

An FPGA Implementation of 3-D FDTD Field Compute Engine

Robin Pottathuparambil¹ Ryan S. Adams² Ron Sass¹
rpottath@uncc.edu radams41@uncc.edu rsass@uncc.edu

¹Reconfigurable Computing Systems Lab and ²Microwave Ferrites Lab
University of North Carolina at Charlotte
9201 University City Blvd.
Charlotte, NC 28223-0001

Abstract

The Finite-Difference Time-Domain (FDTD) method has been used extensively for several years to analyze planar microstrip circuits. This paper presents a 3-D FDTD field compute engine design on an Virtex-4 FPGA, which is used to analyze discontinuities in planar microstrip circuits. The hardware design presented here does not grow on resources, as the problem size scales up. The paper also presents a mathematical formulation to compute the theoretical speed-up of the design. Gated clocking techniques have been used to reduce the power consumption. Details of the implementation along with the results are presented, which shows that with a careful design a speed-up over a standalone PC and a lower power consumption is achieved.

1. Introduction

Over the past two decades processors and Graphical Processor Units (GPUs) have been extensively used for accelerating scientific applications. One of the important scientific application domains is electromagnetic wave analysis. Electromagnetic wave analysis is done in many areas of research like RF analysis in PCB design, wave propagation in antennas, microstrip discontinuities analysis and others. Electromagnetic wave analysis is performed by solving Maxwell's equations. These equations are partial differential equations which govern the propagation of electromagnetic wave. These partial differential equation are discretized over a finite volume and the derivatives are approximated using central difference approximations. These finite-difference equations are then solved in a leap-frog manner to compute the electric and magnetic fields (E and H respectively) in so called the FDTD method [20]. The work presented here mainly focuses on FDTD computation for analyzing planar microstrip discontinuities [16]. Since

these discontinuities are too close to each other and due to the interaction of higher-order waves, the use of network analysis is not accurate.

In order to use the FDTD method to compute the electric and magnetic fields, a computational domain must be established. This computational domain is built by stacking small cubes and there by form a rectangular structure. Each cube or unit cell is defined by six fields as shown in figure 1. These six field locations are considered to be interleaved in space. The computation domain can be made up of any material, which is specified by permeability, permittivity, and conductivity. The time of simulation is divided into small time-steps, where each time-step represents the time required for the field to travel from one unit cell to the next unit cell. Once the computational domain is setup and the time-step is determined, an excitation is applied using a waveform. The waveform values at each time-step is propagated throughout the computational domain, depending on the characteristics of the unit cell. At each time-step the field values of all the unit cells in the computational domain are updated. These updates or computations are to be performed until all the field values have decayed to zero or a steady-state condition has reached.

These field value update or the computation needs to be performed for all the fields (electric field and magnetic field along x , y , z axes) in the entire computational domain for each time-step. Computation of electric field E and the magnetic field H depends respectively on the magnetic and electric field of the adjacent unit cell. Typically each unit cell has six field calculations and each field calculation needs at least four adjacent field values along with the previous time-step field value. These above requirements of the FDTD simulation inherently creates bottlenecks in a standalone PC environment.

The first computational bottleneck is the amount of data that is required to compute the field values at each time-step. Every computation of E requires at least four single-precision H values (adjacent cell values) with the corre-

sponding permittivity of the location and the previous time-step value of E . These values are to be fetched from the cache. Since the adjacent cell values are not stored contiguous in the main memory these each set of fetch could give rise to a cache miss, there by increasing the time of computation. The second bottleneck is the amount of data that needs to be fetched for each computation. The third bottleneck is the computation of the E and H that has to be done for the entire computational domain (typically around 1.5 Million nodes) for each time-step. The processor computes each value sequentially and there by requires enormous amount of time to compute.

The paper describes an elegant design and its implementation on an FPGA, that addresses the above computational bottlenecks. Most of the co-processor designs in the literature grows on resources as the problem size increases. The design presented here does not grow on resources as the problem size increases. The 3-D FDTD field Compute engine is built on an Virtex-4 XC4VFX60 FPGA fabric with various advantages. By aligning the data in a contiguous manner, by fetching data directly from the main memory at a very high transfer rate and by paralleling and pipelining the field compute core the above computational bottlenecks are addressed. The rest of this paper is organized as follows. The next section provides an overview of various implementation techniques for FDTD and related FPGA work. A mathematical formulation to compute the theoretical speed-up of the design and the details of the proposed design are described in Section 3. In Section 4, the performance results are compared against competing solutions. The paper concludes with a brief summary and discussion of future directions.

