time Steps (b) Simulation Data Near 10,000 Time Steps

Figure 4.26: Comparison of Hardware vs. Compensated Software (2D) Simulation Data

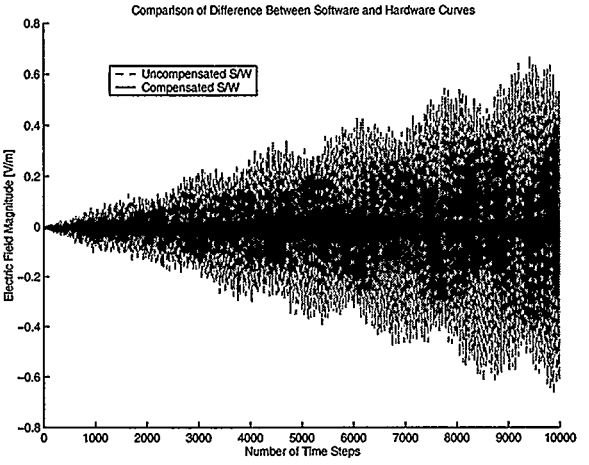


Figure 4.27: Difference Between Hardware and Software (2D) Simulation Curves with Compensated and Uncompensated Δt

both accurate FDTD simulation results and results that are nearly identical to the software, floating-point implementation. Second, the adjusted time calculation is really only valid for this simplified structure with only two sets of coefficients. With varying materials and multiple coefficients, it is expected that the time skew will be difficult to predict in analytical manner.

4.4.8 Three-Dimensional FDTD?

The one-dimensional cell represents two-fields in 86.5 Virtex slices. A two-dimensional cell represents three fields in 120 Virtex slices. Finally, it is estimated that a three-dimensional cell would cost 265 slices, representing 6 fields.

Simple calculations quickly show that a 100x100x100 simulation (one million cells) is too large to fit on the largest FPGA currently available. The Virtex-2 10000 series part offers 61,440 slices. It would take 4,313 of these parts to implement the entire simulation. From another perspective, the largest FPGA parts represent 10 million logic gate equivalents. Thus, for one million cells there are only 10 logic gate equivalents available per FDTD cell, per FPGA. Clearly, the presented design is not economically or technologically feasible for a reasonably-sized three-dimensional simulation.

In order to yield a useful acceleration of FDTD, in a shorter developmental time frame, other avenues are explored. The next chapter addresses a resource-sharing approach.

4.5 Summary of Results

Table 4.7 provides the results obtained for the FDTD hardware implementation using three metrics: (i) hardware size/cost, (ii) computation speed (acceleration) and (iii) simulation accuracy.

Table 4.7: Summary of Performance for the Bit-Serial Implementation of FDTD

Type of Cell	Cell Size (Virtex Slices)	Acceleration over Software	Simulation Prediction Error
One-Dimensional	86.5	8.4X	0.024-0.39%
Two-Dimensional	120	18.4X	0.27-2.61%
Three-Dimensional	265†		

†Estimated

There are a number of important results and observations from this phase of the research. These include:

- The FDTD algorithm, in one and two dimensions, is successfully implemented in hardware.
- The acceleration achieved for two-dimensional computations is very promising and well within the range of the thesis goals, despite the use of an older generation FPGA. It is anticipated that the reported accelerations can be easily improved by using newer technology.
- The implementations for both one- and two-dimensional FDTD computations

yield accurate simulation results that are virtually identical to a software implementation.

- The runtime for a fixed system-wordlength and fixed number of time steps is constant; thus, the more hardware available, the higher the number of cells computed in parallel and the higher the acceleration achieved.

4.6 Applications

The results reported in this chapter are useful for accelerating current FDTD applications. A reasonably-sized three-dimensional FDTD simulation, implemented using the techniques discussed in this chapter, would be too large to fit on a practical number of FPGA's. However, the one-dimensional and two-dimensional implementations could support reasonably sized simulations.

Given that the largest available Xilinx Virtex device offers 61,440 slices, this device could implement either 512 two-dimensional or 710 one-dimensional FDTD cells. Results from this chapter have already shown that the hardware FDTD provides significant acceleration compared to the software on the reference computer. Hundred of cells in parallel, compared to the previous experiments' 10 cells, would offer even greater acceleration.

Ongoing research predicts that the hardware size of the one- and two-dimensional computational cells could be reduced further. As well, several of the largest FPGA's could be coupled together. In this instance, simulations with thousands of one- or two-dimensional cells could be computed.

There are FDTD problems, currently solved using software implementations,

that could make use of this research. Certain three-dimensional models with rotational symmetry can be represented as modified two-dimensional models. Furthermore, mode solvers typically seek eigenfunctions in a two-dimensional cross-section of waveguide structures. This is actually a two-dimensional problem. Applying the techniques described in this chapter would yield significant accelerations for these and other FDTD applications.

Chapter 5

The Hardware Accelerator: An FDTD Co-Processor

"One of the main causes of the fall of the Roman Empire was that, lacking zero, they had no way to indicate successful termination of their C programs." - Robert Firth

This chapter describes two concepts: (i) the concept of a resource-sharing approach for computing FDTD and (ii) the envisioned specification, design and implementation for an FDTD hardware accelerator. This phase of the research is in the preliminary stages.

The first section describes the approach of both resource-sharing and the FDTD hardware accelerator. This is followed by a discussion of the technical challenges posed by this phase of the research.

In the third section, the design space is explored and ideas are described for achieving the desired order of magnitude acceleration. Finally, a simple plan is presented for realizing the long term goal of the research, an FDTD hardware accelerator.

5.1 Description of the Approach

The results for Chapter 4 are very promising for one- and two-dimensional hardware FDTD. Sadly, the hardware requirements for a reasonably-sized three-dimensional FDTD simulation are simply too large. The implementation in Chapter 4, however, approaches maximum possible parallelism[1] by implementing every computational cell in hardware. The approach proposed in this chapter is to reuse a smaller set of hardware resources to compute the field updates for an arbitrarily-sized three-dimensional mesh.

The results presented in Chapter 4 suggest that significant acceleration, as much as several orders of magnitude, can be achieved by computing a number of FDTD update equations (computational cells) in parallel. It is anticipated that these large accelerations can be traded for smaller hardware designs that fit on a practical number of FPGA's and still accelerate FDTD by more than an order of magnitude.

