Politecnico di Torino

Project for Advanced Design for Signal Integrity and Compliance

A Power Bus-Model for High-Speed Memory Circuits

Alessandro Rocco Scisca
s276032

Contents

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

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 in recognizing the data which may be distorted at such high speeds.

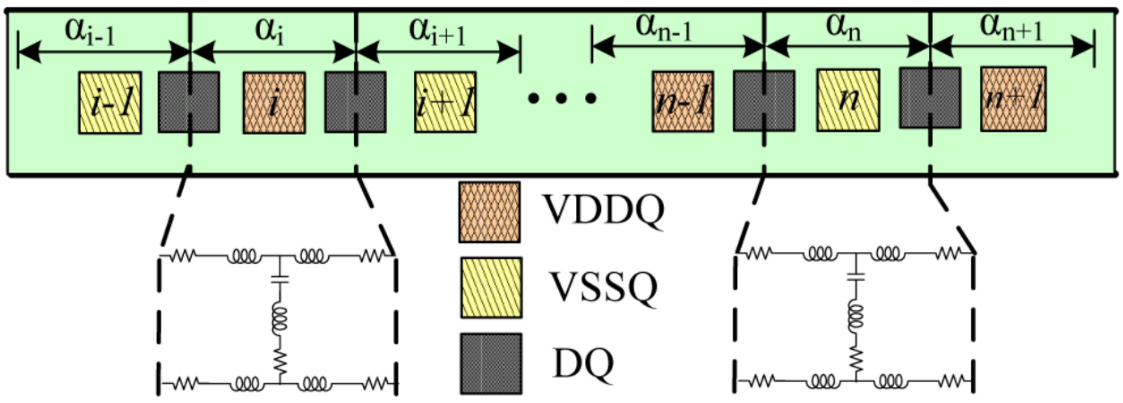


Figure 1: Pin layout of a memory chip

It is possible to split the power bus in sections - one for each power pin - of lengths α_i as shown in Figure 1. Each section can be modeled with a two port device strucured as in Figure 2.

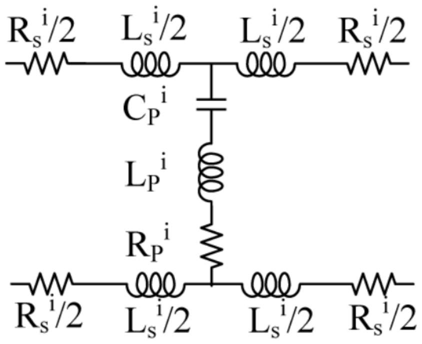


Figure 2: Model of a single section of a memory output

2.2 Parameters

Each section has five characterizing parameters: R_s^i, L_s^i, R_p^i, L_p^i, C_p^i which depend on the overall Per-Unit-Length (PUL) parameters R_s, L_s, R_p, L_p, C_p and the section length α_i. PUL parameters also take the skin effect into consideration by using a DC resistance and an AC resistance which depends on frequency. The effective resistance of a memory section will be the greatest of the two for a given frequency.

In general, section parameters are computed from PUL parameters with the following formulas:

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

Thanks to these formulas, it is possible to accurately represent the behavior of a power pin based on the power bus PUL parameters and the section lengths.

2.3 Fitting

While section lengths are known a-priori from layout information, PUL parameters can instead be determined by fitting real world electrical measurements. The measured quantity is the two port Z matrix between two arbitrary power pins. In order to compare the model to its real world counterpart, the procedure in subsection 2.4 explains how the Z matrix can be computed from the model parameters.

The fitting process is thus reduced to a multi-variable optimization process, where the unknowns are the PUL parameters and the error to the real world device is to be minimized. The error function $U\left(\vec{p}\right)$ can be expressed as:

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

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 order to obtain $U(\vec{p})$ it is necessary to compute $Z_s\left(\vec{p}, f_i\right)$, which means computing an impedance matrix starting from the model's PUL parameters.

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 ($M_1 = \prod_{i=0}^{\mathrm{P1}} \mathrm{ABCD}_i$), between port 1 and port 2 ($M_2 = \prod_{i=\mathrm{P1}}^{\mathrm{P2}} \mathrm{ABCD}_i$) and after port 2 ($M_1 = \prod_{i=\mathrm{P2}}^{R} \mathrm{ABCD}_i$).

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 which limits the model error. The simulation ran from the author returns the results shown in Figure 2.5, which show how close the simulation is to the actual measurements.

