Chapter 1

Simulation Tool

In the previous chapters, all the models are introduced. The last step is to create a complete simulation tool including them. A short description of the model categories are reported below.

1. YIG Logical Model: the blocks are described by their logic. The model is quite simple, and it is used to generate reference solutions for other models.

2. YIG (100 nm) Behavioral Model: the blocks are modelled taking into account some physical and geometric characteristics, but the damping is not included. Its execution time is much lower than the Physical ones. For this reason, this model can be used as a fast verification of circuits.

3. YIG (100 nm) Physical Model: it is the Behavioral Model with the damping and other geometric characteristics. The model shows a simulation result as close as possible to the micromagnetic simulation one.

4. YIG (30 nm) Physical Model: it is the same model of the previous one, but implemented by the 30 nm YIG technology.

The idea is to integrate all these categories. In order to increase the flexibility of the tool, some dedicated "user interface" scripts are built and described in the following paragraphs.

1.1 Organization of scripts

A scheme of the organization of the scripts that compose the tool is reported in Figure 1.1.

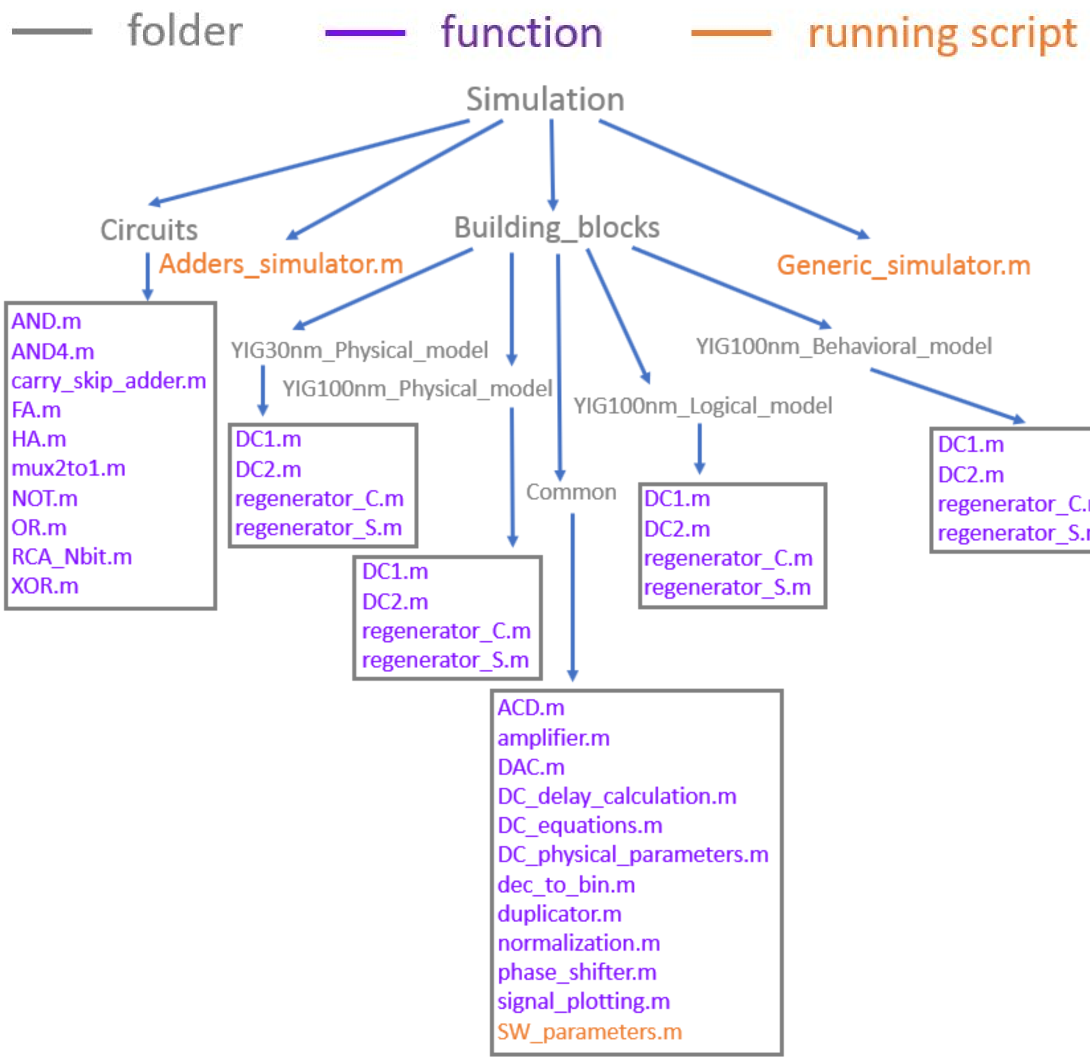


Figure 1.1: Organization of the scripts

The starting folder is called *Simulation*, which contains the entire tool. The scripts are divided into two categories and contained in two folders: *Circuits* and *Building_blocks*.

1. **Building_blocks**: the folder contains the building blocks of all the model categories. Each category is characterized by four blocks (functions): *DC*1*.m*, *DC*2*.m*, *regenerator_C.m* and *regenerator_S.m*. The user can select the correct folder to instantiate the building blocks according to the model category

choice. The folder *Common* contains the common scripts among all the categories.

2. **Circuits**: the folder contains all the models of the modelled circuits. Every circuit is expressed as a function, and its implementation is not dependent on the chosen model category.

In the same folder *Simulation*, two user interface scripts, *Adders_simulator.m* and *Generic_simulator.m*, are implemented.

