# FastImp USER'S GUIDE

Zhenhai Zhu Ben Song Jacob White

Research Laboratory of Electronics
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139 U.S.A.

July 1, 2003

This work was sponsored by the Semiconductor Research Corporation, the NSF program in computer-aided design, and grants from Synopsys, Compaq and Intel.

# Contents

FastImp is a wideband impedance extraction program. It can perform Magneto-Quasi-Static(MQS), Electro-Magneto-Quasi-Static(EMQS) and full-wave analysis of general 3-D conductors embedded in a constant permeable dielectric material.

FastImp includes two parts: the implementation of a wideband surface integral formulation and a pre-corrected FFT based fast integral equation solver, pfft++. The program pfft++ is an independent package and can be adapted for other applications, such as computational chemistry, computational aerodynamics. Tutorial information about pfft++ is available at rleweb.mit.edu/vlsi/codes.htm.

# 1 How to Prepare Input Files

FastImp supports two input file formats. One file format is designed for test 3D structures and the other is a general format that can be used to describe complex 3D structures.

## 1.1 The Simple Input File Format

This file format is mainly for debugging purpose because only simple shapes like wires, rings, circular spirals, rectangular spirals and ground planes are supported by this format. The following is an input file describing the geometry of a straight wire as well as the discretization of the surface of the wire.

```
   # This is an input file for FastImp
   # Structure type of each conductor is specified by a number
   # 1: straight wire
   # 2: ring
   # 3: spiral
   # 4: ground
   #
   # The unit of structre size is:
   # 1: m
   # 1e-2: cm
   # 1e-3: mm
   # 1e-6: um

{File
   1e-3   unit
   1   number of conductors
   { cond 1
    1   structure type
    { origin
      0
      0
      0
    } origin
    { The other end along width
```

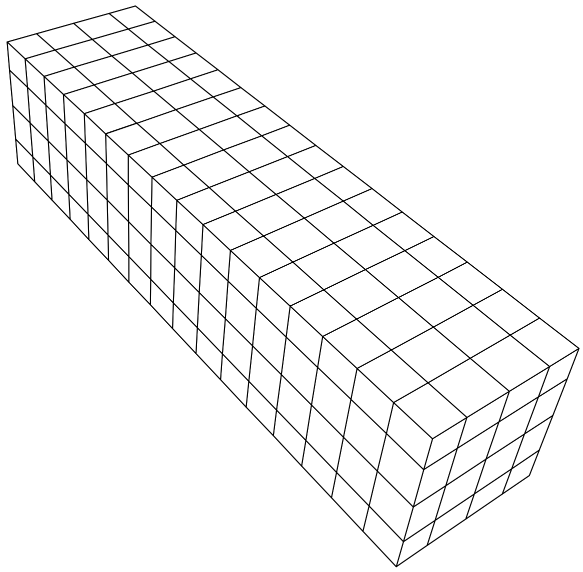

Figure 1: A Straight Wire

```
        1
        0
        0
      } The other end along width
      { The other end along thickness
        0
        0
        1
      } The other end along thickness
      { The other end along length
        0
        4
        0
      } The other end along length
      4   number of panels along width
      4   number of panels along thickness
      16  number of panels along length
      5.8e7  conductivity of copper
     } cond 1
}File
```

This wire has a $1 \times 1mm$ cross-section and is $4mm$ long. Each of the two ends of this wire is divided into $4 \times 4$ arrays of panels, and each of the four sides of this wire is divided into $4 \times 16$ arrays of panels. Figure 1 shows the mesh on the wire surface. Please note that the unit for conductivity is always $(m\Omega)^{-1}$.

