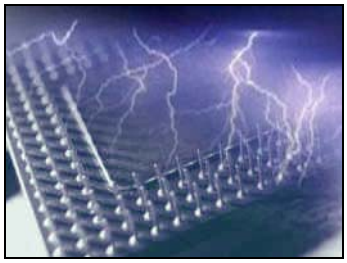
Vatios 1.0

Programmer's Manual



Juan Antonio Victorio Ferrer
University of Zaragoza

Table of contents

1. Presentation

Vatios is a free simulator, written in C that can be used to get power and energy consumption values from super-scalar processors. It implements the same architecture as Wattch and not only provides the same functionality, but also gives more flexibility and new interesting features.

Vatios has been developed trying to allow the advanced-user to modify it with few effort. In order to do that, it's recommended to be familiar with the SimpleScalar simulator.

As Vatios and Wattch are based on SimpleScalar, if you have a modified simulator based on SimpleScalar you can easily add power/energy prediction, only including some headers, their correspondent files in the directory and some code in your simulator.

Moreover, if you have a simulator that isn't based on SimpleScalar, you can generate a file with the same structure and required information as the dump files that `sim-vatios` generates and then, you can process it with `power-vatios`, calculating the predictions.

This manual has been written to help you understanding how Vatios works and will guide you if you try to modify Vatios. If you are not familiar with the simulator or you haven't read the user manual, we strongly recommend doing that first.

Basically, there are 4 ways you would like to modify Vatios.

- Add another technology point.
- Add a new power model for a unit (a new way of calculating the peak power of a unit)
- Add a new unit, with its power model.
- Add the Vatios library to an existing SimpleScalar based simulator.
- Use the Vatios library to calculate power/energy based on the statistics generated by a non-SimpleScalar based simulator.

2. Simulator Software Architecture

This simulator is based on SimpleScalar, thus it contains the same files as SimpleScalar, but there are 2 files that have been modified:
- `options.c` has been slightly modified to allow dump use/access info.
- `sim-outorder.c` has been renamed to `sim-vatios.c` and we have added the code to measure the use of the units and dump that use to a file.

We have replaced the CACTI 1.0 tool with CACTI 4.2, therefore, you will find a directory called `cacti4_2.` We use CACTI to calculate subbanking of cache memories and to directly calculate the power of cache-like arrays.

We have created a new executable `power-vatios` created from `power-vatios.c` that uses files dumped by `sim-vatios` to calculate power/energy consumption (allowing the user to modify power models, frequency, technology…)

We have created `access_vatios.c` and `access_vatios.h`. In these files we define the number of units that the processor is made of, and we define functions to count the use of units. These functions are used by `sim-vatios.c`