1.2 Adder simulator

1.2.1 Description

In order to simulate different technologies adders with a high number of simulations, a dedicated user interface script, *Adders_simulator.m*, is implemented. The script allows the user to simulate adders, to evaluate their accuracy and to report the results.

1.2.2 Data conversion

In this section, some useful functions for simulation are presented.

The numbers used in simulation are pseudorandom, they can be generated by a MATLAB command "randi". This command generates the pseudorandom integers with a uniform discrete distribution. It is possible to set the range of these integers, that depends on the input of circuit. However, in digital field the numbers are represented in binary form. To do this, the first function, the so called decimal-binary converter (DBC), is created: the function receives a decimal number and the number of bits N, and it returns the number represented in N-bit unsigned representation. The unsigned numbers are compatible with Logical model circuits, but not with the Behavioral model that requires the SW form. To solve this, the next function, the digital-analog converter (DAC), is implemented as in Figure 1.2. To compare the simulation results, the output SWs are converted in digital numbers. This operation is performed by an analog-digital converter (ADC): the function

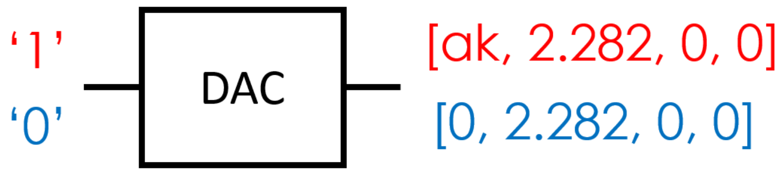


Figure 1.2: Digital-analog converter. ak is the SW amplitude chosen for logic '1'.

evaluates the normalized power of the SW according to the separation thresholds that are shown in Figure 1.3.

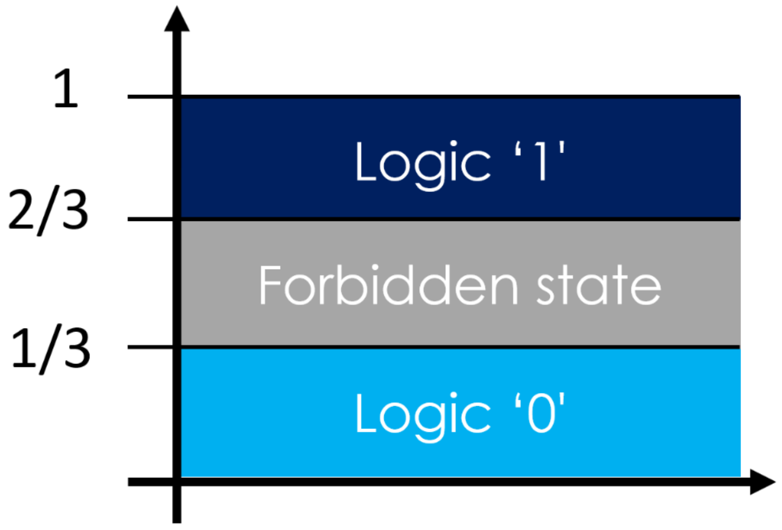


Figure 1.3: Logic values separation.

The function "normalization" is created to compute the normalized power of the SWs with respect to the logic '1' power: it receives the SW amplitude and returns the percentage of normalized power.

1.2.3 Structure of code

In this paragraph, the building sections of the *Adders_simulator.m* are presented.

1. **Simulation setting:** the initial simulation parameters should be set by the user.

```
Nbit = 32;  % parallelism of adder
```

```
N_simulation = 50; % number of simulations
result_rep_file = 'RCA_32bit_50sim.txt'; % result
    report file_name
result_rep_flag = 0; % =1 to write the output powers
    (in terms of normalized power) in "the
    result_rep_file".txt
err_search_flag = 0; % =1 to search and to display
    the combinations that generated wrong outputs
plot_range_flag = 1; % =1 to set ranges for logic '1'
     and '0' in the final plot, using the following
    parameters:
logic1_sup = 101;
logic1_inf = 99;
logic0_sup = 1;
logic0_inf = -1;
```

2. **Choosing of simulation model category and adder type:** in the first part, the program asks the user to choose the model category for simulation and selects the correct building blocks folder according to the choice. In the second part, the program asks the user to choose the adder type (RCA or CSA) for simulation.

```
model=0;
while model ~= [1,2,3,4]
model = input('Choose one model category from the
    following list:\n 1) YIG (100 nm) Behavioral
    Model \n 2) YIG (100 nm) Physical Model \n 3) YIG
     (30 nm) Physical Model \n 4) QUIT \n');
end
if model~=4  % if model==4, the program terminates

switch model % to get the correct folder for building
     blocks
    case 1
```

```
        model_path = 'Building_blocks/
            YIG100nm_Behavioral_model';
    case 2
        model_path = 'Building_blocks/
            YIG100nm_Physical_model';
    case 3
        model_path = 'Building_blocks/
            YIG30nm_Physical_model';
end
addpath(model_path)
addpath('Building_blocks/Common')
addpath('Circuits')

adder=0;
while adder ~= [1,2]
    adder = input('\nChoose one simulation adder from
         the following list: \n  1) Ripple-Carry
        Adder \n  2) Carry-Skip Adder \n');
end
```

3. **Pseudorandom inputs generation:** the decimal inputs are generated using the command ”randi”.

```
A = zeros(N_simulation,Nbit);
B = zeros(N_simulation,Nbit);
C = zeros(N_simulation,1);

for i = 1:N_simulation
    A_dec = randi([0,2^Nbit-1],1,1); % random inputs
        generation
    B_dec = randi([0,2^Nbit-1],1,1);
```

To evaluate the accuracy of the circuit, in general $N\_simulation > 1$, in this case the steps 3-8 are repeated for $N\_simulation$ times.