2. Background and Related work

This section gives a brief idea of various implementation techniques and approaches that are often used to accelerate FDTD method. The major models include standalone computing model and the co-processor model [14]. However the co-processor model uses either a FPGA, ASIC or a GPU with its own memory connected to a processor or a host PC using a PCI bus. To normalize the performance metrics of these different approaches, the literature adopts the methodology of Million nodes per seconds (Mnps). Each node is described as a unit cell and each unit cell is described by six fields E_x, E_y, E_z and H_x, H_y, H_z . Mnps describes how many complete computation (six of them) for a single node or unit cell can be done in a second. [13] computes Mnps Mathematically as $(N_x \times N_y \times N_z \times N_t) / runtime$, where N_x, N_y, N_z is the number of unit cells in x, y, z directions respectively. N_t is number of time-steps.

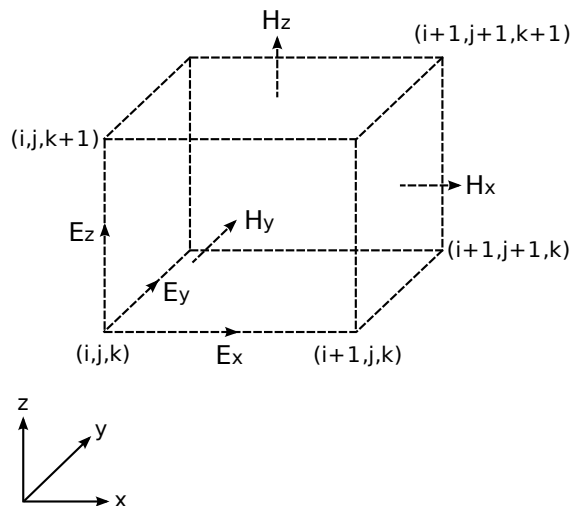


Figure 1. Field components on Yee's Cell

2.1. Standalone Design

A standalone computing model represents a complete implementation of FDTD design on an FPGA or on an ASIC. A 1-D FDTD was designed by [15] on a Xilinx Virtex XCV300 part. The FDTD computational cells are implemented as a pipelined bit serial arithmetic structure for computing E and H fields. The computation result was available every 849ns. The 1-D design was then extrapolated for two and three-Dimension FDTD Computation. The design implemented the entire computational unit cell on the hardware and a larger problem would require more hardware.

2.2. ASIC Based Co-Processor Design

In a ASIC based co-processor design the compute intense code is implemented on a custom VLSI chip and is interfaced with a host PC through a PCI. [12], [19] and [11] implemented an FDTD compute engine with a set of control units and registers. The performance of the design was evaluated using a fairly simple problem of $21 \times 21 \times 21$ unit cells. The design stores the field values of the cells which are placed in a straight row. The major drawback of the design is, as the problem scales up (i.e as the number of unit cells increases), the field values of the cells in a straight row cannot be stored in the same memory row and hence the required field values for computation may not come in a sequence. However if the problem is divided into a smaller block which aligns up in the SDRAM, then

the there could be pipeline stalls which would decrease the speed of computation. The design does not discuss about the computation of the cells at the dividing boundary. A speed-up of the order of $4\times$ is obtained when comparing the hardware simulated results with a software version. A better design was proposed in [18], where the design can handle a maximum problem size of $80 \times 80 \times 128$. However if the problem scales up the resources have to be scaled up and there is no mechanism to incorporate larger problems without hardware changes. All the above designs are just simulated using a VHDL simulator. The simulation results claim a speed-up of a order of $100\times$ when compared to a software code.

2.3. FPGA Based Co-processor Design

In a FPGA based Co-Processor (accelerator) design, the FDTD compute engine is designed in a FPGA and it communicates with a host PC using a PCI or a standard interface. The host processor or PC initially describes the problem to be solved and sends the relevant information to the hardware accelerator (FPGA) for computation. The accelerator computes the values of E and H for each iteration and the computed data is send back to the host computer for post processing and visualization. A full co-processor model using an FPGA has been designed by [7] where a massive parallel architecture simultaneously computes multiple values of E and H . This method is far better than the single processor method as single processor method supports only a single computational pipeline. A $25\times$ speed-up was observed compared to a standard PC running the FDTD application with a processing power of 42 Mnps. The design does not throw any insight on the power consumed per cell computation. The paper does not discuss the issues when the problem scales. A 3-D FDTD design was developed by [6] and [5] where the paper discusses about design of computation engine, data storage and special nodes. The design was evaluated on a Xilinx Virtex-II 6000 FPGA with RAM banks and a PC interface. The results were then compared with standalone PC implementations (Matlab and C). The results showed that the PC implementation was faster than the FPGA implementations.

