

# Politecnico di Torino

Project for Advanced Design for Signal Integrity and Compliance

## A Power Bus-Model for High-Speed Memory Circuits



# Politecnico di Torino

ALESSANDRO ROCCO SCISCA  
s276032

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Power Integrity . . . . .	3
<b>2</b>	<b>Model</b>	<b>4</b>
2.1	Sections . . . . .	4
2.2	Parameters . . . . .	4
2.3	Fitting . . . . .	5
2.4	From PUL Parameters to Z Matrix . . . . .	5
2.5	Results . . . . .	6
<b>3</b>	<b>Implementation</b>	<b>7</b>
3.1	Features and Organization . . . . .	7
3.2	Reading Measurements . . . . .	8
3.3	Geometric and Complete Models . . . . .	8
3.4	Fitting . . . . .	9
3.5	Importing and Exporting . . . . .	10
3.6	Results . . . . .	10
	<b>References</b>	<b>14</b>

# 1 Introduction

This project is based on the results of the article *A Broadband Chip-Level Power-Bus Model Feasible for Power Integrity Chip-Package Codesign in High-Speed Memory Circuits* [1]. All the source code and experimental data is publicly available on [GitHub](#).

## 1.1 Power Integrity

As for most devices gaining traction both in the consumer oriented and in the general electronics markets, memory systems have been greatly increasing their performance over the past years and newer standards are always proposed for future architectures.

Along with the significant increase in capacity, it is the memory speed -intended as its data transfer rate- which is central in this constant improvement process. Speed is such a crucial parameter because memories are generally much slower than the main processor, meaning that trying to access data may cost a few precious CPU cycles whenever such data is to be read from the main memory (e.g.: not present in caches or local CPU registers). The system will most certainly benefit from a faster memory, but the design process of such high speed memories becomes more and more of concern. Specifically, the chip's Power Integrity has to be well analyzed and taken care of.

Writing data to a bus at such speeds requires not only faster bus drivers, but also a solid internal power bus which may suffer from the constant high frequency, high current switching that is required. Because of the parasitics of the bus, voltage drops and unintended filtering may cause the outputs to be unrecognizable. This is the reason why it is necessary to study an efficient model for the power bus in order to simulate its behavior at arbitrary frequencies.

## 2 Model

### 2.1 Sections

Regardless of the Power Integrity issues, memories are organized so that each output data pin DQ is interleaved with VDDQ and VSSQ power pins as in Figure 1 to help the receiver recognizing the data which may be distorted at such high speeds.

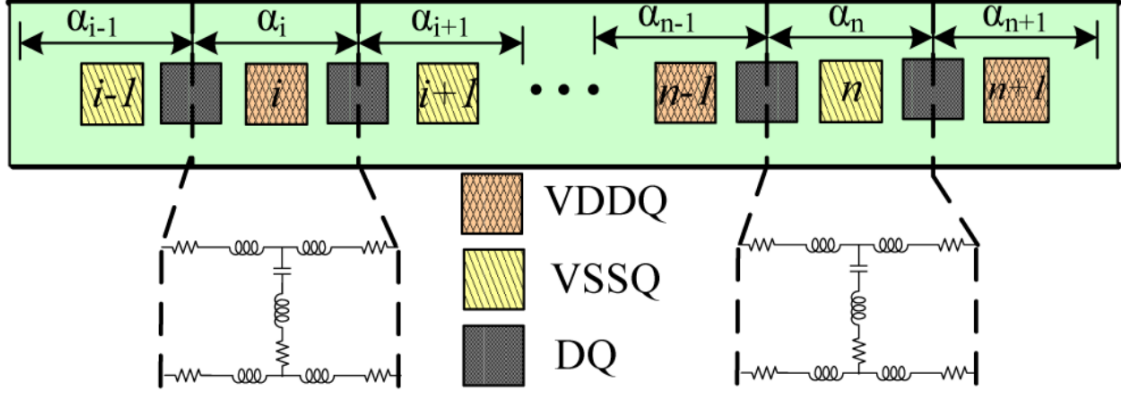


Figure 1: Pin layout of a memory chip

Figure 1 also shows how the memory can be subdivided in sections of length  $\alpha_i$  which are not necessarily equal. Each section can be modeled with a two port device structured as in Figure 2.