4. **Decimal-binary conversion:** the decimal numbers are converted in binary ones, that are represented in N-bit unsigned representation.

```
A(i,:) = dec_to_bin(A_dec,Nbit);      %
    std_logic_vector(N-1 downto 0)
B(i,:) = dec_to_bin(B_dec,Nbit);      %
    std_logic_vector(N-1 downto 0)
C(i,:) = randi([0,1],1,1);            % std_logic
```

   The converted numbers will be used by Logical Model circuits.

5. **Digital-analog conversion:** using the DAC function, it is possible to obtain the SW vectors for the circuit.

```
in_A = DAC(A(i,:),model);
in_B = DAC(B(i,:),model);
in_C = DAC(C(i,:),model);
```

6. **Simulation:** in this section, the adder is instantiated and simulated according to the user choice.

```
if adder == 1
    output = RCA_Nbit(in_A,in_B,in_C,Nbit,model);
elseif adder == 2
    output = carry_skip_adder(in_A, in_B, in_C,
        Nbit, model);
end

% in order to evaluate their amplitudes, and at
    the "accuracy evaluation and report" section
    we will normalize them
output_sig(i,:)= output(:,1)'; % amplitudes
```

   the *output_sig* collects the output SW amplitudes, and it will be used to evaluate the accuracy of the circuit.

7. **Reference solution generation:** the same adder (in terms of parallelism) using the Logical Model is instantiated to calculate the correct binary outputs, that are used to verify the simulated adder ones.

```
    model_t = model;
    model = 0; % logical model
    addpath('Building_blocks/YIG100nm_Logical_model')
        % model category change
    exact_output(i,:) = RCA_Nbit(A(i,:)',B(i,:)',C(i
        ,:),Nbit,model);
    addpath(model_path)
    model = model_t;
```

8. **Outputs comparison:** this section checks the simulation result. Since the reference outputs are digital, the outputs of the circuit, which are the SWs, are converted into digital ones using the ADC function.

```
    % analog to digital conversion of the simulation
        solution
    output_bin(i,:) = ADC(output,model);
end % for i = 1:N_simulation

correct = 0;
if output_bin == exact_output  % comparison
    display('the simulation result is correct')
    correct = 1;
else
    display('the simulation result is not correct')
end
```

9. **Error search:** this is a hidden section. It is activated when the user sets the *err_search_flag* to 1 and the simulation result is not correct (*correct* = 0).

```
if err_search_flag == 1 % you can set this flag at
    the beginning
    if correct == 0
        m = 1;
        for n=1:N_simulation % search of the errors
            if exact_output(n,:) ~= output_bin(n,:)
                % we need to decompose n into (i,j)
                err_A(m,:) = A(n,:);
                err_B(m,:) = B(n,:); % A and B are
                    std_logic_vectors
                err_C(m,:) = C(n,:);
                m = m+1;
            end
        end
        display('The following combinations (A and B)
             generated wrong outputs:')
        m = m-1;
        for i=1:1:m % error combination display
            fprintf('\nCombination %d: \n \t A = ',i)
            fprintf('%d',err_A(i,:))
            fprintf('\n \t B = ')
            fprintf('%d',err_B(i,:))
            fprintf('\n \t C = ')
            fprintf('%d',err_C(i,:))
        end
        fprintf('\n')
    end
end
```

The goal of this section is to search the combinations (A,B) that generated the wrong outputs.

10. **Accuracy evaluation:** in order to evaluate the accuracy of the circuit, the output SW amplitudes collected in section 6 (the variable *output_sig*) are normalized here using the function "normalization".

```
% normalization of the output bits of every simulation
normalized_output = normalization(output_sig,model);
```

The matrix *normalized_output* contains all the outputs (in terms of the power normalized with respect to the logic '1') of every simulation.

11. **Accuracy storing:** if the user sets the *result_rep_flag* to 1, this section is active and it writes all the elements of the matrix *normalized_output* into a file.

```
if result_rep_flag == 1 % report of the simulation result
    f = fopen(result_rep_file,'w');
    for i= 1:N_simulation
        for j = 1:Nbit+1
            t = normalized_output(i,j);
            if t >= 100
                fprintf(f,'%3.4f  ',t);
            elseif t >= 10
                fprintf(f,'%3.4f   ',t);
            else
                fprintf(f,'%3.4f    ',t);
            end
        end
        fprintf(f,'\n\n');
    end
    fclose(f);
end
```

12. **Accuracy plot:** the matrix *normalized_output* is plotted as in the **??**.

```
figure
hold on
for i=1:N_simulation
```

```
    scatter([1:1:Nbit+1], normalized_output(i,end
        :-1:1), 'filled')
end
axis([1, Nbit+1, -10, 110])
xlabel('bit','FontSize',20)
ylabel('Normalized power (%)','FontSize',20)
if plot_range_flag == 1
    plot([1:1:Nbit+1],logic1_sup*ones(1,Nbit+1))
    plot([1:1:Nbit+1],logic1_inf*ones(1,Nbit+1))
    plot([1:1:Nbit+1],logic0_sup*ones(1,Nbit+1))
    plot([1:1:Nbit+1],logic0_inf*ones(1,Nbit+1))
end
hold off
```

1.2.4 Run a simulation

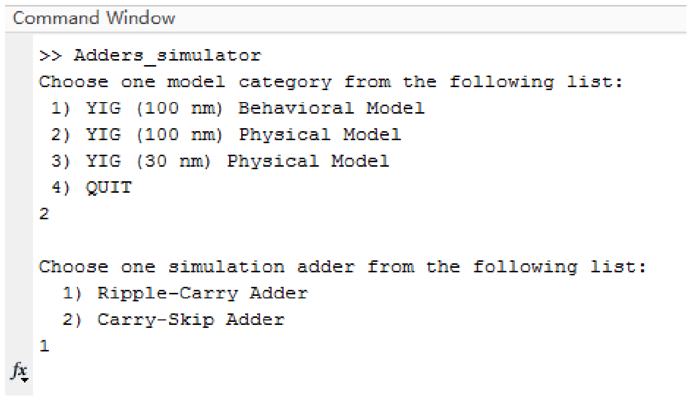
In order to show the steps to run a simulation, a simple example is reported.