Similar to the bit-parallel vs. bit-serial implementation paradigms, the resource-sharing can take on many forms. At one end of the spectrum the computational hardware may contain only enough hardware resources to compute a single update equation at a time. In this case, a single set of input data values (field values and coefficients) are loaded; the field update is computed and then stored. This process is repeated for all of the required field updates. This is very similar to the way in which the FDTD software algorithm is computed on a single CPU. At the opposite end of the spectrum, no resources are shared at all. This is the implementation described in Chapter 4, but it requires too much hardware. In the middle of this

[1]The design would further approach maximum possible parallelism using a bit-parallel instead of bit-serial paradigm.

spectrum, the computational hardware can compute a small sub-volume or plane of field updates. In this case, the sub-volume or plane is loaded with input data, the results are computed and then stored. The sub-volume/plane is reused until all field updates have been performed. The size of the sub-volume/plane is determined by the available hardware resources and the rate at which the input data can be loaded into the structure.

The investigation of a resource-sharing implementation is combined with the development of the specifications for an FDTD hardware accelerator. The research presented in the previous chapters indicates that a successful FDTD accelerator will act as a "co-processor" in a host computer.

5.2 Technical Challenges

As noted, the size of a three-dimensional mesh of FDTD cells prohibits an entire simulation from being implemented on a practical amount of hardware. The shift to full reuse of the computational hardware creates different technical challenges and requirements than the "FDTD Computational Engine" of the previous chapter. Furthermore, in keeping with one of the goals of the research, the hardware accelerator should be able to accelerate *existing* software implementations. Four important challenges for the proposed accelerator design are described below:

1. External Memory – Field components and coefficients that are waiting to be processed must be stored until they are needed. This will require some form of external memory (accessible by the accelerator), for sufficiently large simulation models.

2. Memory Management – Field components and update equation coefficients must be loaded into the accelerator's computational resources. Then, the field updates must be computed and stored again.

3. Accelerator and Host Computer Interaction – Simulation data must be exchanged on a regular basis between the host computer and the hardware accelerator.

4. Software Modifications – Performing the FDTD computations on the hardware accelerator and exchanging data back and forth will require changes to the *existing* software implementations.

The first two challenges come from the resource sharing approach. The remaining two challenges are due to the requirements of the hardware accelerator. It is assumed that the challenge of implementing the FDTD update equations using integer arithmetic is already solved by the two-dimensional FDTD cell presented in Chapter 4.

The following section describes the characteristics of the hardware accelerator to meet the four challenges.

5.3 Overview of the Hardware Accelerator

The FDTD accelerator is envisioned as one or more hardware cards that are inserted into the host PC. FDTD computations will be performed by the accelerator card(s) while interacting with the host computer's CPU and memory. For now, the chosen hardware interface is the PCI bus. Figure 5.1 depicts the potential application of

the hardware accelerator.

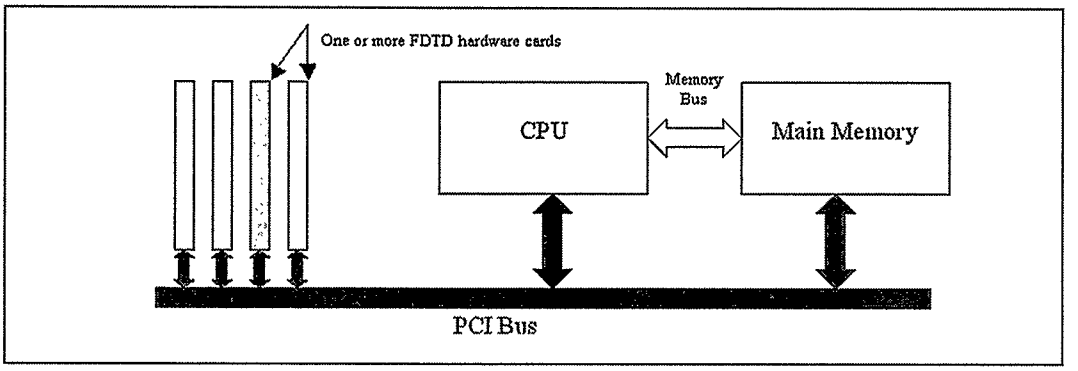


Figure 5.1: Potential Application of the Hardware Accelerator Inside the Host Computer

PCI bus technology is chosen over AGP for several reasons. There is a large degree of support for PCI-based devices, both from operating systems and hardware board manufacturers. There are numerous products available with PCI interfaces and FPGA devices that could be used as a prototype hardware accelerator. On the other hand, AGP is very specific to video cards and therefore offers little support for other uses. Although not confirmed, it is believed that the AGP standard is highly optimized for one-way communication of data. While a tremendous amount of data is sent to the video processor from the host computer's memory and CPU, very little is sent back. That leaves the PCI bus as the only choice.

The communications bus between the hardware accelerator and the CPU is viewed as a potential bottleneck. To reduce the communication on this bus, it is decided that the hardware accelerator should have a large amount of onboard memory. Section 3.3.1 described the asymmetrical input/output requirements of the FDTD computations. While nine values are loaded for each calculation of the update equation, only one value is returned. Communicating the result of field updates requires

only about 10-11% of the input bandwidth. For constant material coefficients, it also makes sense to send this data only once to the hardware accelerator. Thus, the FDTD simulation data is pushed as close to the hardware accelerator as possible by using onboard memory.

Another justification for the above decision arises when one compares the predicted bandwidth requirement for the hardware accelerator, that is presented in Chapter 3, to the available bandwidths of the PCI and AGP buses. The required bandwidth of 5397 MB/s between the host computer and the hardware accelerator is presently not possible. Thus, the data must be moved closer to the accelerator where larger bandwidth can be achieved. It is expected that the onboard memory will be connected to the hardware via several channels, in order to increase the available memory bandwidth. The design considerations for achieving the required memory bandwidth are discussed in Section 5.4.2.

In summary, Figure 5.2 depicts the envisioned hardware accelerator card:

The next section describes more details about the digital design that is envisioned in order to achieve an order of magnitude acceleration.