A second generation accelerator card was proposed and designed by [3] and [4]. The design was realized on a Xilinx Virtex-II 8000 FPGA with 16GB of DDR and 36 Mb of DDR SRAM. The FDTD simulation design was written in VHDL and pipelined for a peak performance of 15 GFLOPS. The performance was compared with a 30-Node cluster made of 3.0 GHz processors. The FPGA design was 3 to 4 times faster than the cluster implementation and 23 times faster than single PC implementation. The peak throughput was 30 Million nodes per seconds (Mnps). The board costs about 10K USD and does not discuss the power

consumption per node. A 3D FDTD implementation [17] was done on a custom board consisting of four Virtex-II Pro XC2VP70 with an additional FPGA for the PCI control. The design is completely pipelined for higher throughput. A problem size of $100\times 100\times 100$ was used to compare the execution with a standard PC. The standard PC took about 2046 seconds and that of the FPGA design with three improvement took about 21 seconds. There was 22-fold speed-up when compared with commercial software. The major drawback of the design is that the computation is done in 32-bit fixed-point arithmetic and there is a error of 0.01 or less with the simulation results. A 3-D FDTD accelerator is design in [2] using reconfigurable hardware, but however uses fixed-point arithmetic for computation.

2.4. GPU Based Co-Processor Design

In a co-processor design using GPUs, the GPU is used for the computational intense part and the rest of the post-processing is done in a high speed PC. A 2D and 3D FDTD solvers has been implemented by [13] using 7800 GTX 512 nVidia GPU. A point source in free-space and ring resonator problems were ported on the GPU. The problem sizes were $400 \times 400 \times 40$ and $296 \times 482 \times 40$ which took about 104 and 227 seconds respectively for 2500 and 8000 time steps. The average throughput was 153 Mnps and 164 Mnps (Million nodes per second) respectively. The paper does not give a insight about the hardware design and the GPUs normally consumes a lot of power and there could be a bottleneck if the design has to be scaled up.

There is lot of work that has been done in the area of 2-D implementation which are reported in [9], [10], [1], [8], but our major interest is in the design of a 3-D FDTD compute engine, its scalability in accordance with the problem size and the power consumption.

3. Design and Implementation

We begin by considering the maximum theoretical speed-up. Then we describe the design motivated by this formulation.

3.1. Formulation

By taking into consideration the on-chip memory (BRAM) size, the transfer rate from the main memory (DDR/DDR2) to the FPGA, and the speed of computation, the theoretical speed-up is formulated below. Let M_B be the total BRAM memory in bytes available in a FPGA. Equations (1), (2), (3) each require five elements for computation. Each element is a single-precision floating-point of four bytes in size. Hence the total number of elements that

can be stored on-chip is $N = M_B/4$.

$$E_{x,i,j,k}^{n+1} = E_{x,i,j,k}^n + C_1 \left(H_{z,i,j+1,k}^{n+1/2} - H_{z,i,j,k}^{n+1/2} \right) + C_2 \left(H_{y,i,j,k+1}^{n+1/2} - H_{y,i,j,k}^{n+1/2} \right) \quad (1)$$

$$E_{y,i,j,k}^{n+1} = E_{y,i,j,k}^n + C_3 \left(H_{x,i,j,k+1}^{n+1/2} - H_{x,i,j,k}^{n+1/2} \right) + C_4 \left(H_{z,i+1,j,k}^{n+1/2} - H_{z,i,j,k}^{n+1/2} \right) \quad (2)$$

$$E_{z,i,j,k}^{n+1} = E_{z,i,j,k}^n + C_5 \left(H_{y,i+1,j,k}^{n+1/2} - H_{y,i,j,k}^{n+1/2} \right) + C_6 \left(H_{x,i,j+1,k}^{n+1/2} - H_{x,i,j,k}^{n+1/2} \right) \quad (3)$$