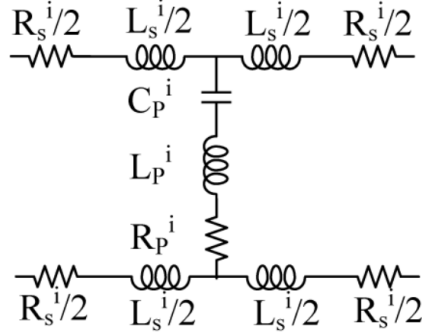


Figure 2: Model of a single section of a memory output

### 2.2 Parameters

Each section will thus have five characterizing parameters:  $R_s^i$ ,  $L_s^i$ ,  $R_p^i$ ,  $L_p^i$ ,  $C_p^i$ . These parameters depend on the overall Per-Unit-Length (PUL) parameters  $R_s$ ,  $L_s$ ,  $R_p$ ,  $L_p$ ,  $C_p$  and the length of each section  $\alpha_i$ . Given this model, the objective is to fit a set of real measurements in order to infer the memory PUL parameters which, in turn, will determine each section's parameters.

This model can also account for the skin effect with a simple change, which is integrated in the final solution. Specifically, it is possible to split the PUL parameters

$R_s$  and  $R_p$  into  $R_{s,dc}$ ,  $R_{s,ac}$ ,  $R_{p,dc}$  and  $R_{p,ac}$ . In order to map the PUL parameters to section-specific parameters it is possible to use:

$$\begin{aligned} R_s^i &= \alpha_i \max(R_{s,dc}, R_{s,ac}\sqrt{\omega}) \\ L_s^i &= \alpha_i L_s \\ R_p^i &= \frac{1}{\alpha_i} \max(R_{p,dc}, R_{p,ac}\sqrt{\omega}) \\ L_p^i &= \frac{1}{\alpha_i} L_p \\ C_p^i &= \alpha_i C_p \end{aligned}$$

## 2.3 Fitting

The unknowns of the problem are the PUL parameters - section lengths are to be manually measured or retrieved from layout information. The measured quantity is the two port  $Z$  matrix throughout a certain range of frequencies between two arbitrary ports of the device.

To be able to compare the real-world measurements to the simulated values we need to transform the device PUL parameters to its  $Z$  matrix between the same two ports of the measurements. This procedure is discussed in [subsection 2.4](#).

The fitting process will thus be reduced to a multi-variable optimization process, where the input is a vector representing the PUL parameters and the evaluated function will be an error function - the distance between the model and physical device. In this case:

$$U(\vec{p}) = \sum_{i=0}^N ||Z_s(\vec{p}, f_i) - Z_m(f_i)||^2$$

where  $\vec{p}$  is a vector representing the device PUL parameters,  $N$  is the total number of measured frequencies,  $f_i$  is the  $i$ -th measured frequency,  $Z_s$  is the impedance matrix obtained from the simulation and  $Z_m$  is the measured matrix.

## 2.4 From PUL Parameters to Z Matrix

In the previously mentioned  $U(\vec{p})$  formula it is necessary to compute  $Z_s(\vec{p}, f_i)$ , which means computing an impedance matrix between two fixed ports of the device starting from its PUL parameters. The passages are as follows:

1. Compute each section's parameters from PUL parameters as in [subsection 2.2](#)
2. Compute each section's ABCD matrix with

$$\text{ABCD} = \begin{bmatrix} 1 + z_i y_i & (z_i y_i + 2) z_i \\ y_i & 1 + z_i y_i \end{bmatrix}$$

where

$$\begin{aligned} z_i &= j\omega L_s^i + R_s^i \\ y_i &= \frac{j\omega C_p^i}{1 + j\omega R_p^i C_p^i - \omega^2 L_p^i C_p^i} \end{aligned}$$