5.4 Design Choices for the Hardware Accelerator Implementation

This section describes the design space for implementing the hardware accelerator described in the previous section. At the end of Chapter 3 the required computation speed and memory bandwidth for an order of magnitude acceleration are presented. The hardware accelerator will need to compute field updates at approxi-

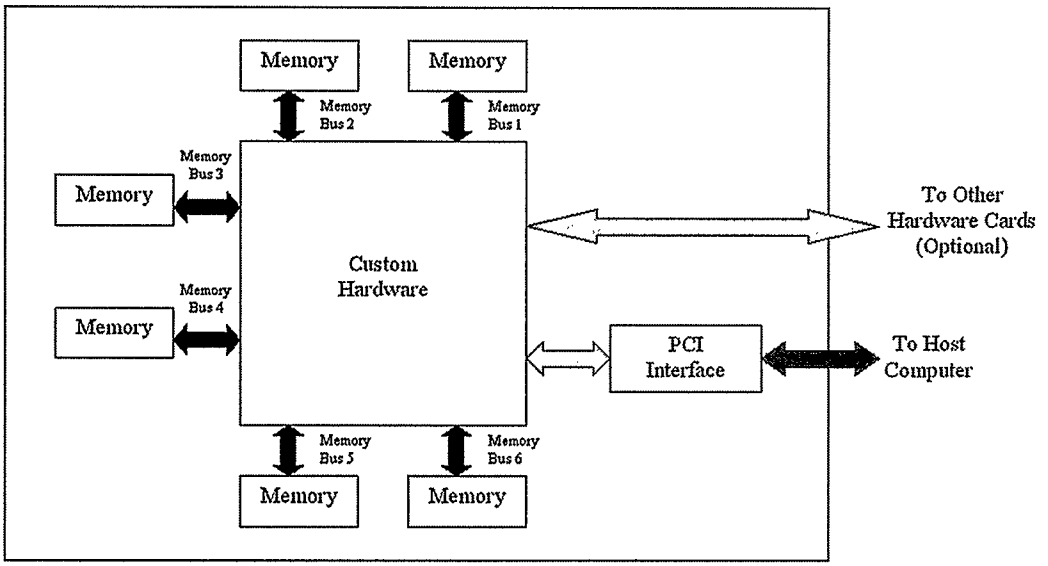


Figure 5.2: Envisioned Hardware Accelerator Card

mately $f_{comp} = 140 - 150$ MHz. Furthermore, it would require a memory bandwidth of 5397 MB/s. Finally, it is assumed that the hardware accelerator will use either 64-bit PCI at 66MHz or the PCI-X standard.

The potential design choices for achieving the desired computation speed and input memory bandwidth are described in the following sections. This is followed by the design choices available to minimize the data communication requirements between the accelerator and the host PC.

5.4.1 Achieving the Desired Computation Speed

A number of statements should be made about the preceding requirements for the hardware accelerator. The computation speed requirement, of $f_{comp} = 140 - 150\ MHz$, could be satisfied through a few different (non-exclusive) approaches:

- A brute force approach could be used to implement a single update equation that generates new field updates at 150 MHz.

- A pipelined approach could allow for the computations of several update equations to be overlapped. This would make more efficient use of some hardware resources if they were otherwise idle. The net benefit is that the parallelization of some of the equations could allow for a slower clock speed.

- A parallel approach could use a number of slower computational units to achieve the same computation speed as a single, very fast unit.

5.4.2 Achieving the Desired Memory Bandwidth

Several design choices are also available to achieve the required memory bandwidth. It should be noted that this requirement contains the assumption of 32-bit numbers for all values in the update equation. Larger or smaller widths would change this value. Each coefficient and field value do not necessarily require the same precision.

The desired memory bandwidth of 5397 MB/s could be achieved with a combination of the following choices:

- Data values (fields, coefficients) could be compressed before being stored and uncompressed after being fetched. This would require additional computations but may be required if memory bandwidth limits the acceleration.

- Data values can share a memory fetch. As an example, field components are fixed at 48-bits wide. These could be stored with their update coefficient of 16-bits in the same 64-bit memory location.

- Multiple memory buses could be used to divide the bandwidth requirement across many slower memory banks. Given a memory technology for the reference computer, 10 memory banks would satisfy the corresponding bandwidth requirement. However, other factors like number of input/output pins, economics and technical challenges may prevent this brute force solution. As well, the FDTD simulation data may not partition into ten memory banks in a useful way.

- Caching on the hardware device could be used to either (i) reuse previously fetched data and (ii) ensure that the maximum possible memory bandwidth is achieved for a given technology.

- Similar to the above point, field updates for a cross-section or sub-volume of the FDTD mesh could be computed at the same time; thus, overlapping field values between adjacent cells could be reused.

- The number of values required to complete a field update could be reduced. One radical proposal is to compute all of the field updates in the mesh for a fixed material or very small set of materials. In this case, none of the material coefficients would require loading. In order to accommodate other materials, the default update equations would be reversed (by the host computer) and re-computed with the proper coefficients. Depending on the ratio of ambient (common) material to other materials, this can yield a significant performance gain while reducing the required bandwidth.

5.4.3 Achieving the Desired Memory Bandwidth Between the Hardware Accelerator and the Host Computer

As noted earlier, the communications bandwidth between the hardware accelerator and the host computer is perceived as a potential bottleneck. The optimal situation occurs when only the data that is specifically required by either the host computer or the hardware card is transmitted on the bus. Several techniques can be used to minimize the communications bandwidth required:

- For simulations with constant material properties for the entire simulation, only the observation and excitation information is exchanged between the host computer and the hardware accelerator.
- For more advanced simulations, where certain fields may need to be re-computed by the host computer, only the minimum number of fields (a sub-volume or plane) are exchanged.
- The data could be compressed for transmission across the bus.

5.4.4 Discussion of the Design Choices

Ultimately, the selection of design choices will affect the performance of the hardware accelerator and the resource-sharing approach. It is suggested that most of the preceding design choices are investigated as the next phase of the accelerator development. It is worth noting that in order to investigate many of the design choices, new hardware platforms must be procured or designed.

The following section describes the suggested hardware platforms for the continuation of this phase of the research.

5.5 Future Generations of Hardware Accelerators

This section is intended as a "plan" to describe the path from the current hardware platform to the final hardware accelerator. It is anticipated that there will be three hardware platforms used for this research: (i) the current XESS development board, (ii) an incoming hardware board from the Dini Group and (iii) a fully custom hardware card. The application of each board, to the investigation of the technical challenges of an FDTD Co-processor and hardware accelerator, is discussed in the following sections.

5.5.1 The XESS Board

This hardware development board is the same one used to implement and verify the FDTD simulations in Chapter 4. This board does not support a PCI connection or any sufficiently fast hardware interface. Currently, data is exchanged between the host computer and the FPGA via the parallel port.

The FPGA device is somewhat older than current technology and does not offer a large amount of hardware resources. Nevertheless, this platform is currently being used to develop and verify a resource-sharing approach for FDTD computations.

There are four 8-bit by 512K static RAM memories on board. Thus, several key concepts can be investigated using this platform, including:

1. A small-sized resource-sharing approach for a size-limited, three-dimensional FDTD simulation. This will provide a reference for the performance that could be achieved by moving to faster FPGA technology.

2. The control structures required to coordinate the fetch, computation and stor-

age sequence for the field update equations. It is desirable that these control structures are scalable to larger and smaller hardware implementations (for the resource-sharing structure).

3. A memory addressing scheme for storing the three-dimensional FDTD mesh and related coefficients in linear memory. Preliminary research shows that the memory management and address generation represents a significant challenge.