- Set the simulation parameters.

```
Nbit = 8;  % parallelism of adder
N_simulation = 5; % number of simulations
result_rep_file = 'RCA_8bit_5sim.txt'; % result
    report file_name
result_rep_flag = 1; % =1 to write the output powers
    (in terms of normalized power) in "the
    result_rep_file".txt
err_search_flag = 1; % =1 to search and to display
    the combinations that generated wrong outputs
plot_range_flag = 1; % =1 to set ranges for logic '1'
     and '0' in the final plot, using the following
    parameters:
logic1_sup = 101;
logic1_inf = 99;
```

```
logic0_sup = 1;
logic0_inf = -1;
```

- Run the script.
- Choose the model category and the adder type as shown in Figure 1.4.



Command Window

```
>> Adders_simulator
Choose one model category from the following list:
 1) YIG (100 nm) Behavioral Model
 2) YIG (100 nm) Physical Model
 3) YIG (30 nm) Physical Model
 4) QUIT
2

Choose one simulation adder from the following list:
  1) Ripple-Carry Adder
  2) Carry-Skip Adder
1
```

Figure 1.4: Choosing of the model category and the adder type.

- Obtain the result as shown in Figure 1.5.

1.3 Generic simulator

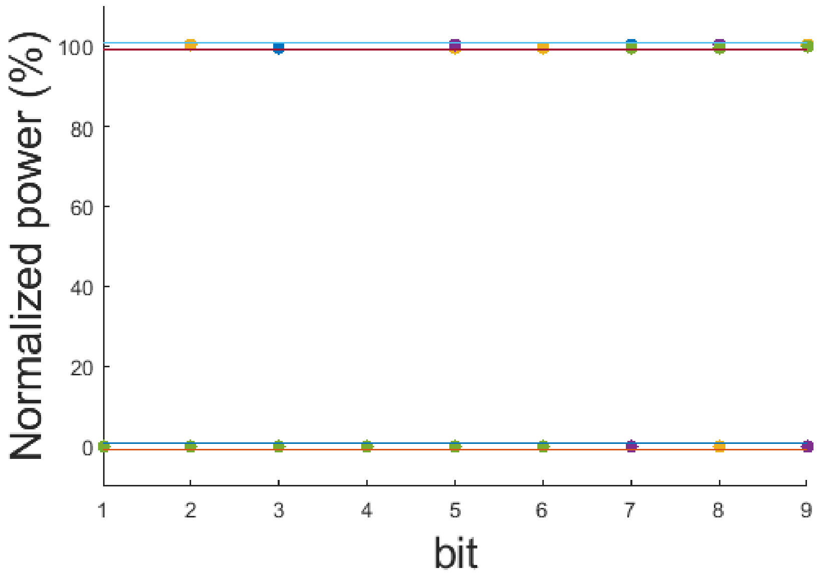
1.3.1 Description

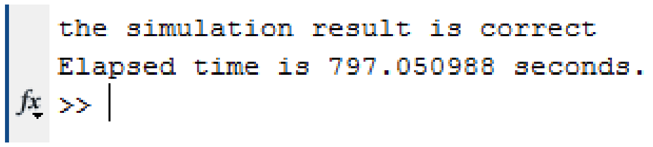
The tool also allows the user to simulate other simple circuits as OR, HA, AND, etc. To do this, another user interface script, the *Generic_simulator.m*, is created. This script allows the user to simulate every modelled circuit with some optional input parameters.

1.3.2 Structure of code

In this paragraph, the building sections of the *Generic_simulator.m* are described.

1. **Simulation setting:** the user has to set some initial parameters for the simulation.





```
the simulation result is correct
Elapsed time is 797.050988 seconds.
>>
```

Figure 1.5: Simulation result.

```
A_bin = [0,1;1,1];
B_bin = [1,1;0,1];
C_bin = [0;1];
D_bin = [1 1;0 1];

Nbit = 2;     % it is used by RCA and CSA (parallelism
    ). For the Carry-Skip Adder, the Nbit must be a
    multiple of 4, which is a constraint of the CSA
    model
opt_parameters = {'out_signal_plot'}; % optional
    parameters, it can be empty
titleFontSize = 25;   % title FontSize of the plots
axisFontSize = 13;     % axes FontSize of the plots
labelFontSize = 20;   % labels FontSize of the plots
```

```
legendFontSize = 14;  % legend FontSize of the plots
```

A_bin, *B_bin*, *C_bin* and *D_bin* are the inputs of the circuit (see the code of item 3). Some circuits do not need all the inputs. It is possible to run more than one simulation. In that case the inputs are matrices: every row is a *std_logic_vector*(*MSB downto LSB*) that will be used for a single simulation. The *Nbit* is used if the circuit is RCA or CSA. The *opt_parameters* is an array of cells that contains some optional input parameters of the simulated circuit. For example, the option *'out_signal_plot'* requests the program to display and to plot the output SWs of the circuit. Every circuit owns the *out_signal_plot*, and the HA has other options, which will be discussed afterwards. The user must set all these parameters of simulation in this section.