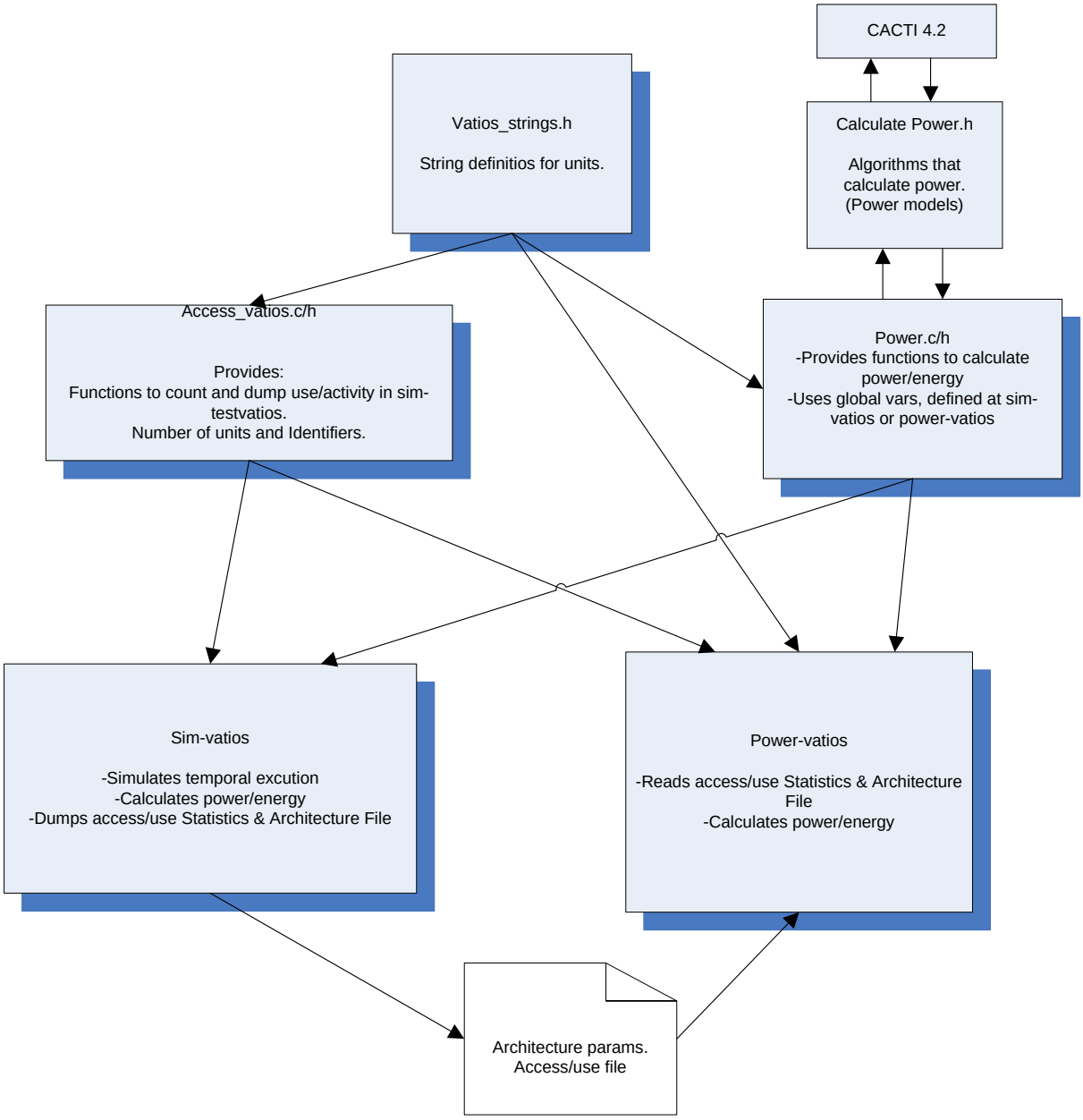
In `vatios_strings.h` we have written the strings used to refer to the units and used to display information to the user.

At the file `calculatePower.h` we have written the code (power models) to calculate the peak power of a unit, which will be escalated to calculate the actual energy consumption. In this file we call CACTI when necessary. This file is included in `power.c`.

In `power.c` and `power.h` you will find generic functions to calculate the power of a unit. You will find the calculate function, which calculates and displays all the energy/power results and is called by `power-vatios` and by `sim-vatios`.

Once you have compiled the simulator, 2 executable files will be generated: `sim-vatios` and `power-vatios.`

This diagram shows the simulator software architecture:



3. Adding another technology point

A technology point is the definition of some parameters determined by your fabrication process. Vatios comes with some technology point defined, but if you want to make simulations in a technology point not defined in Vatios, you can add the information needed to simulate in that point.

Wattch's author developed some functions to calculate the power of several kinds of structures (for example RAM, CAM…). These functions use some variables, whose initial values depend on some parameters: CSCALE, RSCALE, LSCALE, ASCALE, VSCALE, VTSCALE, SSCALE, GEN_POWER_SCALE, which are factors relative to the 0.80um fabrication process that represent the wire capacitance, wire resistance, length , area, voltage, threshold voltage, sense voltage and power.