To compute E and H six elements are required and each unit cell is described by six elements. So the number of unit cells that can be stored on chip is $N_{uc} = N/6$. Let us assume the problem has $i_{pts} \times j_{pts} \times k_{pts}$ unit cells in x , y , z directions respectively, denoted as N_{up}

case 1 If $N_{up} \leq N_{uc}$ i.e the number unit cells in the problem is less than or equal to the number unit cells that can be stored on-chip, then there would be one set of computation for computing E field and one for computing H field values. However when the number of unit cells does not fit into the on-chip memory, then the problem becomes interesting,

case 2 If $N_{up} > N_{uc}$ i.e the number unit cells in the problem is greater than the number unit cells that can be stored on-chip, then the computation has to be done in blocks. The problem is typically divided into blocks along the longest path. Hence a block is defined as a set of contiguous unit cells. Assuming the longest path to be in the y direction. The number of divided blocks are

$$N_b = \frac{j_{pts}}{\frac{N_{uc}}{i_{pts} \times k_{pts}}} = \frac{i_{pts} \times j_{pts} \times k_{pts}}{N_{uc}} = \frac{N_{up}}{N_{uc}}$$

Let us assume the time taken for computing each block is t_b seconds and the time for computing E and H blocks are the same, then the computation time is,

$$t_E = t_H = \frac{t_b \times N_{up}}{N_{uc}} = t_b \times N_b \quad (4)$$

Or the time for computing both E and H blocks is,

$$t_E + t_H = 2 \times t_b \times N_b \quad (5)$$

However the time for computing each block t_b depends on various factors. Some of the factors are the data transfer from the main memory to the FPGA BRAM memory, the time for actual computation, the time to write back the re-

sults to the main memory. Hence

$$t_b = t_{mc} + t_c + t_{cm} \quad (6)$$

Where t_{mc} is the time taken to transfer data from the main memory to the on-chip memory. t_c is the time taken for the computation and t_{cm} is the time taken to transfer data from the on-chip memory to the main memory. The time taken to transfer data from the main memory to the on-chip memory depends on the bus transfer rate. The NPI (Native Port Interface) Bus transfers data at a theoretical rate of 1.6 GB/s. There are six NPI interfaces to the DDR2 main memory. The total theoretical bandwidth is 3.2 GB/s. Let us assume the size of the divided block which fits the on-chip memory is $i_{pts} \times dj_{pts} \times k_{pts}$, where dj_{pts} is a small portion along the y axis.

$$N_{uc} = i_{pts} \times dj_{pts} \times k_{pts}$$

Each unit cell has six elements (single-precision). All the six elements of a unit cells are to be transferred to compute the field values. The total number of elements or bytes to transfer from main memory to on-chip memory or the size of the block (S_b) in bytes,

$$S_b = 6 \times 4 \times N_{uc}$$

Time taken to transfer S_b from main memory to on-chip memory at 4.8GB/s is

$$t_{mc} = \frac{24 \times N_{uc}}{3.2 \times 10^9}$$

$$t_{mc} = 7.5 \times N_{uc} \times 10^{-9} \quad (7)$$

The time of computation, t_c depends on rate at which the computation is done. The theoretical rate of computation is 10ns per computation of E or H . Assuming the computation of E_x , E_y , E_z are done in two clock cycles (a detailed explanation of why two clock cycles is required described in the design section), the time of computation, t_c is

$$t_c = 1.5 \times N_{uc} \times 10^{-9} \quad (8)$$

The core computes E_x , E_y , E_z in two clock cycles and each element is 4 bytes long and hence the time to transfer back the computed data, t_{cm} to the main memory is derived as

$$t_{cm} = \frac{3 \times 4 \times N_{uc}}{3.2 \times 10^9}$$

$$t_{cm} = 3.75 \times N_{uc} \times 10^{-9} \quad (9)$$

The actual time to compute, t_b from equations (7), (8), (9) and (6)

$$t_b = 12.75 \times N_{uc} \times 10^{-9} \quad (10)$$

Speed-up: The FDTD computation algorithm contains a sequential section S_{sec} , a critical section C_{sec} and non-critical section NC_{sec} . The sequential section is executed once. The critical and non-critical section is executed n times for each port p . Let us assume t_s as the time of execution for the sequential part S_{sec} , t_c as the time of execution for the critical part C_{sec} and t_{nc} as the time of execution for the non-critical part NC_{sec} . The total time of execution is

$$t_t = t_s + n \cdot p \cdot (t_c + t_{nc})$$

The time for the critical section consists of time for computing both E and H .