It is the goal that a scalable architecture is constructed that can later be applied and extended to the future hardware accelerators.

5.5.2 The Dini Group Board

In order to investigate solutions to the some of challenges in Section 5.2 a new hardware platform is required. For now, this platform should have (i) external memory, (ii) a PCI-interface and (iii) a large FPGA device.

The chosen board is the Dini Group and provides two Xilinx Virtex2 series FPGA's, a PC133 SDRAM memory bank and a PCI/PCI-X hardware interface.

This development board will allow for the investigation/development of some very important concepts with respect to the final hardware accelerator design.

The PCI support will allow for the development of a software interface between *existing* software implementations and the accelerator. At this point, it is anticipated that the hardware will not provide an order of magnitude acceleration due to memory bandwidth limitations. However, the accelerator hardware will be coupled to the FDTD software, which represents an important milestone.

The onboard PC133 SDRAM will allow for (i) the development/improvement of

an SDRAM memory controller and (ii) investigation into the maximum achievable memory bandwidth. The memory controller will be an important component of any future designs. The scalable architecture and control logic, described in the previous section, may change to facilitate better memory performance.

At this stage of the research, sufficient knowledge should be available to design and implement a full custom hardware accelerator board.

5.5.3 Full-Custom Board

This hardware platform is envisioned as a "product" available to FDTD users in academia and industry that would provide the order of magnitude acceleration to the existing FDTD software. It is anticipated that this design will make use of multiple memory banks and the latest hardware bus technology in order to succeed. It is also projected that the hardware board will involve a custom PCB design and (potentially) a custom silicon implementation.

Achieving the desired level of acceleration and making it accessible to FDTD users will open up many new opportunities for research.

Chapter 6

Conclusions

"People on Jolt cola write the funniest things."

- A-10 Obedience Guide, Kitty Hawk Studios

This chapter summarizes the accomplishments of the thesis and relates them to the objectives defined in the first chapter. This discussion is followed by a description of the envisioned future work.

6.1 Accomplishments of the Thesis

The primary achievement of this thesis was to successfully confirm the potential for a custom hardware implementation to significantly accelerate the FDTD algorithm. This was achieved by using programmable hardware, integer arithmetic, and fine-grained parallelism.

The following accomplishments have been demonstrated:

- Custom hardware accelerates the computation of the FDTD algorithm by as much as 18.4 times.
- Hardware FDTD produces virtually identical time domain results and modelling accuracy as floating-point, software implementations.
- Pipelined bit-serial arithmetic successfully implements fixed-precision computations for the FDTD algorithm.

6.2 Future Work

The goals achieved in the thesis provide a foundation for many areas of future work. This is a large project with many milestones and potential avenues for new research. Four main topics are discussed in relation to future work: improvements to the existing work, the next generation of hardware accelerators, research using hardware accelerators and possible extensions of the current research.

6.2.1 Improvements to the Existing Work

One of the primary improvements to the existing work would be to implement the presented designs in VHDL. Currently, all the digital designs provided are entered using schematic capture tools. Using VHDL would allow for parameterizable designs for the multiplier coefficient widths, the system wordlengths and the control structures. It is envisioned that a higher level software program could automatically generate pipelined bit-serial implementations from a core VHDL library of (parameterizable) bit-serial operators.

These parameterizable designs could be extended to an investigation of the stability of the FDTD algorithm on hardware, accounting for finite-precision effects. The stability and accuracy of the current implementations need to be investigated and quantified from an analytical perspective. It is desirable to have a deterministic evaluation of quantized coefficients to ensure/predict the stability of a hardware-based FDTD simulation.

Furthermore, an investigation of the necessary coefficient and field component precision (bit width) would be valuable.

6.2.2 Future Generations of the Hardware Accelerator

Several FDTD hardware accelerator prototypes are described in the previous chapter. Some preliminary research into a resource-sharing approach has been completed. It is desirable that this work is continued to produce a scalable architecture that could be applied to future accelerator designs.

6.2.3 Research Following a Successful Hardware Accelerator

There are a few research problems that are of interest once significant acceleration is possible.

First, it is desirable to compare the overall performance of the accelerated FDTD algorithm to two competitors: the traditional software, floating-point algorithms and other electromagnetic modelling methods. The numerical methods community has a number of standard electromagnetic problems which are used to evaluate the performance (speed, accuracy) of various computational methods. It is suggested that these standardized models be compared for accuracy and speed among the hardware FDTD algorithm, software FDTD and other computational methods.

Second, accurate absorbing boundary conditions like PML's can account for a large portion of the runtime of advanced FDTD simulations. It is definitely a priority to accelerate these computations in the same manner as the update equations.

Third, the idea of optimization-over-FDTD is seldom pursued because simulations just take too long. With significant acceleration, optimization algorithms specific to FDTD could be investigated and implemented.

6.2.4 Extensions of the Current Research

On the long term horizon, the hardware acceleration concept and the knowledge/experience gained from this research may be applicable to other numerical methods in both electromagnetics and other fields.

References

[1] K. S. Yee, "Numerical solution of initial boundary value problems solving Maxwell's equations in isotropic media," *IEEE Trans. Antennas and Propagation*, vol. 14, pp. 302–307, 1966.

[2] A. Taflove, *Advances in Computational Electrodynamics - The Finite Difference Time Domain Method*, Artech House Inc., Norwood, MA, 1998.

[3] M. M. Okoniewski, ," Personal Communication, 2002.

[4] G.A. Schiavone, I. Codreanu, and R. Panaiappan R. Wahid, "FDTD speedups obtained in distributed computing on a linux workstation cluster," in *IEEE Antennas and Propagation Society International Symposium*, Salt Lake City, UT, 2000, vol. 3, pp. 1336–1339.

[5] K.D. Tatalias and J.M. Bornholdt, "Mapping electromagnetic field computations to parallel processors," *IEEE Transactions on Magnetics*, vol. 25, no. 4, pp. 2901–2906, 1989.

[6] C.J. Gillan and V. Fusco, "Optimizing FDTD electromagnetic field computation on distributed networks," *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, vol. 11, no. 6, pp. 277–287, 1998.

[7] C.J. Gillan and V. Fusco, "Enhanced distributed computing message passing strategies for FDTD," *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, vol. 12, no. 6, pp. 483–488, 1999.

[8] K. Kunz and R. Luebbers, *The Finite Difference Time Domain for Electromagnetics*, CRC Press Catalog Number 8657, 1993, 496 pages.

[9] A. Taflove, *Computational Electrodynamics - The Finite Difference Time-Domain Method*, Artech House Inc., Norwood, MA, 1996.