2. **Choosing of model category:** the program asks the user to choose the model category for the simulation and selects the correct building blocks folder according to the choice.

```
model=0;
while model~=1 && model~=2 && model~=3 && model~=4
model = input('Choose one model category from the
    following list:\n 1) YIG (100 nm) Behavioral
    Model \n 2) YIG (100 nm) Physical Model \n 3) YIG
     (30 nm) Physical Model \n 4) QUIT \n');
end
if model~=4

switch model
    case 1
        model_path = 'Building_blocks/
            YIG100nm_Behavioral_model';
    case 2
        model_path = 'Building_blocks/
            YIG100nm_Physical_model';
    case 3
```

```
        model_path = 'Building_blocks/
            YIG30nm_Physical_model';
end
addpath(model_path)
addpath('Building_blocks/Common')
addpath('Circuits')
```

3. **Choosing of simulation circuit:** the program asks the user to choose the simulation circuit from a list.

```
circuit=0;
while circuit ~= [1,2,3,4,5,6,7,8,9,10]
    fprintf('\nChoose one simulation circuit from the
         following list:');
    fprintf('\n  1) AND(A,B)');
    fprintf('\n  2) AND4(A,B,C,D)');
    fprintf('\n  3) %d-bit Carry_skip_adder(A,B,C=
        Carry_in)',Nbit);
    fprintf('\n  4) FA(A,B,C)');
    fprintf('\n  5) HA(A,B)');
    fprintf('\n  6) Mux2to1(A,B,C=sel)');
    fprintf('\n  7) NOT(A)');
    fprintf('\n  8) OR(A,B)');
    fprintf('\n  9) %d-bit RCA(A,B,C=Carry_in)',Nbit)
        ;
    circuit = input('\n  10) XOR(A,B) \n');
end
```

4. **Simulation:** the program instantiates and simulates the chosen circuit.

```
size_A = size(A_bin);
N_simulation = size_A(1);
analyzed_figures = 0;
for ii = 1:N_simulation
```

```
    switch circuit
        case 1
            A = DAC(A_bin(ii,:),model);
            B = DAC(B_bin(ii,:),model);
            fprintf('Simulation %d:  A = "',ii)
            fprintf('%d',A_bin(ii,:))
            fprintf('", B = "')
            fprintf('%d',B_bin(ii,:))
            fprintf('"\n')
            AND_out = AND(A, B, model, opt_parameters
                {:})
            figures = findobj(0,'Type','Figure');
            New_figures = 0;
            for j=1:max(size(figures))-
                analyzed_figures % for each plot
                ax = figures(j).CurrentAxes;
                ax.Title.String = 'Simulation ' +
                    string(ii);
                ax.Title.FontSize = titleFontSize;
                ax.Legend.FontSize = legendFontSize;
                ax.XAxis.FontSize = axisFontSize;
                ax.YAxis.FontSize = axisFontSize;
                ax.XLabel.FontSize = labelFontSize;
                ax.YLabel.FontSize = labelFontSize;
                New_figures = New_figures + 1;
            end
            analyzed_figures = analyzed_figures +
                New_figures;
        .
        .
        .

    end
```

```
end

end % if model~=4
```

In this code only the first switch case is reported, the other ones are identical apart from the simulation circuit.

1.3.3 Run a simulation

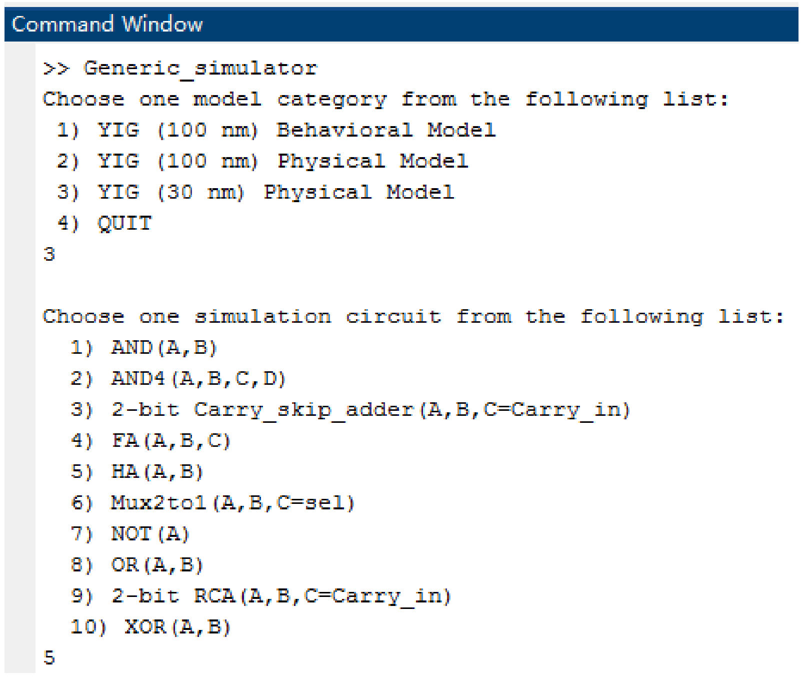
In order to show the steps to run a simulation, a simple example of simulation is reported below.