The input file in this format for a wire, a ring, a circular spiral and a rectangular spiral can be found in fastImp/test/wire, fastImp/test/ring, fastImp/test/circularSpiral

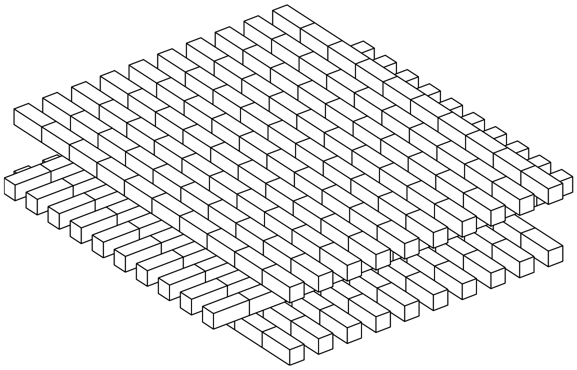

Figure 2: Three-layer 10x10x10 cross-over bus

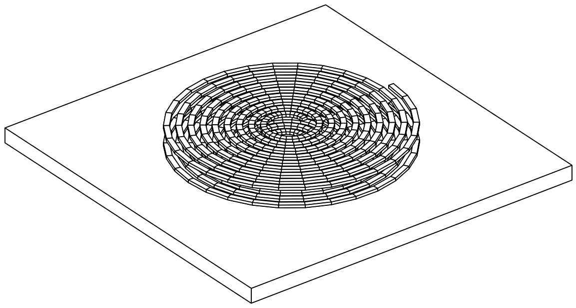

Figure 3: stacked circular 9-turn spiral with a substrate ground plane

and fastImp/test/rectSpiral.

This file format can be used to describe some more complicated structures, as shown in figures 2, 3 and 4. All these structures are just simple combinations of the wires, rectangular or circular spirals and ground planes. Input files for structures in figures 2, 3 and 4 are in fastImp/test/cross-over, fastImp/test/circularSpiral and fastImp/test/rectSpiral, respectively.

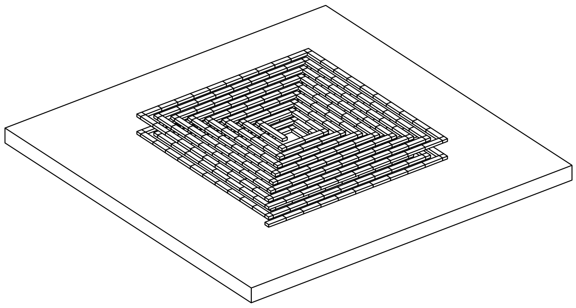

Figure 4: stacked rectangular 8-turn spiral with a substrate ground plane

## 1.2 The General Input File Format

In order to describe more general 3D structures, we have designed a file format very similar in spirit to the standard output of PATRAN, a general purpose, three-dimensional solid modeler with interactive graphics. The goal is that with minimal effort the user can interface FastImp with a surface mesh generator.

### 1.2.1 A simple example

The straight wire from section 1.1 is used as an example here. The input file (called `wire.pat`) in this general format is

```
c 1                      % number of conductor
n 290                    % number of vertexes
p 288                    % number of panels
u 0.001                  % unit of structure size
v 0 0 0 0 0              % first vertex
v 0 1 0.2173913043 0 0
v 0 2 0.5 0 0
v 0 3 0.7826086957 0 0
...
...
v 0 288 0.5 0 0.7826086957
v 0 289 0.7826086957 0 0.7826086957 % last vertex
Q 0  B 0 1 17 16         % buffer layer panel
Q 0  B 1 2 18 17
...
...
Q 0  B 31 16 32 47
Q 0  NC 32 33 49 48      % non-contact panel
Q 0  NC 33 34 50 49
...
...
Q 0  NC 222 223 239 238
Q 0  NC 223 208 224 239
Q 0  B 224 225 241 240
Q 0  B 225 226 242 241
...
...
Q 0  B 254 255 271 270
Q 0  B 255 240 256 271
Q 0  RC 256 257 272 271  % right-contact panel
Q 0  RC 257 258 273 272
...
...
Q 0  RC 279 280 265 266
```

```
Q 0  RC 280 263 264 265
Q 0  LC 15 281 1 0       % left-contact panel
Q 0  LC 281 282 2 1
...
...
Q 0  LC 10 9 289 288
Q 0  LC 9 8 7 289
```

### 1.2.2 Details of the general format file

**Summary** The first four lines of the general format input file give a summary of the input file. This summary information includes the number of conductors, number of vertexes, number of panels and the unit of the structure size.