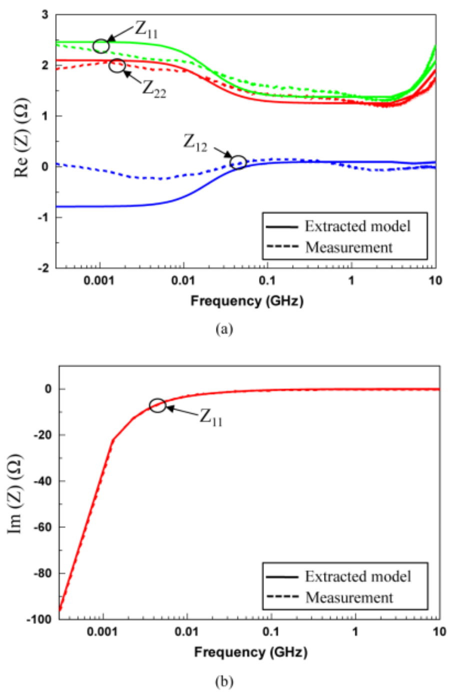


Figure 3: Results from paper

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. The original implementation is not publicly available, but it is reported that *'the computation time is about 5 minutes at a quad core computer'*[1] on a MATLAB script[2].

This proposed implementation, titled **mem-pmod** (MEMory-PowerMODel) is open source and based on C++ plus the support of Python to generate graphics. It aims at a better performance and a higher degree of reusability. The developed program can simulate and extract a model on the same amount of data in about 50 s, or one sixth 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 4 shows a typical flow of execution with the supported features. The

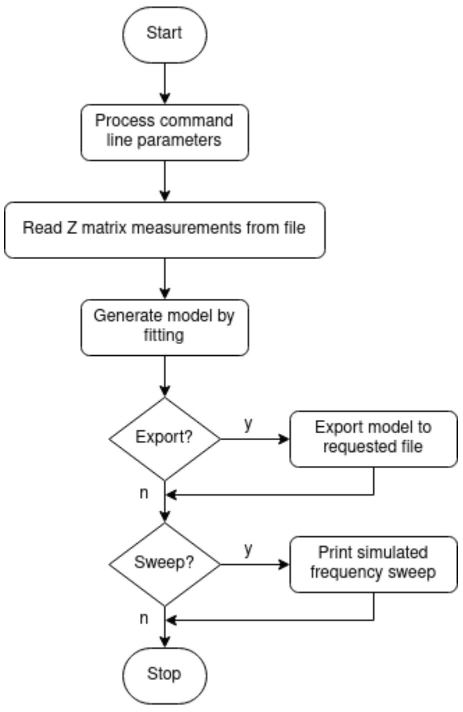

Start
Process command line parameters
Read Z matrix measurements from file
Generate model by fitting
Export?
y
Export model to requested file
n
Sweep?
y
Print simulated frequency sweep
n
Stop

Figure 4: 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.

```
f Z11 Z12 Z21 Z22
+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)
+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)
+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)
+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)
+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 Ω separated by spaces. Each matrix element is a complex number and it is written as (*real part*, *imaginary part*).

Inside the program, measurements are stored in a specific structure:

```
struct Measurements {
    std::vector<double> frequencies;
    std::vector<Matrix2> Z;
    unsigned port1;
    unsigned port2;
    std::size_t nsamples;
};
```

Which is initialized by the function

```
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 structures: there is a model for a single TSection and a MemoryModel which holds an array of TSections plus the PUL parameters. There is a distinction between Geometric and Complete models: geometric models only keep track of geometry information, complete models also contain electrical parameters and additional information. Below are some snippets of the various models in their two forms:

Snippets for Geometric and Complete TSection

```cpp
class GeometricTSection {
public:
    GeometricTSection();
    ...
protected:
    double _length;
};

class TSection : private GeometricTSection {
public:
    TSection();
    ...
private:
    LumpedParameters _parameters;
};
```

Snippets for Geometric and Complete MemoryModel

```cpp
template <std::size_t NPowerPorts>
class GeometricMemoryModel {
public:
    GeometricMemoryModel();
    ...
private:
    std::array<GeometricTSection, NPowerPorts> _sections;
};

template<std::size_t NPowerPorts>
class MemoryModel : private GeometricMemoryModel<NPowerPorts> {
public:
    MemoryModel();
    ...
private:
    std::array<TSection, NPowerPorts> _sections;
    PULParameters _pul_parameters;
};
```

3.4 Fitting

The MemoryModel class also offers the interface for fitting and this is where multithreading happens. This class offers a static method called `fit()` to initialize a model by fitting measurements using the optimization module.

```cpp
static MemoryModel<NPowerPorts> fit(
        const Measurements &measurements,
        const std::array<double, NPowerPorts> lengths,
        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 of a given model. This function is very expensive because it computes the Z matrix of a model for each measured frequency, but luckily this process can be parallelized with multithreading by splitting the job in chunks, each handled by a thread. The number of threads depends on hardware information in order to always maximize performance.