So, let's see how we would add a new technology point:

In the file `power.c` there is a function called:

```
void init_vatios_tech_params(double techPoint)
```

At the beginning of this function, there's a `switch` clause. You have to add your technology point there. As an example, you can see how the tech point for 90nm is defined (Numbers are not real!). The 8 scaling factors you have to define for your tech point are relative to the 800nm fabrication process.

```
case 180:
     CSCALE    =(19.7172)/* wire capacitance scaling factor */;
     RSCALE    =(20.0000)/* wire resistance scaling factor */;
     LSCALE    =0.2250   /* length (feature) scaling factor */;
     ASCALE    =(LSCALE*LSCALE)    /* area scaling factor */;
     VSCALE    =0.4      /* voltage scaling factor */;
     VTSCALE   =0.5046   /*threshold voltage scaling factor */;
     SSCALE    =0.85     /* sense voltage scaling factor */;
     GEN_POWER_SCALE = 1;
     break;
case 90:
     CSCALE    =(35.232)/* wire capacitance scaling factor */;
     RSCALE    =(37.5000)/* wire resistance scaling factor */;
```

```
     LSCALE    =0.1150   /* length (feature) scaling factor */;
     ASCALE    =(LSCALE*LSCALE)    /* area scaling factor */;
     VSCALE    =0.3      /* voltage scaling factor */;
     VTSCALE   =0.5046   /*threshold voltage scaling factor */;
     SSCALE    =0.64     /* sense voltage scaling factor */;
     GEN_POWER_SCALE = 1;
     break;
case 400:
…
…
```

4. Adding new power models

First of all we want to introduce what a power model is. In Vatios, a power model is simply an algorithm that calculates a unit's peak power. For one unit, can be defined several power models, for example, we can have a cache and two power models, the first one that uses the functions defined by Wattch, and the second one that uses the latest version of CACTI.

When you are using your simulator, you can select the model for the unit introducing this parameter in the command line:

`-unit_name:model power_model (Example: –dcache:model 3)`

You may want to add new power models to improve or to give different choices when calculating the energy of a unit. The power model calculates the power of the unit and returns a `double` value.

First of all, you have to write the C code that calculates the maximum power that this unit can consume in one cycle. There's a file in the main program directory whose name is `calculatePower.h`. For every unit there's a switch instruction and at least "case 0:" and "case 1:" and "case 2". What you have to do is to add another "case #:"where # is the number you are going to use for this model.
Let's see an example:

This would be the code after adding my new model.

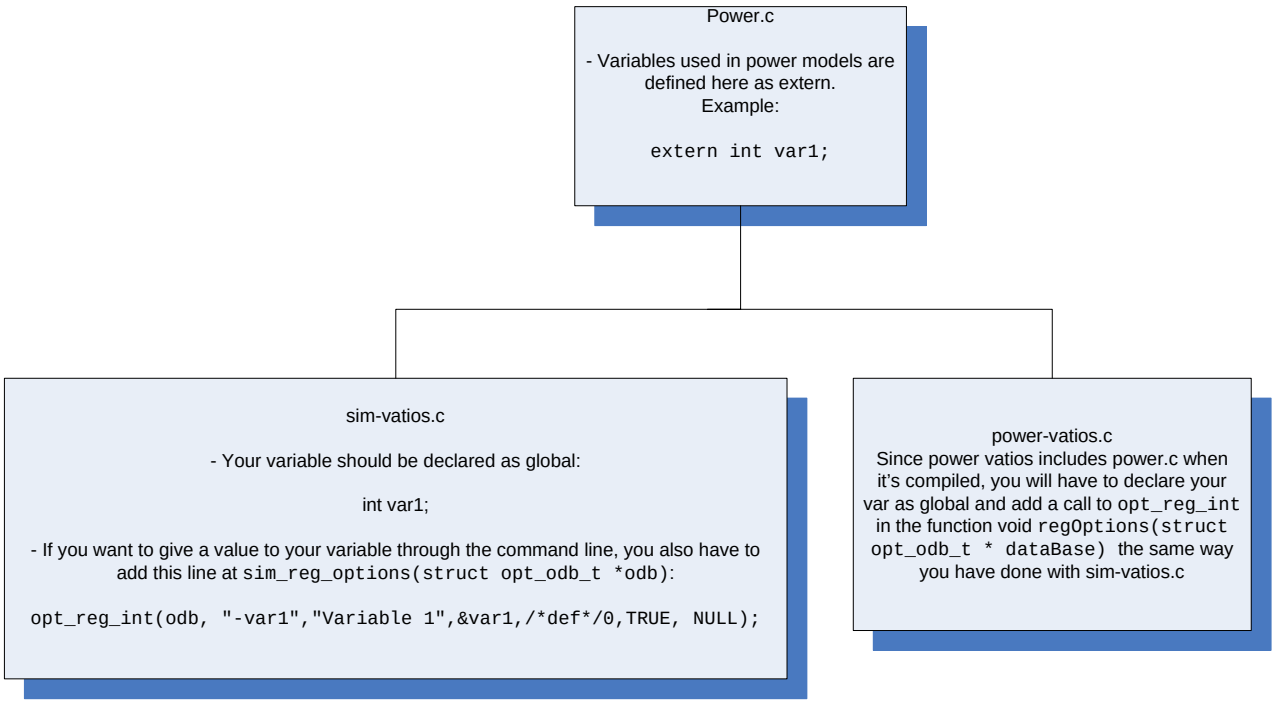
```
switch(energyModels[RENAME]){
     case 0: /*Don't' count this unit*/
          power.powerAFIndependent[RENAME] = 0.0;
          break;
     case 1:/* Default Wattch model*/
          power.powerAFIndependent[RENAME] = rat_power +
          dcl_power + inst_decoder_power;
          break;
     case 2:
          power.powerAFIndependent[RENAME] =
               userPowerAFIndependent[RENAME];
          break;
     case 3:
          power.powerAFIndependent[RENAME] = myNewRenamePower();
          break;

     default:
          fprintf(stderr,"Invalid energyModel for %s\n",
          varUnitNames[RENAME]);
          exit(-1);
}
```