**Vertexes** The following section gives the detailed information about vertexes. Each line gives information about one vertex and it always starts with the letter `v` or `V`. The syntax of each line in vertex section is:

```
v  ci  vi  x  y  z
```

where ci is the index of the conductor where the vertex belongs, vi is the index of the vertex, (x, y, z) are the global coordinates of the vertex. Please note that the vertex index is local to each conductor, i.e., it starts from zero on each conductor. Also (x, y, z) should be normalized by the structure size. For example, if it is (1, 1, 1) and the unit is $1mm$, then the real location of this vertex in the global coordinate system is $(1mm, 1mm, 1mm)$.

**Panels** The vertex section is followed by the panel section which gives the detailed information about panels. Each line gives information about one panel and starts with either the letter `Q` or `q` (for quadrilateral panel) or `T` or `t` (for triangle panel). The syntax of each line in the panel section is:

```
Q  ci  pt  i1 i2 i3 i4
```

or

```
T  ci  pt  i1 i2 i3
```

where ci is the index of the conductor where the panel belongs, pt is the panel type, integers i1, i2, i3 and i4 are the indexes of the vertexes of this panel. Please note this set of indexes should always be a subset of the indexes defined in vertex section. Please also note that the order of the vertexes should follow the left-hand rule, i.e., if the fingers of your left hand follow i1-i2-i3, then the thumb of your left hand should point to the outside of the conductor.

There are four panel types: buffer layer (`B`), non-contact (`NC`), left contact (`LC`) and right contact (`RC`). The left and the right contact panels belong to the left and the right end of the conductor. The excitation is posed on these two ends and these two ends are also the location of the ports of the equivalent circuit. All other panels that are not on the two ends of the conductor are called non-contact panels. For the purpose of computing current at high frequencies, two layers of quadrilateral panels are needed around each contact. These panels are called buffer panels. Please note that the shape of the buffer panels must be quadrilateral.

**Conductivity file** The conductivity of all conductors should be put into a different file. The file format is a simplified version of the simple input file in section 1.1, A companion file for `wire.pat` (called `oneCond.inp`) is

```
    # This is a conductivity file for FastImp
    # Structure type of each conductor is specified by a number
    # 5: non-ground
    # 6: ground

{File
   1e-3  unit
   1   number of conductors
   { cond 1
    5   structure type
    5.8e7  conductivity of copper
   } cond 1
}File
```

Please note that the unit in `oneCond.inp` must be the same as the one in `wire.pat`. The file `oneCond.inp` is available at fastImp/test/wire. The structure type of each conductor in conductivity file only has two possible values. Type 5 means the conductor is not grounded, type 6 means it is grounded. The grounded conductor will be removed from solve list. For example, if the structure to be analyzed is a wire above a lossy substrate ground plane, then the conductivity file should be something like this

```
    # This is a conductivity file for FastImp
    # Structure type of each conductor is specified by a number
    # 5: non-ground
    # 6: ground

{File
   1e-3  unit
   2   number of conductors
   { cond 1, wire
    5   structure type
    5.8e7  conductivity of copper
   } cond 1
   { cond 2, substrate ground plane
    6   structure type
    2.9e4  conductivity of substrate
```

```
    } cond 2
}File
```

FastImp will only do one field solve and the extracted impedance matrix has just one entry. Section 2 shows how to run `fastimp` using the general format input file and the conductivity file.

The details of the general format is documented in fastImp/src/mesh/mesh.cc::readMeshFile(). Please refer to head comments of this function if you want to write a program to generate general input files.

### 1.2.3 Generate the general input file from the simple input file

FastImp can read an input file in the simple format explained in section 1.1 and output the discretization into a user-specified file using the general format in section 1.2. For example, the command

```
fastimp -i wire.inp -o wire.pat
```

generates the example file `wire.pat` in section 1.2.1. This function is also useful in visualization, as will be shown in section 3.

# 2 Running FastImp

The basic form of the FastImp program command line is

```
fastimp [-i<input file>] [-b<beginning of frequency range>]
        [-e<end of frequency range>] [-n<number of frequency points>]
        [-s<extraction mode>]
```

## 2.1 Using the input file in simple format

The command

```
fastimp -i wire.inp -b1e6 -e1e6 -n1 -s0
```

runs `fastimp` on the example file `wire.inp` described in Section 1.1. The extracted input impedance is calculated using Magneto-Quasi-Static mode and is computed for the single frequency $f = 1MHz$.