```
static double fittingError(
            const GeometricMemoryModel<NPowerPorts> &model,
            const PULParameters &pul_parameters,
            const Measurements &measurements,
            bool enable_threading = true)
```

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

```
template<std::size_t N>
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 a modified version of the Coordinate Descent algorithm, which can be slower and may get trapped in local minima more easily. The algorithm simply minimizes the function along each dimension in turn. In order to minimize the like-

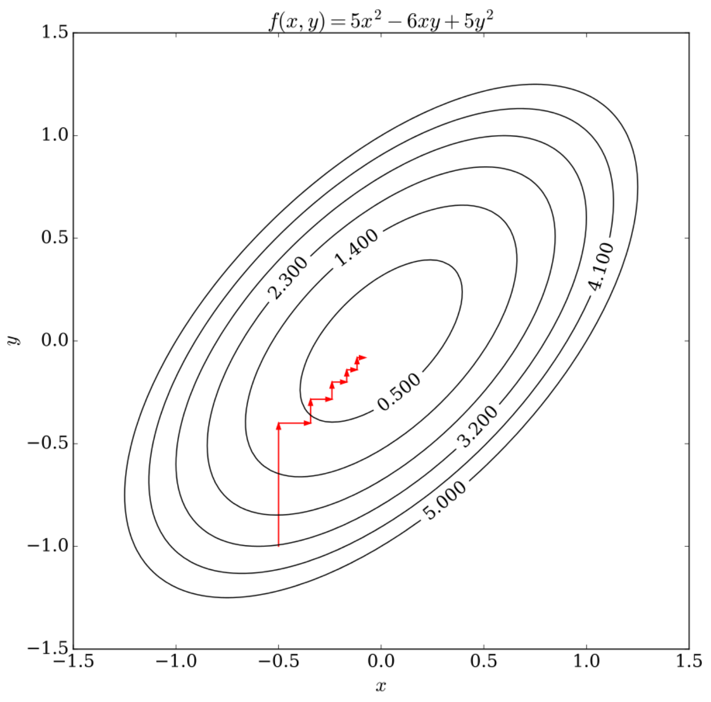


Figure 5: Behavior of the Coordinate Descent algorithm. Courtesy of Wikipedia[3]

lihood of getting trapped in local minima, the algorithm detects when stability is reached too soon and, in that case, adds some noise to free the point from the trap. The best minimum is always recorded and it is the one that is returned as a result.

3.5 Importing and Exporting

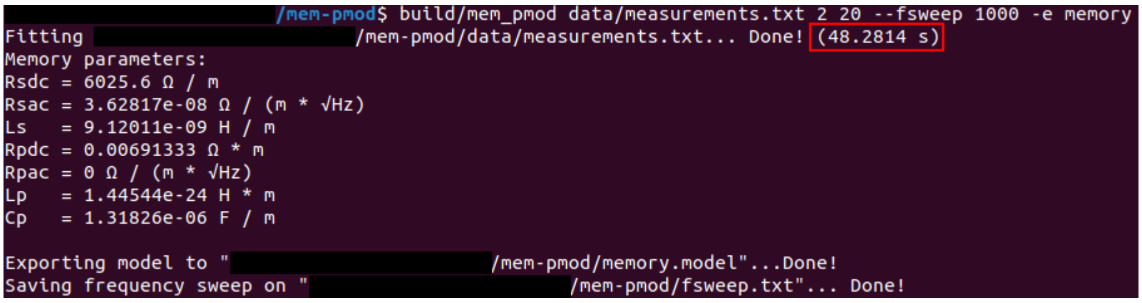
As mentioned above, it is also possible to import and export memory models. This is done by specific functions implemented for the MemoryModel and TSection classes which internally manage all the necessary steps. The current implementation simply dumps all the relevant parameters in order in a text file 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 used in the paper are not available nor easily accessible, so they were manually generated
2. The paper only specifies the overall memory length (4074 µm), so it was assumed all sections were equally long. This assumption is likely imprecise
3. The optimization algorithm is different

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



```
[redacted]/mem-pmod$ build/mem_pmod data/measurements.txt 2 20 --fsweep 1000 -e memory
Fitting [redacted]/mem-pmod/data/measurements.txt... Done! (48.2814 s)
Memory parameters:
Rsdc = 6025.6 Ω / m
Rsac = 3.62817e-08 Ω / (m * √Hz)
Ls   = 9.12011e-09 H / m
Rpdc = 0.00691333 Ω * m
Rpac = 0 Ω / (m * √Hz)
Lp   = 1.44544e-24 H * m
Cp   = 1.31826e-06 F / m

Exporting model to "[redacted]/mem-pmod/memory.model"...Done!
Saving frequency sweep on "[redacted]/mem-pmod/fsweep.txt"... Done!
```

Figure 6: Example execution of mem-pmod. Execution time is highlighted in red.