3. Divide the layout into three groups by cumulating ABCD matrices: before port 1 ( $M_1 = \prod_{i=0}^{P_1} \text{ABCD}_i$ ), between port 1 and port 2 ( $M_2 = \prod_{i=P_1}^{P_2} \text{ABCD}_i$ ) and after port 2 ( $M_3 = \prod_{i=P_2}^R \text{ABCD}_i$ ).
4. Compute the  $Y$  matrix with

$$\begin{aligned}
Y(1, 1) &= \frac{M_1(2, 1)}{M_1(2, 2)} + \frac{M_2(2, 2)}{M_2(1, 2)} \\
Y(1, 2) &= -\frac{1}{M_2(1, 2)} \\
Y(2, 1) &= Y(1, 2) \\
Y(2, 2) &= \frac{M_2(1, 1)}{M_2(1, 2)} + \frac{M_3(2, 1)}{M_3(1, 1)}
\end{aligned}$$

5. Invert  $Y$  to get the desired  $Z$  matrix

## 2.5 Results

After finding the best  $\vec{p}$  that minimizes  $U(\vec{p})$ , the resulting PUL parameters are a good description of the physical device and the model error is limited. It is possible at this point to simulate a frequency sweep and compare the simulated model to its real world counterpart.

### 3 Implementation

The presented model is very simple yet shows a good accuracy on a wide band of frequencies. A possible improvement on the original discussion is the performance of its implementation, which is not public, but the authors mention ‘*the computation time is about 5 minutes at a quad core computer*’[1] on a MATLAB script[2].

This new implementation, titled **mem-pmod** (MEMory-PowerMODEL) is [open source](#) and based mainly on C++ plus the support of Python to generate graphics, aims at a better performance and a higher degree of reusability. The final result is able to execute the same operations on the same amount of data in about 50 s, or one fifth of the original amount of time.

#### 3.1 Features and Organization

Mem-pmod is developed in C++ and is focused on the optimization of the process. It uses multithreading together with the highly optimized Eigen library to maximize performance. It offers:

- Memory model fitting to experimental data as described from the paper
- Possibility to implement any generic fitting algorithm
- Model exporting and importing

It also includes some testing based on the GTest framework to check some of the available mechanisms.

[Figure 3](#) shows a typical flow of execution with the supported features. The

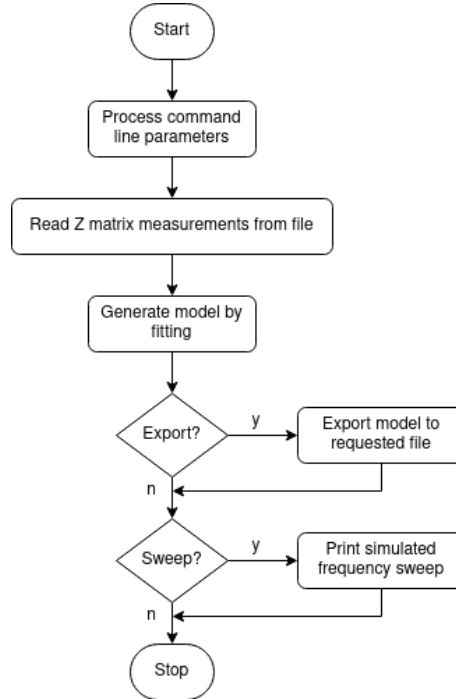


Figure 3: Execution flow

program is organized in modules which may be grouped in three categories:

- Models, which offer the infrastructure to build and interact with models
- Optimization, used by models to treat the fitting process as a generic optimization problem
- General control, which control the flow of the program and also offer some helpful, general purpose structures and functions