## 2.2 Using the input file in general format

The command

```
fastimp -i oneCond.inp -r wire.pat -b1e7 -e1e9 -n3 -s1
```

runs `fastimp` on the example files `oneCond.inp` (defines conductivity) and `wire.pat` (defines the panel discretization) described in Section 1.2. The extracted input impedance is calculated using Electro-Magneto-Quasi-Static mode and the three sampling frequency points are $10MHz$, $100MHz$ and $1GHz$.

## 2.3 Example run

This section contains a sample run of FastImp for the straight wire example, `wire.inp` shown in Figure 1. The command

```
fastimp -i wire.inp -b1 -e1
```

produces the output below.

```
 fastimp v1.0  06/01/03
 FAST IMPedance extraction program

File [wire.inp] has been opened for read

File [wire.inp] has been closed for read

 Panels and nodes have been generated
 Number of conductors := 1
 Number of panels := 288, number of nodes := 290
 Non-uniform mesh has been used
 Only quadrilateral panels are used

setup kernel Independent part of pfft for each conductor surface
Number of src elements := 288
Min src element size := 0.307438
Max src element size := 0.399669
Average src element size := 0.35439
Number of grids := (8,16,8)  grid size := 0.314262
Max number of elements mapped to one grid point := 4
Min number of neighbors to one element := 31
Max number of neighbors to one element := 70

setup kernel independent part of pfft for the union of all non-contact conductor surface
Number of src elements := 288
Number of eval elements := 256
Min src element size := 0.307438
Max src element size := 0.399669
Average src element size := 0.35439
Number of grids := (8,16,8)  grid size := 0.314262
Max number of elements mapped to one grid point := 4
Min number of neighbors to one element := 20
Max number of neighbors to one element := 70

Total time for setting up frequency-independent part of pfft := 0.3 (seconds)

Estimated memory usage by pfft := 1 (MB)

Estimated memory usage by the pre-conditioner Matrix := 0 (MB)
Assumed number of non-zero per row after LU factorization:=   15

Estimated memory usage by the Hessenberg Matrix in Gmres := 1 (MB)
Assumed restart in Gmres :=   30

Estimated memory usage by the whole program := 2 (MB)
```

```
 Sampling frequency := 1
 Simulation Type is MQS
 Formulation Type used is OLD
 Current Computation mode is AUTO
 An iterative solver with pFFT is used
 Pre-condtioner is RIGHT_PRE_CONDITIONER
 Pre-conditioner is generated in local coord.
 The non-uniform mesh is used
 Only quadrilateral panel is used

 The whole system has 2018 unknowns

setup kernel dependent part of pfft for each conductor surface

time for the pfft setup for one conductor:= 3.77 (seconds)

setup kernel dependent part of pfft for the union of all non-contact conductor surface

time for the pfft setup := 0.18 (seconds)

Total time for setting up frequency-dependent part of pfft := 3.95 (seconds)

Time used for forming the pre-conditioner matrix := 0.07 (seconds)

The sparse matrix has been LU factored
The number of nonzero's before LU :=  8082
The number of nonzero's after LU :=  20337
The number of fill-in's :=   12255

Time used for LU factorizing the pre-conditioner matrix := 0.02 (seconds)
Time used for IE operator on a vector := 0.01 (seconds)
Time used for IE operator on a vector := 0 (seconds)

Time used for one matrix vector product := 0.05 (seconds)


Time used for one pre-conditioner solve := 0 (seconds)
1 2 3 4 5 6 7 8 9  outer resid = 0.000792216

 Number of Gmres iterations := 9
 Final residual := 0.000792216

Time for solving the system := 0.58 (seconds)
Terminal current computation uses low-frequency mode

 Zm[0, 0] = (6.91284e-05,9.51852e-09)
 f = 1 R = 6.91284e-05 L = 1.51492e-09

 All impedance matrices dumped to file Zm.dat
 Real and imaginary parts dumped to file R.dat and L.dat

Total time for extraction at one frequency point := 4.62 (seconds)
```

```
Total run time of fastImp := 4.62 (seconds)
```