- Set the simulation parameters.

```
A_bin = [0;1];
B_bin = [1;1];
C_bin = [1];
D_bin = [0];

Nbit = 2;   % it is used by RCA and CSA (parallelism)
    . For the Carry-Skip Adder, the Nbit must be a
    multiple of 4, which is a constraint of the CSA
    model
opt_parameters = {'out_signal_plot'}; % optional
    parameters, it can be empty
titleFontSize = 25;   % title FontSize of the plots
axisFontSize = 13;     % axes FontSize of the plots
labelFontSize = 20;   % labels FontSize of the plots
legendFontSize = 14;  % legend FontSize of the plots
```

- Run the script.
- Choose the model category and the circuit as shown in Figure 1.6.
- Obtain the result as shown in Figure 1.7.


Command Window

```
>> Generic_simulator
Choose one model category from the following list:
 1) YIG (100 nm) Behavioral Model
 2) YIG (100 nm) Physical Model
 3) YIG (30 nm) Physical Model
 4) QUIT
3

Choose one simulation circuit from the following list:
  1) AND(A,B)
  2) AND4(A,B,C,D)
  3) 2-bit Carry_skip_adder(A,B,C=Carry_in)
  4) FA(A,B,C)
  5) HA(A,B)
  6) Mux2to1(A,B,C=sel)
  7) NOT(A)
  8) OR(A,B)
  9) 2-bit RCA(A,B,C=Carry_in)
  10) XOR(A,B)
5
```

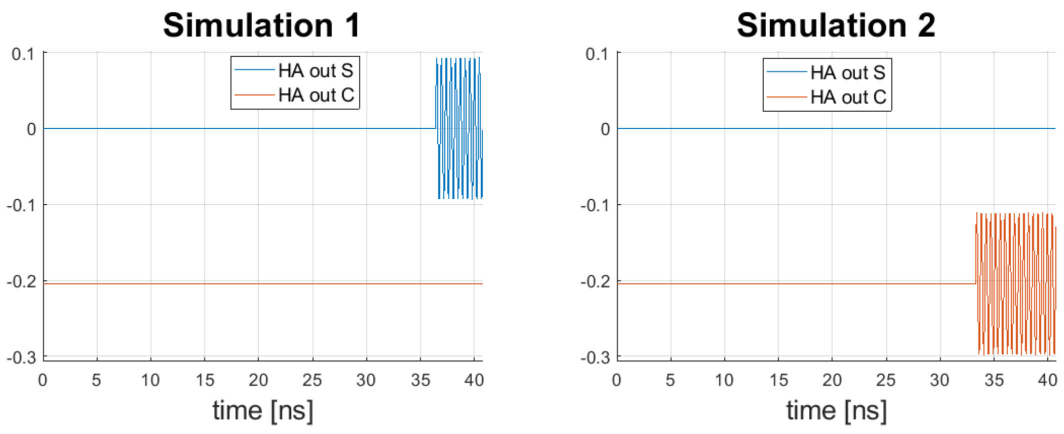
Figure 1.6: Choosing of the model category and the simulation circuit.

1.3.4 Half-adder analyzer

One application of the *Generic_simulator.m* is to use it as a half-adder analyzer exploiting the optional input parameters. In the HA simulation, it is possible to check all the building blocks. In the following, the optional input parameters are described.

In the previous paragraph, a very useful optional parameter, the *out_signal_plot*, is introduced. Since the HA can send the *out_signal_plot* to each building block (DC1, DC2, regenerator S and regenerator C), it needs some additional information to distinguish the blocks. For example, it is possible to use $DC1\_out\_signal\_plot$ for the DC1, and the $DC2\_out\_signal\_plot$ for the DC2. Let's define the syntax of a generic optional input parameter:

$$optional\_parameter = tag + argument \tag{1.1}$$



```
Simulation 1:  A = "0", B = "1"

 HA: out_S = u(t-t0) a sin(2 π f t + φ), where t0 = 3.640456e+01 ns,
 a = 9.303447e-02, f = 2.290000e+00 GHz and φ = 0, normalized power = 1.000741e+02%

 HA: out_C = u(t-t0) a sin(2 π f t + φ), where t0 = 3.328047e+01 ns,
 a = 1.078437e-04, f = 2.290000e+00 GHz and φ = 0, normalized power = 1.344695e-04%

HA_S =

    0.0930    2.2900         0   36.4046


HA_C =

    0.0001    2.2900         0   33.2805
Simulation 2:  A = "1", B = "1"

 HA: out_S = u(t-t0) a sin(2 π f t + φ), where t0 = 3.640456e+01 ns,
 a = 2.354463e-05, f = 2.290000e+00 GHz and φ = 0, normalized power = 6.409406e-06%

 HA: out_C = u(t-t0) a sin(2 π f t + φ), where t0 = 3.328047e+01 ns,
 a = 9.310655e-02, f = 2.290000e+00 GHz and φ = 0, normalized power = 1.002293e+02%

HA_S =

    0.0000    2.2900         0   36.4046


HA_C =

    0.0931    2.2900         0   33.2805
```

Figure 1.7: Simulation result.

where tag indicates the reference building block, argument is the optional operation as the *out_signal_plot*.

The possible tags are:

- DC1_ $\Rightarrow$ directional coupler 1.
- DC2_ $\Rightarrow$ directional coupler 2.
- regS_ $\Rightarrow$ regenerator S.
- regC_ $\Rightarrow$ regenerator C.

For example, *regC_out_signal_plot* is the optional parameter to display and to plot the output SW of the regenerator C.

All the optional parameter combinations (tag + argument) of the HA are reported in Table 1.1 and Table 1.2.