[10] S.V. Grinin, "Integer-arithmetic FDTD codes for computer simulation of internal, near and far electromagnetic fields scattered by three-dimensional conductive complicated form bodies," *Computer Physics Communications*, vol. 102, no. 1–3, pp. 109–131, May 1997.

[11] J.R. Marek, M.A. Mehalic, and A.J. Terzuoli, "A dedicated VLSI architecture for finite-difference time domain calculations," in *8th Annual Review of Progress in Applied Computational Electromagnetics*, Monterey, CA, March 1992, vol. 1, pp. 546–553.

[12] P. Placidi, L. Verducci, G. Matrella, L. Roselli, and P. Ciampolini, "A custom VLSI architecture for the solution of FDTD equations," *IEICE Trans. Electron.*, vol. E85-C, no. 3, pp. 572–577, March 2002.

[13] IEEE, *IEEE Standard 1076: IEEE Standard VHDL Reference Manual*, 1987.

[14] M.M. Mano, *Digital Design (Second Edition)*, Prentice-Hall, Englewood Cliffs, NJ, second edition, 1991.

[15] J.F. Wakerly, *Digital Design: Principles and Practices*, Prentice Hall, Englewood Cliffs, NJ, second edition, 1994.

[16] R. Andraka, "Andraka consulting group, inc.," Website: http://www.andraka.com/.

[17] Xilinx Inc., *Xilinx Data Book: Virtex 2.5V Field Programmable Gate Arrays*, September 2002.

[18] XESS Development Corp., ," http://www.xess.com.

[19] R.I. Hartley and K.K. Parhi, *Digit-Serial Computation*, Kluwer Academic Publishers, Norwell, Massachusetts, 1995.

[20] P.B. Denyer and D. Renshaw, *VLSI Signal Processing - A Bit-Serial Approach*, Addison-Wesley, Reading, MA, 1985.

[21] R.M.M. Oberman, *Digital Circuits for Binary Arithmetic*, The Macmillan Press Ltd., 1979.

[22] Denis Howe, "The free online dictionary of computing," Website http://burks.brighton.ac.uk/burks/foldoc/17/16.htm, 2002.

[23] E. Fear, ," Personal Communication, 2001.

[24] W.K. Gwarek, "Analysis of arbitrarily shaped two-dimensional microwave circuits by finite-difference time-domain method," *IEEE Trans. Microwave Theory and Techniques*, vol. 36, no. 4, pp. 738–744, April 1988.

[25] W.K. Gwarek, "Analysis of an arbitrarily-shaped planar circuit - a time domain approach," *IEEE Trans. Microwave Theory and Techniques*, vol. MTT-33, no. 10, pp. 1067–1072, October 1985.

[26] L.T. Bruton, "Low sensitivity digital ladder filters," *IEEE. Trans. Circuits Syst.*, vol. CAS-22, pp. 168–176, March 1975.

[27] A. Fettweis, "On the connection between multiplier wordlength and roundoff noise in digital filters," *IEEE Trans. on Circuit Theory*, vol. 19, no. 5, pp. 486–491, September 1972.

[28] "SmartDIMM," Website: http://www.cedcc.psu.edu/smartdimm/.

[29] H. Monson, "RAM technology primer: CAS latency," Web: http://www.sysopt.com/articles/latency/, September 1999.

[30] "PC133 unbuffered DIMM system timing and design," Tech. Rep., Micron, http://www.via.com.tw/pdf/presentations/9904pc133/9904micron1.pdf, April 1999.

[31] Intel Corp., "Accelerated graphics port technology," Web: http://www.intel.com/technology/agp/.

[32] M. M. Okoniewski, "Totem: Electromagnetic field simulator," Website http://www.enel.ucalgary.ca/ michal/totem, 2002.

[33] Intel Corp., "Intel high-performance compilers," Website: http://www.intel.com/.

[34] P. Alfke, "Efficient shift registers, LSFR counters and long pseudo-random sequence generators," Xilinx Application Note XAPPP 052, Xilinx Inc., 1996, http://www.xilinx.com.

[35] W. Cheney and D. Kincaid, *Numerical Mathematics and Computing*, Brooks/Cole Publishing Company, third edition, 1994.

Appendix A

Derivation of the Basic FDTD Algorithm

The finite-difference time-domain (FDTD) algorithm is an approximation to Maxwell's equations that is well suited for implementation on a computer. Maxwell's equations are the classical physics equations used to mathematically describe dynamic electromagnetic behavior.

A.1 Maxwell's Equations in Three Dimensions

Maxwell's equations mathematically describe the dynamic behavior of electric and magnetic fields. In differential form they are stated as:

Faraday's Law:

$$\frac{\partial \vec{B}}{\partial t} = -\nabla \times \vec{E} - \vec{J}_m \tag{A.1}$$

Ampere's Law:

$$\frac{\partial \vec{D}}{\partial t} = \nabla \times \vec{H} - \vec{J}_e \tag{A.2}$$

Gauss's Law for the electric field:

$$\nabla \cdot \vec{D} = \rho \tag{A.3}$$

Gauss's Law for the magnetic field:

$$\nabla \cdot \vec{B} = 0 \tag{A.4}$$

Where t = time $[s]$

$\vec{E}$ = electric field $[V/m]$

$\vec{D}$ = electric flux density $[C/m^2]$

$\vec{H}$ = magnetic field $[A/m]$

$\vec{B}$ = magnetic flux density $[Wb/m^2]$

$\vec{J}_e$ = electric current conduction density $[A/m^2]$

$\vec{J}_m$ = equivalent magnetic conduction current density in $[V/m^2]$

ρ = volume charge density $[C/m^3]$

Maxwell's equations can also be expressed in integral form, see [] for an example.

$\vec{B}$ and $\vec{H}$ are simply related by following constitutive equations:

$$\vec{B} = \mu\vec{H} \tag{A.5}$$

and $\vec{D}$ and $\vec{E}$ by:

$$\vec{D} = \varepsilon\vec{E} \tag{A.6}$$

Where μ = magnetic permeability $[H/m]$

ε = electric permittivity $[F/m]$

For linear, isotropic, and non-dispersive materials (whose properties are independent of field-intensity, direction or frequency) ε and μ are scalar quantities.

Furthermore, $\vec{J}_m$ and $\vec{J}_e$ are expressed as magnetic and electric losses, which dissipate the electromagnetic field energy as heat. These losses are defined as:

$$\vec{J}_m = \rho' \vec{H} \tag{A.7}$$

$$\vec{J}_e = \sigma \vec{E} \tag{A.8}$$

Where ρ' = magnetic resistivity $[\Omega/m]$.