## 2.4 Quick Reference

`fastimp` Options

| Option | Default | Range | Function |
|---|---|---|---|
| `-i` | — | — | Specifies simple input file name |
| `-r` | — | — | Specifies general input file name |
| `-o` | — | — | Specifies output mesh file name |
| `-s` | 0 | `0, 1, 2` | Specifies simulation mode, 0-MQS, 1–EMQS, 2–fullwave |
| `-b` | 1GHz | $> 0$ | Specifies the beginning of frequency range |
| `-e` | 1GHz | $> 0$ | Specifies the end of frequency range |
| `-n` | 1 | $> 0$ | Specifies number of sampling frequency points |
| `-m` | 0 | `0, 1` | Specifies frequency sampling method, 0–logarithmic, 1–linear |
| `-d` | 2 | `0, 1, 2` | Specifies linear system solver, 0–LU , 1–GMRES, 2– GMRES+pfft |
| `-l` | — | $> 0$ | Only extract one column of [Z] |
| `-t` | `1` | — | Transmission line switch, 1–on, otherwise–off |
| `-w` | `0` | `0, 1` | check panel size against skin depth or wave length, 1–yes, 0–no |
| `-h` | — | — | display command line option on-line help |
| `-v` | — | — | display fastimp version number |

Arguments can be specified in any order and are case sensitive.

## 2.5 Bugs

This section attempts to itemize the major shortcomings of `fastimp`.

1. Each conductor can not have more than two ports.

2. Layered media is not supported by this release.

3. Special discretization (the buffer layers) must be created around ports of each conductor. All panels in the buffer layers must be quadrilateral.

4. If the real part and the imaginary part of the impedance are different by more than four or five orders of magnitude, the smaller part might be inaccurate.

5. In order to capture the field singularity as well as the skin effect around corners and edges, the panel size at these locations should be comparable to skin-depth. On the other hand, the fast solver pfft++ is less efficient if the aspect ratio of panels is large (more than 20). Therefore, a large number of panels are needed when the skin-depth is small compared to the conductor length. One should bear in mind the dependency of discretization on frequency when running `fastimp`. The command line option `-w1` enables `fastimp` to check panel size.

# 3 Generating Pictures of the Input Geometry

FastImp can read in an input file in the general format and output the discretization into a user-specified file in FastCap input format. For example, the command

```
fastimp -i oneCond.inp -r wire.pat -o wire.fc
```

reads the example files `wire.pat` and `oneCond.inp` shown in section 1.2.1 and outputs the same panel discretization information to FastCap input file `wire.fc`. The following is the top section of the file `wire.fc`

```
0 mesh in FastCap format output from FastImp
Q 1  0 0 0  0.000217391 0 0  0.000217391 0.00025 0  0 0.00025 0
Q 1  0.000217391 0 0  0.0005 0 0  0.0005 0.00025 0  0.000217391 0.00025 0
Q 1  0.0005 0 0  0.000782609 0 0  0.000782609 0.00025 0  0.0005 0.00025 0
Q 1  0.000782609 0 0  0.001 0 0  0.001 0.00025 0  0.000782609 0.00025 0
......
......
```

The user can then use FastCap's or FastHenry's visualization option to generate a postscript file from this FastCap input file. We recommend using FastHenry's `zbuf` program to create the postscript file because it handles panel ordering better. In addition, the program `zbuf` in FastHenry version 3.0 takes the "-m" argument to produce a Matlab file for faster visualization in Matlab. See FastHenry's user guide for details on using program `zbuf`. Once the matlab file is generated, the matlab script `plotfastH.m` (available under fastImp/bin) can be used to generate the postscript file. Table 1 gives the commands used to produce the pictures in this guide as examples. As pointed out in FastCap and FastHenry user's guide, the number of panels used for visualization should be no more than a couple of thousands. Otherwise, it will take long time to generate the postscript file. Though it may not be a serious problem if we use Matlab script to generate the postscript file, it is still wise to reduce the number of panels in the input file. The three input files for visualization purpose to generate figure 2, 3 and 4 are all modified version of the original input file. These three visualization input files `bus10x10x10.visual`, `stack_9turn_withGround.visual` and `stack_8turn_withGround.visual` are also available along with their real input file counterpart.