## 3.2 Reading Measurements

The measurements to be used as fitting data are stored in a text file and passed to the program as a command line parameter. Below is an example of a measurements file.

```

1 f Z11 Z12 Z21 Z22
2 +1.000000e+06 (+1.253488e+01,-3.163626e+01) (-4.890959e+00,-2.934532e+01) (-4.890959e
   +00,-2.934532e+01) (+1.253488e+01,-3.163626e+01)
3 +1.047129e+06 (+1.251280e+01,-3.031854e+01) (-4.869922e+00,-2.792886e+01) (-4.869922e
   +00,-2.792886e+01) (+1.251280e+01,-3.031854e+01)
4 +1.096478e+06 (+1.248878e+01,-2.906428e+01) (-4.847045e+00,-2.657249e+01) (-4.847045e
   +00,-2.657249e+01) (+1.248878e+01,-2.906428e+01)
5 +1.148154e+06 (+1.246268e+01,-2.787069e+01) (-4.822186e+00,-2.527343e+01) (-4.822186e
   +00,-2.527343e+01) (+1.246268e+01,-2.787069e+01)
6 +1.202264e+06 (+1.243432e+01,-2.673511e+01) (-4.795195e+00,-2.402906e+01) (-4.795195e
   +00,-2.402906e+01) (+1.243432e+01,-2.673511e+01)

```

Each line contains, in order, the measurement frequency in Hz and the four elements of the  $Z$  matrix in  $\Omega$ . Each element is a complex number, so it is represented as a pair (*real part*, *imaginary part*).

Measurements are stored in a specific structure:

```

1 struct Measurements {
2     std::vector<double> frequencies;
3     std::vector<Matrix2> Z;
4     unsigned port1;
5     unsigned port2;
6     std::size_t nsamples;
7 };

```

Which is initialized by

```

1 Measurements readMeasurements(const std::string &fname, unsigned
   port1, unsigned port2);

```

The two ports from which the measurements were taken are specified as command line arguments.

## 3.3 Geometric and Complete Models

The overall memory is modeled in two steps: there is a model for a single TSection and the MemoryModel holds an array of TSections plus the PUL parameters. To keep the fitting process as clean as possible, models are also divided between Geometric and Complete. Geometric models only keep geometric information, Complete models expand the concept and also have additional information such as electrical parameters.

Snippets for Geometric and Complete TSection

```

1 class GeometricTSection {
2 public:
3     GeometricTSection();

```



```

4     ...
5 protected:
6     double _length;
7 };
8
9 class TSection : private GeometricTSection {
10 public:
11     TSection();
12     ...
13 private:
14     LumpedParameters _parameters;
15 };

```

### Snippets for Geometric and Complete MemoryModel

```

1 template <std::size_t NPowerPorts>
2 class GeometricMemoryModel {
3 public:
4     GeometricMemoryModel();
5     ...
6 private:
7     std::array<GeometricTSection, NPowerPorts> _sections;
8 };
9
10 template<std::size_t NPowerPorts>
11 class MemoryModel : private GeometricMemoryModel<NPowerPorts> {
12 public:
13     MemoryModel();
14     ...
15 private:
16     std::array<TSection, NPowerPorts> _sections;
17     PULParameters _pul_parameters;
18 };

```

## 3.4 Fitting

The MemoryModel class also offers the interface for fitting and this is where the multithreading action happens. This class offers a static method called `fit()` which sets up and passes the necessary information to the optimization module and finally initializes a MemoryModel based on the input data.

```

1 static MemoryModel<NPowerPorts> fit(
2     const Measurements &measurements,
3     const std::array<double, NPowerPorts> lengths,
4     pmod::optimization::Algorithm method = pmod::optimization::
        Algorithm::CDESCENT);

```

The function `fittingError()`, also offered from the same class, is the function that computes the error function described in [subsection 2.3](#). Since this operation requires computing a Z matrix for each measured frequency, the measurements vector is divided in chunks which are computed in parallel through multithreading. The number of chunks depends on the hardware characteristics and is chosen to be as fast as possible. This is where the greatest improvement in performance happens.

```

1 static double fittingError(
2     const GeometricMemoryModel<NPowerPorts> &model,

```

```

3         const PULParameters &pul_parameters,
4         const Measurements &measurements,
5         bool enable_threading = true);

```

The optimization module is the one that runs the actual optimization algorithm but treats everything as a generic multidimensional optimization problem. It offers the templated function

```

1 template<std::size_t N>
2 Vector<N> optimize(Algorithm algorithm, std::function<double(Vector
    <N>)> function, Vector<N> x0, double threshold);

```

which automatically manages everything and allows for the choice of an optimization algorithm.