Inside your algorithm you will need some architectural parameters, for example, if you are modelling the power of a cache, you will probably need the cache size, the cache assoc, the block size, of if you are modelling the ALU unit, you will probably need the number of ALUs.

If the parameter you need is present in another power model, you will be able to read the value and that would be enough (and you don't have to read the rest of this chapter), but if you creating a more detailed power model, it could be possible that you need a new parameter. Then you have to understand how architectural parameters are defined and used in Vatios.

This is the variable architecture in `sim-vatios`



Now, we are going to give a more detailed explanation of this.

Due to constraint in the options.c/h library, the architectural variables should be declared as global variables in `sim-vatios.c` and you have to register them, like other parameters in the function:

```
void sim_reg_options(struct opt_odb_t *odb)
```

Make sure that the `opt_reg_XXXXX` call has the print parameter set to TRUE, because otherwise, it won't be dumped to the config file. Note: `XXXX` can be `double, uint, int, string`,… If you need the full list of functions, please read `options.h`

You have to do the same at `power-vatios.c`. You have to declare the variables as global in a section where you can find the comment:

```
/****************** GLOBAL VARIABLE DECLARATIONS *************/

…
…
int opcode_length = 8;
int inst_length = 32;
int ruu_issue_width = 4;
int RUU_size = 8;
int myNewVar = 10;
…
…
```

You also have to register the variables in a function called:

```
void regOptions(struct opt_odb_t * dataBase)

{

  opt_reg_int(dataBase, "-myNewVar ","My new Var",&myNewVar
,MY_NEW_VAR_DEFAULT_VAL,/* print */TRUE, /* format */NULL);
```

Be careful, there are a lot of `opt_reg_XXXXX` functions, so use the appropriate one depending on the type of your new variable.
Finally, you have to declare your variable as extern at the beginning of the power.c file.

```
…
…
extern int data_width;
extern int res_ialu;
extern int res_fpalu;
extern int res_memport;
extern int myNewVar;
…
…
```

That's all. If you have some trouble, try to understand the compiler messages and how are defined other variables used in other power models. One of the easiest power models is the one for the IALU, you can follow this unit's model as an example. This model uses a parameter called `res_ialu`.

5. Adding a new unit, with its power model.

Adding a new unit means that you are going to model a processor with another piece of hardware, for example, we can think of adding a L3 cache structure. First of all, we have to modify our simulator to include this L3 cache structure. If we only do this, this would be transparent for Vatios. Therefore, we also have to tell Vatios that we are going to calculate the power/energy for this unit, and how we are going to do this.