# A Compiling FastImp

FastImp was developed on a desktop PC with Linux as its operating system. Therefore, we treat compilation on Linux as the default procedure.

## A.1 Untar the package

A tar file containing the source files for FastImp and its input files may be obtained from `http://rleweb.mit.edu/vlsi/code.htm` or by sending an email to Prof. Jacob White

| Figure | `fastimp` Usage | Comments |
|---|---|---|
| 1 | `fastimp -i wire.inp -o wire.pat` | general input file |
| | `fastimp -i wire.inp -r wire.pat -o wire.fc` | fastcap input file |
| | `fasthenry/bin/zbuf wire.fc` | postscript file |
| 2 | `fastimp -i bus10x10x10.visual -o b10.pat` | general input file |
| | `fastimp -i bus10x10x10.inp -r b10.pat -o b10.fc` | fastcap input file |
| | `fasthenry/bin/zbuf -m b10.fc` | Matlab file |
| | `fastImp/bin/plotfastH('b10.mat')` | postscript file |
| 3 | `fastimp -i stack_9turn_withGround.visual -o s9.pat` | general input file |
| | `fastimp -i stack_9turn_withGround.inp -r s9.pat -o s9.fc` | fastcap input file |
| | `fasthenry/bin/zbuf -m s9.fc` | Matlab file |
| | `fastImp/bin/plotfastH('s9.mat')` | postscript file |
| 4 | `fastimp -i stack_8turn_withGround.visual -o s8.pat` | general input file |
| | `fastimp -i stack_8turn_withGround.inp -r s8.pat -o s8.fc` | fastcap input file |
| | `fasthenry/bin/zbuf -m s8.fc` | Matlab file |
| | `fastImp/bin/plotfastH('s8.mat')` | postscript file |

Table 1: Commands used to generate the figures in this guide.

at white@mit.edu. This address may also be used for general correspondence regarding FastImp.

The tar file has the form

```
fastimp-1.0.tgz
```

and yields a three level directory tree when untarred with the command

```
tar xzvf fastimp-1.0.tgz
```

The top-level directory is called `fastimpRelease` and has two subdirectories. The directory `fastImp` contains all the files for the surface integral formulation. And the directory `pfft++` contains all the files for the fast integral equation solver (also released separately at `http://rleweb.mit.edu/vlsi/code.htm` with tutorial materials). Subdirectories `fastImp` and `pfft++` have the same subdirectory tree structure. The directory `src` and `inc` contain all the source files; `util` contains all the utility files for compilation purpose; and `test` contains the example input files. Executables are stored in the `bin` directory after compilation using the procedures described below.

## A.2 Default Compilation Procedure For Linux

The main `makefile` is under `fastImp`. It contains the commands to compile the source code in both `fastImp` and `pfft++`. The executable `fastimp` can be generated and put into a sub-directory under `fastImp/bin` by typing

```
cd fastImp
make
```

The name of this sub-directory is automatically generated by `fastImp/util/config.guess`. This name is a combination of your computer type and the operating system type. For example, if you use a desktop PC with linux as the operating system, this sub-directory would be called `ix86-linux`.

## A.3 Compilation with Compaq True 64

If you don't use gcc and g++, you might have to modify the following files:
`fastImp/def.makefile`,
`fastImp/makefile`,
`fastImp/util/share.cc.make`,
`fastImp/util/share.c.make`.
`pfft++/def.makefile`,
`pfft++/makefile`,
`pfft++/util/share.cc.make`,
`pfft++/util/share.c.make`.
We have ported the code to Compaq's true64 c++ compiler with minor modification to these 8 files and no change in the source codes. The Compaq version of these 8 files
`fastImp/compaq.def.makefile`,
`fastImp/compaq.makefile`,
`fastImp/util/compaq.share.cc.make`,
`fastImp/util/compaq.share.c.make`
`pfft++/compaq.def.makefile`,
`pfft++/compaq.makefile`,
`pfft++/util/compaq.share.cc.make`,
`pfft++/util/compaq.share.c.make`
are also included in this release and they might give you some hint if you run into compilation problems in other platforms.