$$t_t = t_s + n \cdot p \cdot (t_E + t_H + t_{nc})$$

Let us denote the speed-up calculations with for the first architecture with subscript 1 and subscript 2 for the FPGA design. The total time of execution for the designs is

$$t_{t1} = t_{s1} + n \cdot p \cdot (t_{E1} + t_{H1} + t_{nc1})$$

$$t_{t2} = t_{s2} + n \cdot p \cdot (t_{E2} + t_{H2} + t_{nc2})$$

By substituting t_{EH} from equation (5),

$$t_{t2} = t_{s2} + n \cdot p \cdot (2 \cdot t_b \cdot N_b + t_{nc2})$$

Speed-up,

$$\frac{t_{t1}}{t_{t2}} = \frac{t_{s1} + n \cdot p \cdot (t_{E1} + t_{H1} + t_{nc1})}{t_{s2} + n \cdot p \cdot (2 \cdot t_b \cdot N_b + t_{nc2})}$$

Substituting t_b from (10),

$$\frac{t_{t1}}{t_{t2}} = \frac{t_{s1} + n \cdot p \cdot (t_{E1} + t_{H1} + t_{nc1})}{t_{s2} + n \cdot p \cdot (2 \cdot (12.75 \cdot N_{uc} \times 10^{-9}) \cdot N_b + t_{nc2})}$$

$$\frac{t_{t1}}{t_{t2}} = \frac{t_{s1} + n \cdot p \cdot (t_{E1} + t_{H1} + t_{nc1})}{t_{s2} + n \cdot p \cdot (25.5 \cdot N_{uc} \times 10^{-9}) \cdot N_b + t_{nc2}} \quad (11)$$

According to equation (11) higher speed-up w.r.t to a processor design can be achieved if the time for computing each block (t_b) is reduced.

3.2. Code Analysis

The 3-D FDTD code for analyzing discontinuities in planar microstrip has a sequential, critical and non-critical sections of code. The critical and non-critical sections are iterated for time-steps of the order of 10,000. A fraction of speed-up in the iterated critical and non critical sections can reduce the overall execution time of the application and there by speed-up the application when compared to a processor execution. The set of routines in the sequential, critical and non-critical sections of code are shown in figure 2.



Figure 2. Flowchart showing the computing blocks

The first two blocks constitutes the sequential section of the application. The blocks numbered 1 and 4 are the critical sections of the code. Blocks 2,3 and 5 are the non-critical sections of the code. Block 1 computes the electric field along x , y and z axes. Block 2 computes the soft electric current density. Block 3 probes the voltages and currents for each time step. Block 4 computes the magnetic field and block 5 computes the soft magnetic field. Profiling data shows that blocks 1 and 4 takes the most time to execute followed by blocks 3, 2 and 5. The computation of electric field is done in block 1, if the computation is closely analyzed, the computation of electric field of each unit cell is independent of every other unit cell's electrical field computation. However the computation of electric field depends on the adjacent cell's magnetic field. Similarly is the case of computation of the magnetic field of each unit cell.

Block 3 measures the voltage (It is referenced to the ground plane) and currents (flowing into the device) at each port for each time-step. This is the next time consuming non-critical code in the iterative loop after the main field computation code. Every iterative loop computes voltage and current values for each port. The presented work in this paper just focuses on the design of field compute engine. The design of blocks 3, 2 and 5 are left for future work.

3.3. Design

The overall top-level design is shown in figure 3. Processor Local Bus (PLB), which is the main bus connects UARTLite, Ethernet, Compute Core, DDR2 (Double Data Rate RAM) and the PPC 405 core. The interrupt control block connects the interrupts from the various blocks (UARTLite, Ethernet, compute core) and generates a interrupt signal to the PPC405 core. The PPC core can then decode to find the interrupting peripheral. The UARTLite block is used for Standard I/O(stdio). The Ethernet Block (LL Temac) is used for TCP/IP communication. The PPC 405 core runs at 300 MHz. A 2.6 version of Linux is configured for the above base-system and it runs out of DDR2. With such a configuration the base-system would be able to run MPI based applications.