σ = electric conductance $[\mho/m]$

Substituting Equations A.5 to A.8 into Equations A.1 to A.2, under the assumptions of linear, isotropic and non-dispersive materials, yields:

$$\frac{\partial \vec{H}}{\partial t} = \frac{1}{\mu} \cdot \left(-\nabla \times \vec{E} - \rho' \vec{H} \right) \tag{A.9}$$

$$\frac{\partial \vec{E}}{\partial t} = \frac{1}{\varepsilon} \cdot \left(\nabla \times \vec{H} - \sigma \vec{E} \right) \tag{A.10}$$

For the lossless case, Equations A.9 to A.10 further simplify to:

$$\frac{\partial \vec{H}}{\partial t} = \frac{1}{\mu} \cdot \left(-\nabla \times \vec{E} \right) \tag{A.11}$$

$$\frac{\partial \vec{E}}{\partial t} = \frac{1}{\varepsilon} \cdot \left(\nabla \times \vec{H} \right) \tag{A.12}$$

Equation A.11 simply states that a circulating electric field creates a time-varying (perpendicular) magnetic field. Similarly from Equation A.12, a circulating magnetic field creates a time-varying (perpendicular) electric field. In terms of dynamic electromagnetic behavior the pair of equations are coupled together by the spacial and temporal expression of the electric and magnetic fields.

If one considers the three-dimensional Cartesian coordinate system (x,y,z), decomposing the curl operator in Equations A.9 and A.10 yields:

$$\frac{\partial H_x}{\partial t} = \frac{1}{\mu} \cdot \left(\frac{\partial E_y}{\partial z} - \frac{\partial E_z}{\partial y} - \rho' H_x \right) \tag{A.13a}$$

$$\frac{\partial H_y}{\partial t} = \frac{1}{\mu} \cdot \left(\frac{\partial E_z}{\partial x} - \frac{\partial E_x}{\partial z} - \rho' H_y \right) \tag{A.13b}$$

$$\frac{\partial H_z}{\partial t} = \frac{1}{\mu} \cdot \left(\frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x} - \rho' H_z \right) \tag{A.13c}$$

$$\frac{\partial E_x}{\partial t} = \frac{1}{\varepsilon} \cdot \left(\frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} - \sigma E_x \right) \tag{A.14a}$$

$$\frac{\partial E_y}{\partial t} = \frac{1}{\varepsilon} \cdot \left(\frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} - \sigma E_y \right) \tag{A.14b}$$

$$\frac{\partial E_z}{\partial t} = \frac{1}{\varepsilon} \cdot \left(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} - \sigma E_z \right) \tag{A.14c}$$

This system of six coupled equations form the foundation of the FDTD algorithm for modelling the interaction of electromagnetic waves with general, three-dimensional objects. These equations are not suitable for computer implementation, yet, because they contain continuous-time and continuous space variables. Kane Yee's method provides an efficient way to transform this system of partial difference equations into a second-order accurate system of difference equations.

A.2 Yee's Method

Most of the following material in this section is adapted from Taflove [9] and Yee's original paper [1].

Yee used centered finite-difference operators to represent both the spatial and temporal partial derivatives in Equations A.13a to A.14c. These centered differences are easy to implement on a computer. Furthermore, they are second-order accurate.

The following equation is Yee's centered-difference expression of the spatial derivative at a given point, in space and time:

$$\frac{\partial u}{\partial x}(i\Delta x, j\Delta y, k\Delta z, n\Delta t) = \frac{u\Big|_{i+1/2,j,k}^{n} - u\Big|_{i-1/2,j,k}^{n}}{\Delta x} + O[(\Delta x)^2] \qquad \text{(A.15)}$$

In this case, the derivative of the function u at point $(i\Delta x, j\Delta y, k\Delta z)$ in space is given by the slope of the line formed between the points $\pm\Delta x/2$ on either side of $(i\Delta x, j\Delta y, k\Delta z)$ while $n\Delta t$ remains fixed. Likewise, Yee's description of the temporal derivative at a given point, in space and time, is:

$$\frac{\partial u}{\partial t}(i\Delta x, j\Delta y, k\Delta z, n\Delta t) = \frac{u\Big|_{i,j,k}^{n+1/2} - u\Big|_{i,j,k}^{n-1/2}}{\Delta t} + O[(\Delta t)^2] \qquad \text{(A.16)}$$

Now, the centered difference function u is taken at the same point $(i\Delta x, j\Delta y, k\Delta z)$ in space but $\pm\Delta t/2$ before and after the time $n\Delta t$.

Using Yee's centered-difference expressions (Equations A.15 to A.16), the partial derivatives of Equation A.13a can be expressed as:

$$\frac{\partial H_x}{\partial t} = \frac{1}{\mu} \cdot \left(\frac{\partial E_y}{\partial z} - \frac{\partial E_z}{\partial y} - \rho' H_x \right)$$

$$\Downarrow \text{ centered differences } \Downarrow$$

$$\frac{H_x \Big|_{i,j,k}^{n+1/2} - H_x \Big|_{i,j,k}^{n-1/2}}{\Delta t} = \frac{1}{\mu_{i,j,k}} \cdot \left(\begin{array}{l} \dfrac{E_y \Big|_{i,j,k+1/2}^{n} - E_y \Big|_{i,j,k-1/2}^{n}}{\Delta z} - \\ \dfrac{E_z \Big|_{i,j+1/2,k}^{n} - E_z \Big|_{i,j-1/2,k}^{n}}{\Delta y} - \rho'_{i,j,k} H_x \Big|_{i,j,k}^{n} \end{array} \right) \tag{A.17}$$

Rephrasing Equation A.17 yields the following "update equation":

$$H_x \Big|_{i,j,k}^{n+1/2} = H_x \Big|_{i,j,k}^{n-1/2} + \frac{\Delta t}{\mu_{i,j,k}} \cdot \left(\begin{array}{l} \dfrac{E_y \Big|_{i,j,k+1/2}^{n} - E_y \Big|_{i,j,k-1/2}^{n}}{\Delta z} - \\ \dfrac{E_z \Big|_{i,j+1/2,k}^{n} - E_z \Big|_{i,j-1/2,k}^{n}}{\Delta y} - \rho'_{i,j,k} H_x \Big|_{i,j,k}^{n} \end{array} \right) \tag{A.18}$$

Notice that Equation A.18 also contains the term $H_x \Big|_{i,j,k}^{n}$; this term is not actually stored or computed, so it must be approximated. If a *semi-implicit approximation* is used then $H_x \Big|_{i,j,k}^{n}$ is represented as:

$$H_x \Big|_{i,j,k}^{n} = \frac{H_x \Big|_{i,j,k}^{n+1/2} - H_x \Big|_{i,j,k}^{n-1/2}}{2} \tag{A.19}$$