The original paper uses Powell’s method whose manual implementation is difficult and outside the current scope of this project. The currently implemented algorithm is the Coordinate Descent algorithm, which can be slower and gets trapped in local minima more easily, but this should not be a problem at the moment.

### 3.5 Importing and Exporting

As mentioned above, it is also possible to import and export memory models. This is done by specific functions implemented for the MemoryModel and TSection classes which internally manage all the necessary steps. The current implementation simply dumps all the relevant parameters in order in a text file and then the importer just reads them.

### 3.6 Results

The results discussed in this section will be numerically different from the ones of the paper, but it is evident how the trend is perfectly matching. This is because:

1. Measurements were manually generated since they are not available from the publication and they are not retrievable from the graphs because most functions overlap
2. Section lengths were arbitrarily set since the paper only mentions the overall memory length (4074  $\mu\text{m}$ ) but does not mention the single section lengths nor the memory model or its layout where to retrieve them from
3. The optimization algorithm is different

In any case, everything was set up to be as reasonably close as possible to realistic inputs and the results from the paper.

Figure 4 shows what a typical execution of mem-pmod looks like. The command line parameters specify, in order: the measurements file, the measurements ports, request a frequency sweep simulation with 1000 frequency samples (from 1 MHz to 10 GHz) and export the model to a file simply called `memory.model`. There was a total of 1000 measurement points in this run, the same amount as were used in the original paper. The obtained PUL parameters visible in Figure 4 are also shown in Table 1 for clarity. Finally, the obtained frequency sweep is matched against the

```

/mem-pmod$ build/mem_pmod data/measurements.txt 2 20 --fsweep 1000 -e memory
Fitting /mem-pmod/data/measurements.txt... Done! (48.2814 s)
Memory parameters:
RsdC = 6025.6  $\Omega$  / m
Rsac = 3.62817e-08  $\Omega$  / (m *  $\sqrt{\text{Hz}}$ )
Ls = 9.12011e-09 H / m
RpdC = 0.00691333  $\Omega$  * m
Rpac = 0  $\Omega$  / (m *  $\sqrt{\text{Hz}}$ )
Lp = 1.44544e-24 H * m
Cp = 1.31826e-06 F / m
Exporting model to "/mem-pmod/memory.model"...Done!
Saving frequency sweep on "/mem-pmod/fsweep.txt"... Done!

```

Figure 4: Example execution of mem-pmod. Execution time is highlighted in red.

Quantity	Value
RsdC	$5312.13 \Omega/\text{m}$
Rsac	$0.0238141 \Omega/(\text{m} * \sqrt{\text{Hz}})$
Ls	$9.18165 \times 10^{-9} \text{ H/m}$
RpdC	$0.00695145 \Omega * \text{m}$
Rpac	$1 \times 10^{-38} \Omega/(\text{m} * \sqrt{\text{Hz}})$
Lp	$4.74466 \times 10^{-28} \text{ H/m}$
Cp	$1.31426 \times 10^{-6} \text{ F/m}$