The compute core is directly connected to main memory (DDR2) through the Native Port Interface (NPI) and Multi-Port Memory Controller (MPMC) as shown in figure 4. The theoretical speed of NPI as discussed earlier is about 3.2 GB/s. The compute core connects the DDR2 through six channels. Each channel offering a maximum bandwidth of 1.6 GB/s. However the MPMC fetches data from DDR2 at a rate of 3.2 GB/s. The approach of having six channels to the MPMC was intentionally done so that each NPI channel can fetch data from different memory locations simultaneously, with read stalls. The MPMC controller has built-in read and write FIFOs to reduce the memory read stalls. The design can easily adapt, if the bandwidth of the MPMC interface to the DDR2 changes in future. The six NPI channels fetch the field values of all the three axes. These data is then written to a differential clocked FIFO through the control logic block. The control block decides when to use the NPI channels for read and write. The differential clocked FIFO accepts data (64-bit) at a rate of 200 MHz and outputs data (32-bit) at a rate of 100 Mhz. The data from the differential FIFO (32 word deep) is then written by the write block into the 64 KB BRAM block in a sequential order. Each BRAM block has two read/write ports.

Once the data is completely fetched from the main memory. The read logic block issues read request to read data from the BRAM memory in accordance with the data elements that is required for computing the field values. Every field value computation requires five elements. A total of fifteen elements are required to compute both the field values along all the three axes. The equations (1), (2) and (3) has three elements in common and so the total required data elements is twelve. Out of which three elements are stored in three BRAM blocks. The other nine elements are stored in the other BRAM blocks. Clearly a maximum of three compute cores can be connected to the read logic for a maximum throughput. The computation in the first core is done while the second and the third core waits for data. When

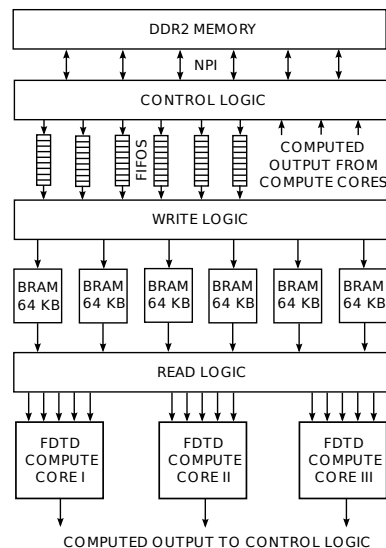


Figure 4. Design of FDTD Compute Core

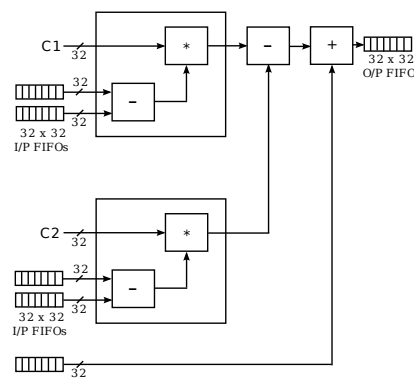


Figure 5. Design of Field Compute Core

the BRAM channels are freed, the computation in the second and third core is performed simultaneously. This cycle repeats to complete the computation of the all the field values. Figure 5 shows the design of the field compute core. The field compute core is a single-precision pipelined core, which is connected to read and write FIFOs. The data from the read logic is directly written into the FIFO and the field compute cores starts the computation when a valid data is available in the FIFO. The compute core only accepts 32-bit floating point values (single precision) for computation. The computed data is then written into the output FIFO. Once the output FIFO starts filling up the control logic starts writing back the data simultaneously, into the main memory using three NPI channels.

The hardware-software interface communicates data from the user program to the hardware. A Linux device

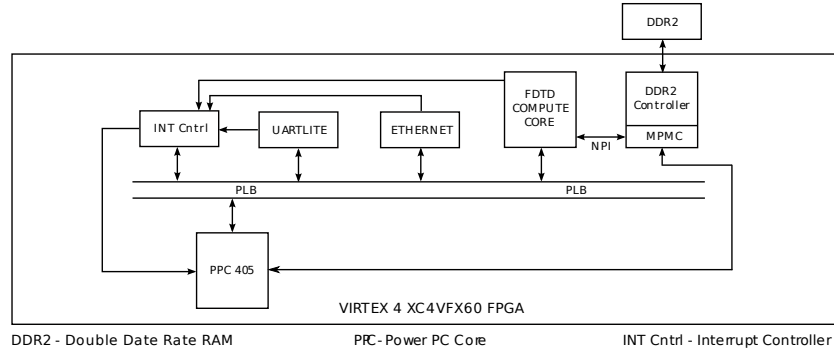


Figure 3. Design of Overall Base System

driver is written to read/write data from and to the DDR2 memory. The field values are mapped into the DDR2, so that the physical address of the field values are known. The field values are divided into smaller chunks along the longest path and is loaded into the BRAM for computation. The software takes care of values at dividing boundary by writing the dividing boundary values to the BRAM along with the other values. The next set of computation overlaps with a dividing boundary to compute the field values. As the problem size grows the number of divided blocks increases. Once the computation is done the field values are updated in the main memory.