Using the above expression (Equation A.19) in Equation A.18 yields:

$$H_x \Big|_{i,j,k}^{n+1/2} = H_x \Big|_{i,j,k}^{n-1/2} + \frac{\Delta t}{\mu_{i,j,k}} \cdot \left[\begin{array}{l} \dfrac{E_y \Big|_{i,j,k+1/2}^{n} - E_y \Big|_{i,j,k-1/2}^{n}}{\Delta z} - \dfrac{E_z \Big|_{i,j+1/2,k}^{n} - E_z \Big|_{i,j-1/2,k}^{n}}{\Delta y} \\ - \rho'_{i,j,k} \left(\dfrac{H_x \Big|_{i,j,k}^{n+1/2} - H_x \Big|_{i,j,k}^{n-1/2}}{2} \right) \end{array} \right] \tag{A.20}$$

Regrouping like terms yields an *explicit* update equation for H_x:

$$H_x\Big|_{i,j,k}^{n+1/2} = \left(\frac{1-\dfrac{\rho'_{i,j,k}\Delta t}{2\mu_{i,j,k}}}{1+\dfrac{\rho'_{i,j,k}}{2\mu_{i,j,k}}}\right)\cdot H_x\Big|_{i,j,k}^{n-1/2} + \left(\frac{\dfrac{\Delta t}{\mu_{i,j,k}}}{1+\dfrac{\rho'_{i,j,k}}{2\mu_{i,j,k}}}\right)\cdot\left[\begin{array}{c}\dfrac{E_y\Big|_{i,j,k+1/2}^{n} - E_y\Big|_{i,j,k-1/2}^{n}}{\Delta z} \\ -\dfrac{E_z\Big|_{i,j+1/2,k}^{n} - E_z\Big|_{i,j-1/2,k}^{n}}{\Delta y}\end{array}\right] \tag{A.21}$$

Using a similar method, to that applied to Hx, the electric field update equation for Ex is:

$$E_x\Big|_{i,j,k}^{n+1} = \left(\frac{1-\dfrac{\sigma_{i,j,k}\Delta t}{2\varepsilon_{i,j,k}}}{1+\dfrac{\sigma_{i,j,k}}{2\varepsilon_{i,j,k}}}\right)\cdot E_x\Big|_{i,j,k}^{n} + \left(\frac{\dfrac{\Delta t}{\varepsilon_{i,j,k}}}{1+\dfrac{\sigma_{i,j,k}}{2\varepsilon_{i,j,k}}}\right)\cdot\left[\begin{array}{c}\dfrac{H_z\Big|_{i,j+1/2,k}^{n} - H_z\Big|_{i,j-1/2,k}^{n}}{\Delta y} \\ -\dfrac{H_y\Big|_{i,j,k+1/2}^{n} - H_y\Big|_{i,j,k-1/2}^{n}}{\Delta z}\end{array}\right] \tag{A.22}$$

To simplify the notation, the terms C_a, C_b, D_a and D_b are introduced to represent the field vector coefficients and local material properties at each point in space (i,j,k). For stationary media, whose properties do not vary with time, these coefficients can be pre-computed. These terms are as follows:

$$C_a\Big|_{i,j,k} = \left(\frac{1-\dfrac{\sigma_{i,j,k}\Delta t}{2\varepsilon_{i,j,k}}}{1+\dfrac{\sigma_{i,j,k}\Delta t}{2\varepsilon_{i,j,k}}}\right) \qquad C_b\Big|_{i,j,k} = \left(\frac{\dfrac{\Delta t}{\varepsilon_{i,j,k}}}{1+\dfrac{\sigma_{i,j,k}\Delta t}{2\varepsilon_{i,j,k}}}\right) \tag{A.23}$$

$$D_a\,|_{i,j,k} = \left(\frac{1 - \dfrac{\rho'_{i,j,k}\Delta t}{2\mu_{i,j,k}}}{1 + \dfrac{\rho'_{i,j,k}}{2\mu_{i,j,k}}} \right) \qquad D_b\,|_{i,j,k} = \left(\frac{\dfrac{\Delta t}{\varepsilon_{i,j,k}}}{1 + \dfrac{\sigma_{i,j,k}\Delta t}{2\varepsilon_{i,j,k}}} \right) \tag{A.24}$$

Applying the above process to Equations A.13a to A.14c yields the six coupled, FDTD update equations:

$$H_x\Big|_{i,j,k}^{n+1/2} = D_{a,H_x}\,|_{i,j,k} \cdot H_x\Big|_{i,j,k}^{n-1/2} + D_{b,H_x}\,|_{i,j,k} \cdot \left[\frac{E_y\Big|_{i,j,k+1/2}^{n} - E_y\Big|_{i,j,k-1/2}^{n}}{\Delta z} - \frac{E_z\Big|_{i,j+1/2,k}^{n} - E_z\Big|_{i,j-1/2,k}^{n}}{\Delta y} \right] \tag{A.25a}$$

$$H_y\Big|_{i,j,k}^{n+1/2} = D_{a,H_y}\,|_{i,j,k} \cdot H_y\Big|_{i,j,k}^{n-1/2} + D_{b,H_y}\,|_{i,j,k} \cdot \left[\frac{E_z\Big|_{i+1/2,j,k}^{n} - E_z\Big|_{i-1/2,j,k}^{n}}{\Delta x} - \frac{E_x\Big|_{i,j,k+1/2}^{n} - E_x\Big|_{i,j,k-1/2}^{n}}{\Delta z} \right] \tag{A.25b}$$

$$H_z\Big|_{i,j,k}^{n+1/2} = D_{a,H_z}\,|_{i,j,k} \cdot H_z\Big|_{i,j,k}^{n-1/2} + D_{b,H_z}\,|_{i,j,k} \cdot \left[\frac{E_x\Big|_{i,j+1/2,k}^{n} - E_x\Big|_{i,j-1/2,k}^{n}}{\Delta x} - \frac{E_y\Big|_{i+1/2,j,k}^{n} - E_y\Big|_{i-1/2,j,k}^{n}}{\Delta x} \right] \tag{A.25c}$$

$$E_x\Big|_{i,j,k}^{n+1} = C_{a,E_x}\Big|_{i,j,k} \cdot E_x\Big|_{i,j,k}^{n} + C_{b,E_x}\Big|_{i,j,k} \cdot \left[\frac{H_z\Big|_{i,j+1/2,k}^{n} - H_z\Big|_{i,j-1/2,k}^{n}}{\Delta y} - \frac{H_y\Big|_{i,j,k+1/2}^{n} - H_y\Big|_{i,j,k-1/2}^{n}}{\Delta z} \right] \tag{A.26a}$$