We have divided this process in these steps:

- Declaring the unit
- Adding auxiliary strings
- Counting the number of accesses
- Creating the power models
- Adding the call to the function that calculates power/energy.

5.1. Declaring the Unit

First of all, you will have to modify the `access_vatios.h` file. You will have to increment the `NUM_UNITS` constant, you will also have to add a new identifier in the first `enum` for example:

```
#define NUM_UNITS 15
…,LSQ_WAKEUP,LSQ_PREG,MY_UNIT};
```

The name used here is not relevant. Be careful to use an unused identifier.

5.2. Adding auxiliary strings

Once you have done that, you will have to add new strings for your unit at the file `vatios_strings.h`, this strings will be used by the user to refer to this unit, and by the simulator to display information.

```
char * varUnitNames[] = {
"-rename",
"-bpred",
…
…
```

```
"-lsq_wakeup",
"-lsq_preg",
"-my_unit"
};

char * comentars[] = {
"Rename Accesses",
"Bpred Accesses",
…
…
"Lsq wakeup Accesses",
"Lsq preg Accesses",
"My unit Accesses"
};
```

Attention: The order of these strings IS relevant. It should match with the order of the units declared in the enum at the previous section.

5.3. Counting the number of accesses

Wattch (and `sim-vatios`) models the Clock-gating technique to calculate energy consumption. The different clock-gating techniques are explained at Wattch's Report. To use this technique to calculate energy consumption, we need to know how many times one unit is accessed. Therefore, during the temporal execution of the simulator, we have to count the number of accesses. Specifically, we have to count the number of times that one unit is used every cycle, and then, increment one position of a vector.

Our goal is to create a vector for each unit, which has this structure:

Number of cycles that this unit has been accessed …

`[0 times, 1 time, 2 times, …]`

If you think that your unit is going to be used 30 o more times per cycle, please increment this constant defined at `access_vatios.h`:

```
#define MAX_ACCESS_CYCLE 30
```

To achieve this goal these steps are necessary:

There are some things you have to add at `access_vatios.c`

First of all, as global variables:

```
…
counter_t ialu_access;
```

```
counter_t falu_access;
counter_t resultbus_access;
counter_t my_unit_access;
```

These variables should be also declared as `extern` in the `sim-vatios.c` file, as you are going to modify them in that program to count accesses.

```
At the function clear_access_stats()
…
…
  ialu_access=0;
  falu_access=0;
  resultbus_access=0;
  my_unit_access=0;
```

`clear_access_stats` is called at the beginning of every cycle.

You also have to add one line at the function:

```
void update_access_stats(access_data_t * access_data){

…

access_data->accesses[LSQ_PREG][lsq_preg_access]++;
access_data->accesses[MY_UNIT][my_unit_access]++;
```

This function is called at the end of every cycle, and increments the correspondent position of the vector (previously explained).

You will have to count the accesses to your unit, you will have to know where your unit is used (at file `sim-vatios` or the name of your simulator main file) and every time you use your unit you have increment the correspondent counter:

```
my_unit_access++;
```

5.4. Creating the power models

Then, you have to create a section at the file `calculatePower.h` that calculates at least one power model for the new unit. If you need help adding power models, please read the previous section in this document. Here there is an example of the section for a unit:

```
/******************** MY UNIT ************************/

//Power model selection

switch(energyModels[MY_UNIT]){
      case 0: /*Don't count this unit*/
            power.powerAFIndependent[MY_UNIT] = 0.0;
            break;
      case 1: /*Default power model*/
            power.powerAFIndependent[MY_UNIT] = myUnitPowerModel();
      case 2:
            power.powerAFIndependent[MY_UNIT] =
                  userPowerAFIndependent[MY_UNIT];
            break;
      default:
            fprintf(stderr,"Invalid energyModel for %s\n",
            varUnitNames[MY_UNIT]);
            exit(-1);
}
```