Gated clocking techniques have been used in the design to keep the power consumption low. The clocks are turned off with gates to the blocks which are idle. When the control logic and write logic is writing data into the BRAM blocks, read logic and the field compute cores are idle. The clocks to these blocks (read logic and field compute cores) are turned off, thereby saving power. Similarly when the read logic is reading data, the write logic and the three NPI channels are idle and the clock are turned off, to save power. These techniques have slightly reduced the running power, which will be discussed in detail in the results section.

4. Results

The design described in the previous section has been implemented in VHDL. It was synthesized to run 100MHz with Xilinx Design Suite 10.1 and Xilinx Embedded Development Kit (EDK). The design consumed about 81% of the slices and 92% of the BRAM blocks. The table 1 shows the results of the implementation. The design was tested using a problem size of $101 \times 301 \times 51$ (around 1.5 million nodes). The test was performed for 10,000 iterations. The execution time of FPGA design was then compared with the execution time of a 2.8 GHz standalone PC. A speed-up of about 3x was observed with a peak performance of 6.5 Mnps. The

Table 1. Features of the Core (ND means no data available in reference)

Details	Our values	Kelmelis [7]	Durbano [4]
Problem Size	$101 \times 301 \times 51$	$100 \times 80 \times 150$	ND
Time Steps	10000	10000	ND
Mnps	6.5	42	30
Benchmark PC	P4 2.8 GHz	P4 1.5 GHz	P4 3.0 GHz
Speed-up	3	25	23
Processor	Virtex 4 FX60	ND	Virtex-II 8000
Core frequency	100 MHz	ND	100 MHz
Power	16.85 W	ND	ND
Cost	\$3,000	ND	\$10,000

Table 2. Power consumption comparison of Hardware

Details	Values	Cost
Pentium 4 2.8 GHz CPU	84 Watts	\$1,000
7800 GTX 512 550 MHz GPU	120 Watts	\$650
Virtex 4 XC4VFX60 FPGA	16.85 Watts	\$3,000

cost of the entire system was \$3,000. Gated clocking techniques were used to keep the power consumption low. The average running power was about 16.85 Watts. The gated clocking technique reduced the power consumption by 0.5 W. When compared with the designs in the literature the presented design has a low power consumption for a 3x speed-up and for a moderate cost. As discussed earlier in the design section if the code blocks 3, 2 and 5 are also implemented the peak performance would approximately double. The table 2 shows the power consumed by various devices that are used of FDTD computations. The table clearly shows that FPGAs stand out in the power aspect and with a careful design the peak performance of the design can be easily improved.

The next major advancements in this area of research is to (i) analyze and implement the code blocks 3, 2 and 5 as discussed in the code review section. Once the implementation is complete, the speed-up and the peak perfor-

mance would approximately double. (ii) To divide the problem space into smaller units and with help of MPI routines, the computation can be distributed to a set of FPGA nodes. Each FPGA node would be configured with the complete design to compute the field values.

5. Conclusion

The paper describes a design and implementation of a 3-D FDTD field compute engine on an platform FPGA. The paper presents a formulation to compute the theoretical speed-up of the design. The design described in the paper was able to compute field values with a complete base-system that runs Linux 2.6 out of the DDR2 memory. A speed-up of 3x was achieved with a peak performance of 6.5 Mnps when compared with a 2.8 GHz standalone PC for a problem size of 1.5 million nodes. Gated clocking techniques was used to reduce the power consumption by 0.5 Watts. The core consumes only 16.85 Watts of power and 81% of the total slices on Xilinx Virtex 4 XC4VFX60.