$$E_y\Big|_{i,j,k}^{n+1} = C_{a,E_y}\Big|_{i,j,k} \cdot E_y\Big|_{i,j,k}^{n} + C_{b,E_y}\Big|_{i,j,k} \cdot \left[\frac{H_x\Big|_{i,j,k+1/2}^{n} - H_x\Big|_{i,j,k-1/2}^{n}}{\Delta z} - \frac{H_z\Big|_{i+1/2,j,k}^{n} - H_z\Big|_{i-1/2,j,k}^{n}}{\Delta x} \right] \tag{A.26b}$$

$$E_z\Big|_{i,j,k}^{n+1} = C_{a,E_z}\Big|_{i,j,k} \cdot E_z\Big|_{i,j,k}^{n} + C_{b,E_z}\Big|_{i,j,k} \cdot \left[\frac{H_y\Big|_{i+1/2,j,k}^{n} - H_y\Big|_{i-1/2,j,k}^{n}}{\Delta x} - \frac{H_x\Big|_{i,j+1/2,k}^{n} - H_x\Big|_{i,j-1/2,k}^{n}}{\Delta y} \right] \tag{A.26c}$$

To compute the new magnetic field value ($H_x\Big|^{n+1/2}$) at a given point, in space and time, Equation A.25a uses only the current fields ($E_y\Big|^{n}$ and $E_z\Big|^{n}$) and previous field ($H_x\Big|^{n-1/2}$), all three of which are stored in computer memory.

Equations A.25a to A.26c comprise the simplest form of the FDTD update equations which are implemented as the core engine of the FDTD algorithm. For each time step, Δt, all of the magnetic fields, followed by all of the electric fields, are calculated for the entire three-dimensional volume of Yee cubes.

A.3 The Yee Cube

The manner in which Equation A.25a calculates the updated value of the field Hx is intuitively explained by the Yee cube, which is a graphical representation of Yee's discrete electric and magnetic fields in discrete space. The Yee cube is shown in Figure A.1.

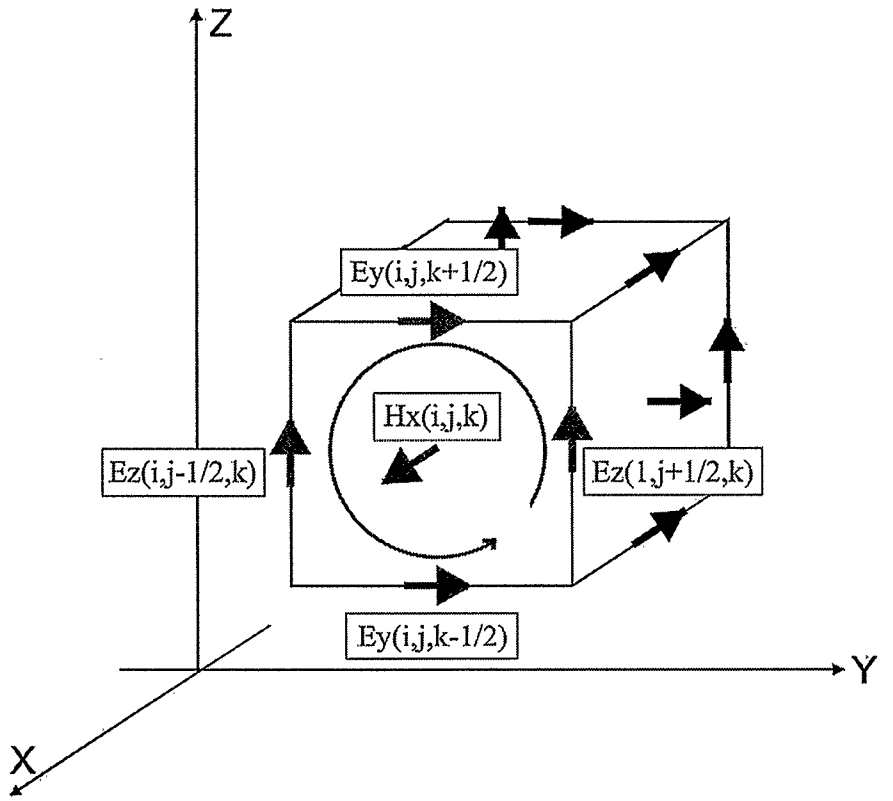


Figure A.1: The Yee Cube

In this case, the new value of Hx, centered at (i,j,k), is determined by the weighted contour sum of the surrounding electric fields ($E_z\left|^{n}_{i,j+1/2,k}\right.$, $-E_y\left|^{n}_{i,j,k+1/2}\right.$, $-E_z\left|^{n}_{i,j-1/2,k}\right.$, $E_y\left|^{n}_{i,j,k-1/2}\right.$). For two of the fields, the negative signs are introduced to maintain sign convention.

A.4 Summary of Yee's Method

The use of centered-difference equations has a few important effects on Maxwell's equations:

- The three-dimensional volume of interest is now divided into a mesh of discrete Yee cubes. Each cube has dimensions Δx, Δy and Δz. The example in Figure A.2 is simplified for visual understanding. A much finer grid is needed in order to generate accurate results and represent the model's materials (a complex cellphone and human head, in this case) with sufficient resolution.

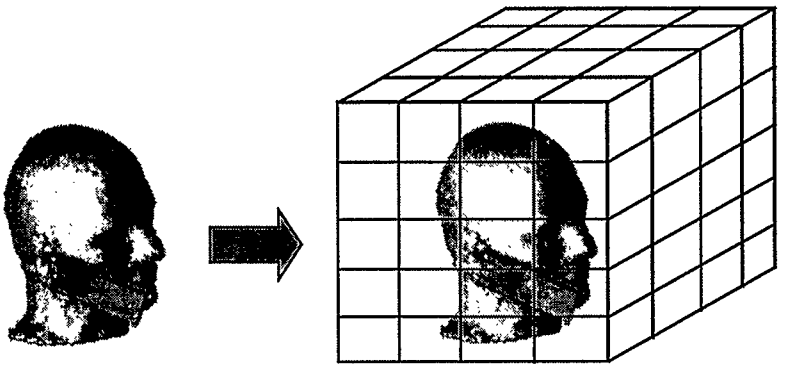


Figure A.2: Model of a Three-Dimensional "Volume of Interest"

- The continuous electric and magnetic fields become discrete fields centered on the edges or faces of the Yee cubes, respectively.
- The continuous function of time is discretized into time steps of Δt.
- At each time step, all of the magnetic fields are updated, followed by all of the electric fields, using Equations A.25a to A.26c. Over an entire simulation, this

models the time-domain behavior of the electromagnetic fields in the "region of interest".

These effects are a result of the mathematics described in the previous sections. The above effects/properties make the algorithm very suitable for a software implementation.