	DC1_	**DC2_**	**regC_**	**regS_**
Lc_avg	X X	X X	X X	X X
dispersion_curves	X X X	X X X	X X X	X X X
out_signal_plot	X X X	X X X	X X X	X X X

Table 1.1: Half-adder optional parameters table 1: these parameters are set to obtain some plots or displays on output. The blue 'X' indicates the validity of the optional parameter for YIG (100nm) Behavioral Model, the red one for YIG (100nm) Physical Model and the green one for YIG (30nm) Physical Model.

The following explanations are valid for both the tables (Table 1.1 and Table 1.2).The first column contains the arguments list, and the first row contains the four possible tags. It is possible to find the symbol 'X' in some cells of combination tag + argument, in these cases the HA possesses the corresponding optional parameter (composed according to the Equation 1.1). An additional information of the 'X' is its color: the blue indicates the validity of the 'X' for YIG (100nm) Behavioral Model, the red for YIG (100nm) Physical Model and the green for YIG (30nm) Physical Model.

In the following list, the description and notes of each argument are reported.

	DC1_	**DC2_**	**regC_**	**regS_**
thickness	X X X	X X X	X X X	X X X
width	X X X	X X X	X X X	X X X
Lw	X X X	X X X	X X X	X X X
gap	X X X	X X X	X X X	X X X
gain_in				X X
gain_interm				X X
gain_out			X X	X X
limitation	X X X	X X X	X X X	X X X
dx	X X	X X	X X	X X
external_field	X X X	X X X	X X X	X X X

Table 1.2: Half-adder optional parameters table 2: these parameters are set to change some default values. The blue 'X' indicates the validity of the optional parameter for YIG (100nm) Behavioral Model, the red one for YIG (100nm) Physical Model and the green one for YIG (30nm) Physical Model.

- **Lc_avg**: *Lc_avg* displays the accumulated phase ($\Delta\phi$) between two modes along the directional coupler, the average coupling length ($L_{c,avg}$) and the output power partition ($P_{out1}/(P_{out1} + P_{out2})$). Example: opt_parameters = {'DC2_Lc_avg'};

- **dispersion_curves**: *dispersion_curves* plots the unshifted dispersion curves (symmetric and antisymmetric) and displays the coupling length (L_c) and the output power partition ($P_{out1}/(P_{out1} + P_{out2})$) according to the curves. Example: opt_parameters = {'DC1_dispersion_curves'};

- **out_signal_plot**: *out_signal_plot* displays all the output spin-waves of the block in the form $u(t-t_0)asin(2\pi ft+\Phi)$ and plots them in the same graph. Example: opt_parameters = {'regC_out_signal_plot'};

- **thickness**: *thickness* must be followed by a number to change the default value of the waveguide thickness (expressed in nm) of the directional coupler. Example: opt_parameters = {'DC1_thickness',50};

- **width**: *width* must be followed by a number to change the default value of the waveguide width (expressed in nm) of the directional coupler. Example: opt_parameters = {'regS_width',40};

- **Lw**: *Lw* must be followed by a number to change the default value of the coupled length (expressed in nm) of the diretional coupler. Example: opt_parameters = {'DC2_Lw',2500}; Note that a small change for the regenerator S: in the categories YIG (100nm) Physical Model and YIG (30nm) Physical Model, the regenerator S is composed by two directional couplers, in these cases the argument *Lw* is replaced by *Lw*1 and *Lw*2 to distinguish them. Example: opt_parameters = {'regS_Lw1',1500,'regS_Lw2',1200};

- **gap**: *gap* must be followed by a number to change the default value of the gap (expressed in nm) of the diretional coupler. Example: opt_parameters = {'regC_gap',70};

- **gain_in**: *gain_in* must be followed by a number to change the default value of the input amplifier gain of the regenerator S. Note that only the regenerator S owns this amplifier. Example: opt_parameters = {'regS_gain_in',7};

- **gain_interm**: *gain_interm* must be followed by a number to change the default value of the intermediate amplifier gain of the regenerator S. Note that only the regenerator S owns this amplifier. Example: opt_parameters = {'regS_gain_interm',5};

- **gain_out**: *gain_out* must be followed by a number to change the default value of the output amplifier gain of the regenerator. Example: opt_parameters = {'regC_gain_out',2.5};

- **limitation**: *limitation* must be followed by a number to change the default value of the integral interval "limitation" of the diretional coupler. Example: opt_parameters = {'DC1_limitation',0.74};

- **dx**: dx must be followed by a number to change the default value of the discretization resolution (expressed in nm) of the diretional coupler. Example: opt_parameters = {'DC2_dx',20};

- **external_field**: $external\_field$ must be followed by a number to change the default value of the external magnetic field (expressed in mT) of the diretional coupler. Example: opt_parameters = {'regC_external_field',5};

It is called half-adder analyzer, because it is possible to analyze each building block using these optional parameters. For example, setting $DC2\_out\_signal\_plot$, $DC2\_Lc\_avg$ and $DC2\_dispersion\_curves$ it is possible to get all the information about the DC2.

The previous tables (Table 1.1 and Table 1.2) show the HA optional parameters to drive the building blocks, but the HA also owns other ones for itself. In the following, these parameters are presented.

- **out_signal_plot:** when this argument is expressed without tag, it is applied to the outputs of the HA.

- **HA_without_regS:** the HA simulation is made replacing the regenerator S by an amplifier to bring the output S close to the required level.

- **HA_without_regC:** the HA simulation is made replacing the regenerator C by an amplifier to bring the output C close to the required level.

The $HA\_without\_regS$ and the $HA\_without\_regC$ can be also applied to the OR gate. In that case, the OR gate instantiates its HAs passing these two parameters.

In Table 1.3, the possible optional parameters about every circuit are listed.