Figure 6 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 to export the model to a file 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 6 are also shown in Table 1 for clarity. Finally, the obtained frequency sweep is matched against the original measurements with the help of a Python script and the matplotlib library. The results of this comparison are shown in Figure 7.

Comparing Figure 7 to Figure 2.5, the one generated from the researchers, shows how even though the numerical values differ the general behavior is the same.

Quantity	Value
Rsdc	$5312.13\,\Omega/\mathrm{m}$
Rsac	$0.023\,814\,1\,\Omega/(\mathrm{m}*\sqrt{\mathrm{Hz}})$
Ls	$9.181\,65\times10^{-9}\,\mathrm{H/m}$
Rpdc	$0.006\,951\,45\,\Omega*\mathrm{m}$
Rpac	$1\times10^{-38}\,\Omega/(\mathrm{m}*\sqrt{\mathrm{Hz}})$
Lp	$4.744\,66\times10^{-28}\,\mathrm{H/m}$
Cp	$1.314\,26\times10^{-6}\,\mathrm{F/m}$

Table 1: Resulting PUL parameters from mem-pmod

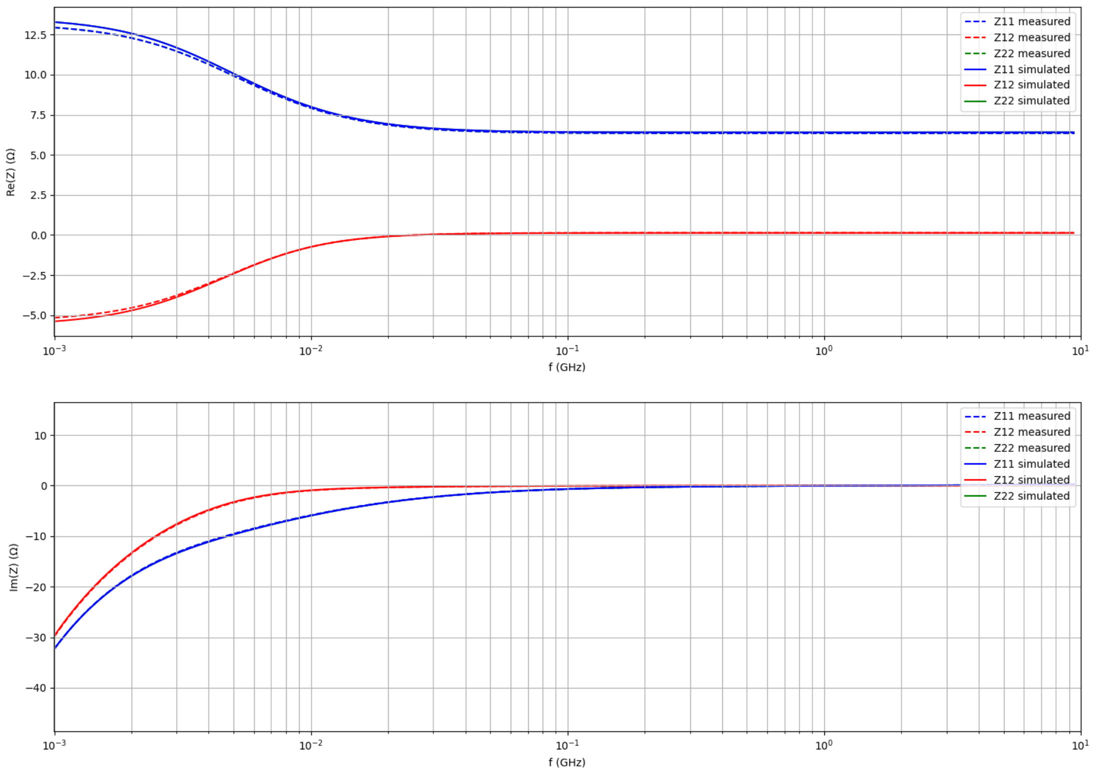


Figure 7: Results from mem-pmod against passed measurements

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/TEMC.2010.2043108.

[3] Wikipedia, *Coordinate descent*. [Online]. Available: https://en.wikipedia.org/wiki/Coordinate_descent#/media/File:Coordinate_descent.svg.