Table 1: Resulting PUL parameters from mem-pmod

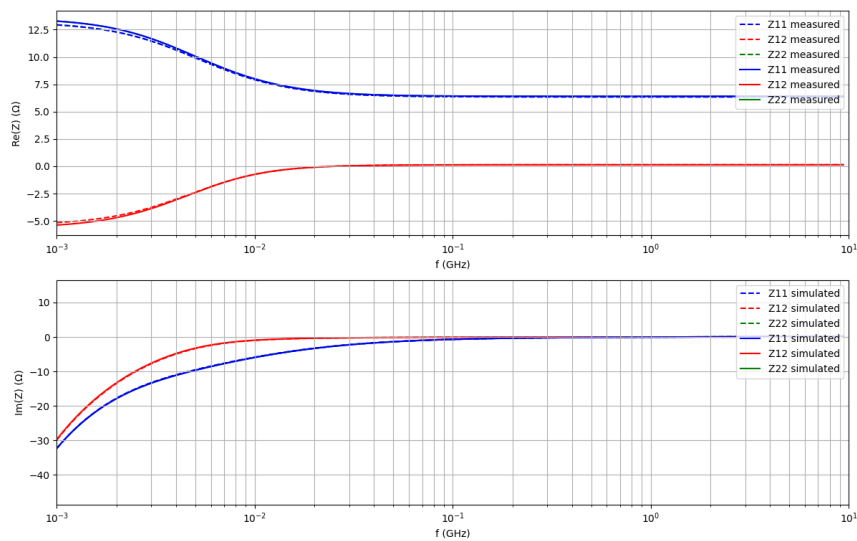
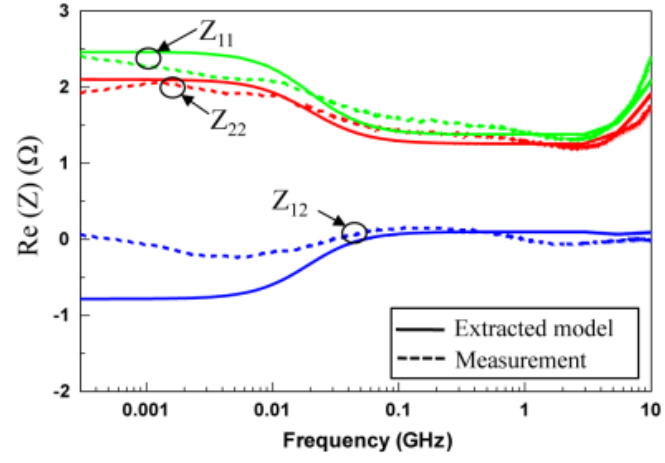
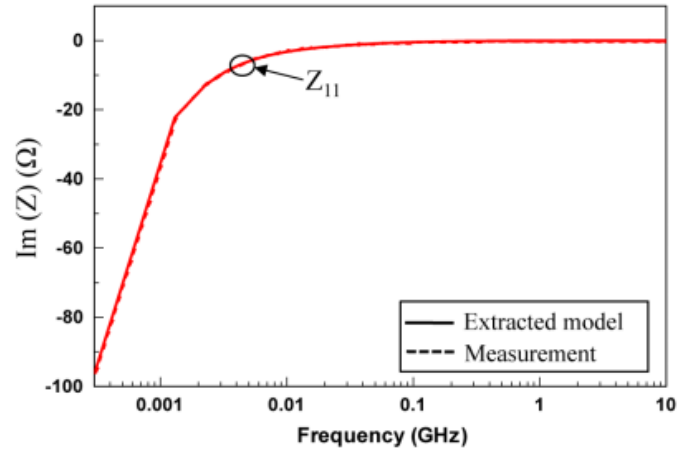


Figure 5: Results from mem-pmod against original measurements



(a)



(b)

Figure 6: Results from paper

original measurements with the help of a Python script and the matplotlib library. The results of this comparison are shown in [Figure 5](#).

Comparing [Figure 5](#) to [Figure 6](#), the one generated from the researchers, shows how even though the numerical values differ the general behavior is the same.

## References

- [1] H.-H. Chuang, C.-J. Hsu, J. Hong, C.-H. Yu, A. Cheng, J. Ku, and T.-L. Wu, “A broadband chip-level power-bus model feasible for power integrity chip-package codesign in high-speed memory circuits,” eng, *IEEE transactions on electromagnetic compatibility*, vol. 52, no. 1, pp. 235–239, 2010, issn: 0018-9375.
- [2] H.-H. Chuang, W.-D. Guo, Y.-H. Lin, H.-S. Chen, Y.-C. Lu, Y.-S. Cheng, M.-Z. Hong, C.-H. Yu, W.-C. Cheng, Y.-P. Chou, C.-J. Chang, J. Ku, T.-L. Wu, and R.-B. Wu, “Signal/power integrity modeling of high-speed memory modules using chip-package-board coanalysis,” *IEEE Transactions on Electromagnetic Compatibility*, vol. 52, no. 2, pp. 381–391, May 2010, issn: 1558-187X. DOI: [10.1109/TEM.2010.2043108](https://doi.org/10.1109/TEM.2010.2043108).