AND	out_signal_plot
AND4	out_signal_plot
CSA	out_signal_plot
FA	out_signal_plot
HA	out_signal_plot, $HA\_without\_regS$, $HA\_without\_regS$ and all the combinations of Table 1.1 and Table 1.2
Mux2to1	out_signal_plot
NOT	out_signal_plot
OR	out_signal_plot, $HA\_without\_regS$ and $HA\_without\_regS$
RCA	out_signal_plot
XOR	out_signal_plot

Table 1.3: Optional parameters of the circuits.

Chapter 2

Performance Analysis

To evaluate the performances of the magnonic RCA, several components are modelled. These scripts are organized as shown in Figure 2.1.

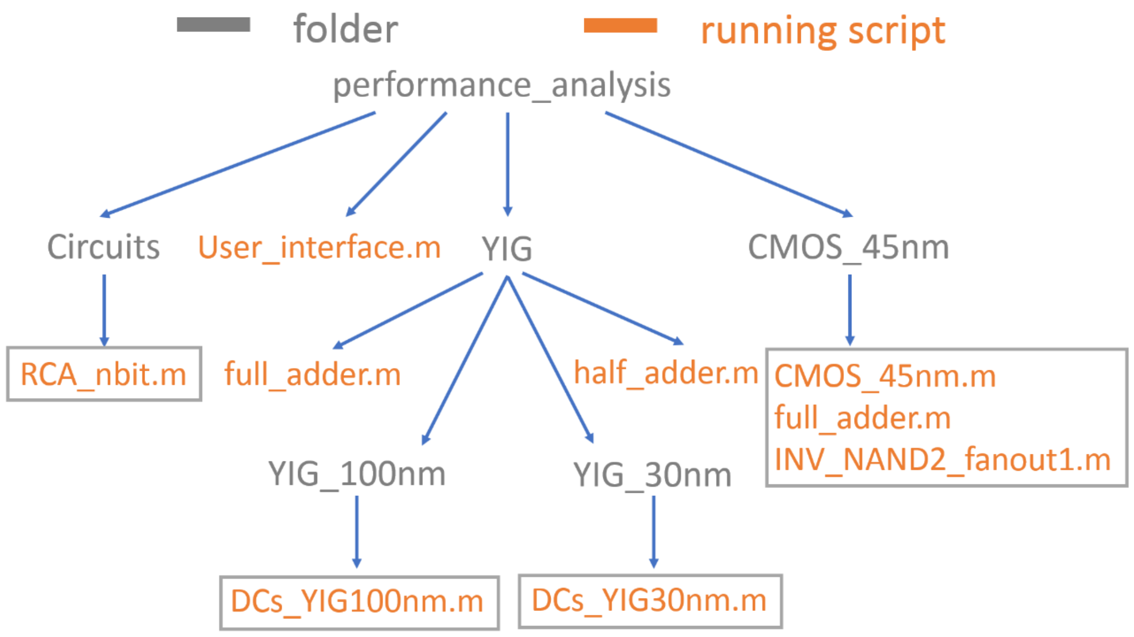
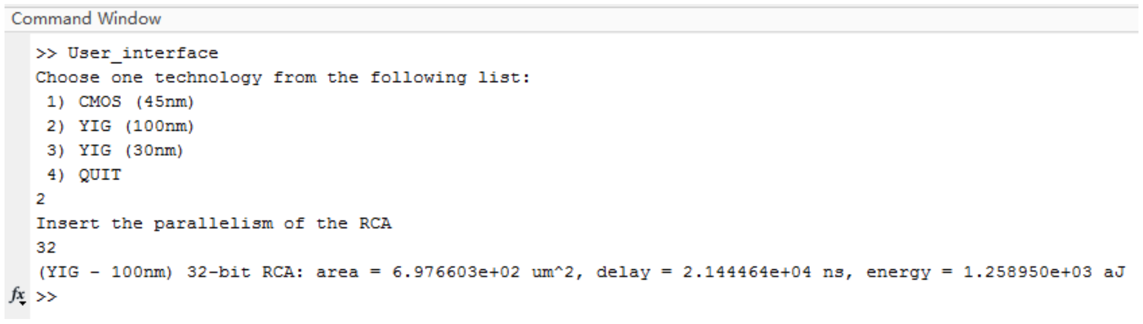


Figure 2.1: Organization of the scripts

The starting folder is called $performance\_analysis$, in which a user interface script, $User\_interface.m$, is created. The program asks the user to choose the technology and the parallelism of the RCA, and it gives the RCA performances in output (see Figure 2.2).



```
Command Window
>> User_interface
Choose one technology from the following list:
 1) CMOS (45nm)
 2) YIG (100nm)
 3) YIG (30nm)
 4) QUIT
2
Insert the parallelism of the RCA
32
(YIG - 100nm) 32-bit RCA: area = 6.976603e+02 um^2, delay = 2.144464e+04 ns, energy = 1.258950e+03 aJ
>>
```

Figure 2.2: $User\_interface.m$ running.

After the program termination, it is possible to find the performances of every sub-component (HA, FA etc...) that builds the RCA from "Workspace" window.

The program calls the $RCA\_nbit.m$ from folder $Circuits$, which calls its sub-component, the FA, according to the technology choice. If the chosen technology is CMOS 45 nm, the $full\_adder.m$ is the one inside the $CMOS\_45nm$. If the technology is YIG, the FA and the HA are modelled in the same way for both technologies (100 nm and 30 nm). However, the basic directional couplers parameters ($DCs\_YIG100nm.m$ and $DCs\_YIG30nm.m$) depend on the chosen YIG technology.