References

- [1] W. Chen, P. Kosmas, M. Leeser, and C. Rappaport. An FPGA implementation of the two-dimensional finite-difference time-domain (FDTD) algorithm. In *FPGA '04: Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pages 213–222, New York, NY, USA, 2004. ACM.
- [2] W. Chen, M. Leeser, and C. Rappaport. Acceleration of the 3D FDTD algorithm in fixed-point arithmetic using reconfigurable hardware. 2-5 August 2006.
- [3] J. P. Durban, J. R. Humphrey, F. E. Ortiz, P. F. Curt, D. W. Prather, and M. S. Mirotznik. Hardware acceleration of the 3D finite-difference time-domain method. *IEEE APS International Symposium on Antennas & Propagation*, June 2004.
- [4] J. P. Durban, F. E. Ortiz, J. R. Humphrey, P. F. Curt, and D. W. Prather. FPGA-based acceleration of the 3D finite-difference time-domain method. In *FCCM '04: Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 156–163, Washington, DC, USA, 2004. IEEE Computer Society.
- [5] J. P. Durban, F. E. Ortiz, J. R. Humphrey, D. W. Prather, and M. S. Mirotznik. Hardware implementation of a three-dimensional finite-difference time-domain algorithm. *IEEE Antennas and Wireless Propagation Letters*, 2:54–57, 2003.
- [6] J. P. Durban, F. E. Ortiz, J. R. Humphrey, D. W. Prather, and M. S. Mirotznik. Implementation of three-dimensional fpga-based FDTD solvers: An architectural overview. In *FCCM '03: Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, page 269, Washington, DC, USA, 2003. IEEE Computer Society.
- [7] E. Kelmelis, J. Durban, P. Curt, and J. Zhang. Field-programmable gate array accelerates FDTD calculations. *Laser Focus World (online)*, 2006. Available on the Web from http://www.laserfocusworld.com/articles/article_display.html?id=272176.
- [8] S. E. Krakiwsky, L. E. Turner, and M. M. Okoniewski. Graphics processor unit (GPU) acceleration of finite-difference time-domain algorithm. In *IEEE*, 2004.
- [9] J. R. Marek, M. A. Mehalic, J. Andrew, and J. Terzuoli. A dedicated VLSI architecture for finite-difference time domain calculations. In *In Proc. of The 8th Annual Review of Progress in Applied Computational Electromagnetics*, pages 546–553, March, 1992.
- [10] S. Matsuoka and H. Kawaguchi. FPGA implementation of the FDTD data flow machine. In *IEEE Topical Conference on Wireless Communication Technology*, 2003.
- [11] P. Placidi, L. Verducci, G. Matrella, L. Roselli, and P. Ciampolini. A custom VLSI architecture for the solution of FDTD equations. In *IEICE Transactions on Electronics*, volume E85-C, pages 572–577, 2002.
- [12] P. Placidi, L. Verducci, D. Tajolini, L. Roselli, and P. Ciampolini. A high-performance VLSI architecture for the ftdt algorithm. In *In Proc. of 2001 International Symposium on Signals, Systems, and Electronics*, July 24-27, 2001.
- [13] D. K. Price, J. R. Humphrey, and E. J. Kelmelis. GPU-based accelerated 2D and 3D FDTD solvers. volume 6468, page 646806. SPIE, 2007.
- [14] R. Schneider, D. Cyca, C. Mason, and M. Okoniewski. Evaluation of emerging hardware platforms for faster electromagnetic simulations. In *The 23rd Annual Review of Progress in Applied Computational Electromagnetics*, pages 97–105, March 19-23, 2007.
- [15] R. N. Schneider, L. E. Turner, and M. M. Okoniewski. Application of FPGA technology to accelerate the finite-difference time-domain (FDTD) method. In *FPGA '02: Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*, pages 97–105, New York, NY, USA, 2002. ACM.
- [16] D. M. Sheen, S. M. Ali, M. D. Abouzahra, and J. A. Kong. Application of the three-dimensional finite-difference time-domain method to the analysis of planar microstrip circuits. 38:849–857, Jul 1990.
- [17] H. Suzuki, Y. Takagi, R. Yamaguchi, and S. Uebayashi. FPGA implementation of FDTD algorithm. In *Microwave Conference Proceedings, 2005. APMC 2005. Asia-Pacific Conference Proceedings*, volume 3, page 4, Dec, 2005.
- [18] L. Verducci, P. Placidi, P. Ciampolini, A. Scorzoni, and L. Roselli. A standard cell hardware implementation for finite-difference time domain (FDTD) calculation. In *IEEE MTT-S International Microwave Symposium digest*, pages 2085–2088, 2003.
- [19] L. Verducci, P. Placidi, G. Matrella, L. Roselli, F. Alimenti, P. Ciampolini, and A. Scorzoni. A feasibility study about a custom hardware implementation of the ftdt algorithm. In *In the Proc. of The 27th General Assembly of the URSI*, 2002.
- [20] K. S. Yee. Numerical solution of initial boundary value problems involving maxwell's equations in isotropic media. *IEEE Transactions on Antennas and Propagation*, 14:302–307, 1966.