5.5. Adding the call to the function that calculates power/energy

The last step is to add one line in a function at `power.c`

The name of the function is:

```
void calculate(int accesses[][MAX_ACCESS_CYCLE],double
activity[][MAX_ACCESS_CYCLE],int energyModels[NUM_UNITS]){
```

You have to add one line, next to the last call to the function:

```
calculateEnergyAndDisplay(…)
```

And the line you have to add looks like:

```
calculateEnergyAndDisplay(accesses,MY_UNIT,power,my_unit_max_a
ccess,FALSE, NULL,0,totalEnergy);
```

`MY_UNIT` is the identifier you have declared at `access_vatios.h` and

`my_unit_max_access` is the max number of accesses to your unit at a cycle.

6. Adding the Vatios library to an existing SimpleScalar based simulator.

Basically, Vatios is composed of a standard SimpleScalar simulator plus some files and some modifications to `sim-outorder.c`. These modifications are only made to counter use and activity of the units, not to modify the behaviour of the simulator. In this chapter we are going to explain what has to be done to add Vatios to another SimpleScalar based simulator.

First of all you have to add to your project the `cacti4_2` directory, the `access_vatios.c/.h`, `power.c/.h`, `calculatePower.h` and `vatios_strings.h`. Additionally, the `options.c` file is slightly modified, so you will have to replace the original.

Now you will have to modify your simulator to link it with the Vatios library.
At the beginning of your simulator (usually `sim-outorder.c`), you will have to include the `access_vatios` library:

```
/* added for Vatios */
#include "access_vatios.h"
```

Then, as global variables, you will have to declare these variables:

```
/* Vatios*/
extern char * dumpConfigFileName;
access_data_t access_data;
int energyModels[NUM_UNITS];

int va_size = 48;
int  technologyPoint = 350;
double crossover_scaling = 1.2;
double turnoff_factor = 0.1;
int opcode_length = 8;
int inst_length = 32;
extern char * varUnitNames[];
extern char * varAfNames[];
extern char * comentars[];
extern char * afComentars[];
char modelOption[NUM_UNITS][128];
```

In the function that registers simulator –specific options, you will have to include these new options.

The function will look like:

```
/* register simulator-specific options */
Void sim_reg_options(struct opt_odb_t *odb){
     //Vatios
     int j;
     for(j=0;j<NUM_UNITS;j++){
          sprintf(modelOption[j],"%s:model",varUnitNames[j]);
          opt_reg_int(odb,modelOption[j],"Energy Model",
                              &energyModels[j],1,FALSE,NULL);
          //We use 1 as default value,
          //because in Vatios 0 means
          // that this unit consumes 0 energy.
     }

opt_reg_double(odb, "-tech","Tech point.(Double in nm)”
     ,&technologyPoint,350,FALSE, NULL);
```

The next step is to add the code to show the results and to dump all the access/use info to a file. You have to find the `sim_aux_stats` function and modify it so that it begins like:

```
/* dump simulator-specific auxiliary simulator statistics */
void
sim_aux_stats(FILE *stream)             /* output stream */
{

//Vatios
if(dumpConfigFileName != NULL)
               dump_access_vectors(stream,&access_data);
calculate(access_data.accesses,access_data.activity,energyMode
ls);
```

The last step is to add at the main simulator, one call to **`clear_access_stats();`** and one call to **`update_access_stats(&access_data);`** at these points of the main loop:

```
  for (;;)
    {
      /* RUU/LSQ sanity checks */
      if (RUU_num < LSQ_num)
     panic("RUU_num < LSQ_num");
      if (((RUU_head + RUU_num) % RUU_size) != RUU_tail)
     panic("RUU_head/RUU_tail wedged");
      if (((LSQ_head + LSQ_num) % LSQ_size) != LSQ_tail)
     panic("LSQ_head/LSQ_tail wedged");

      /* added for Wattch to clear hardware access counters */
```

```
      clear_access_stats();
```

```
…
      else
     ruu_fetch_issue_delay--;

  /* Added by Wattch to update per-cycle power statistics */
      update_access_stats(&access_data);

      /* update buffer occupancy stats */
      IFQ_count += fetch_num;
…
```

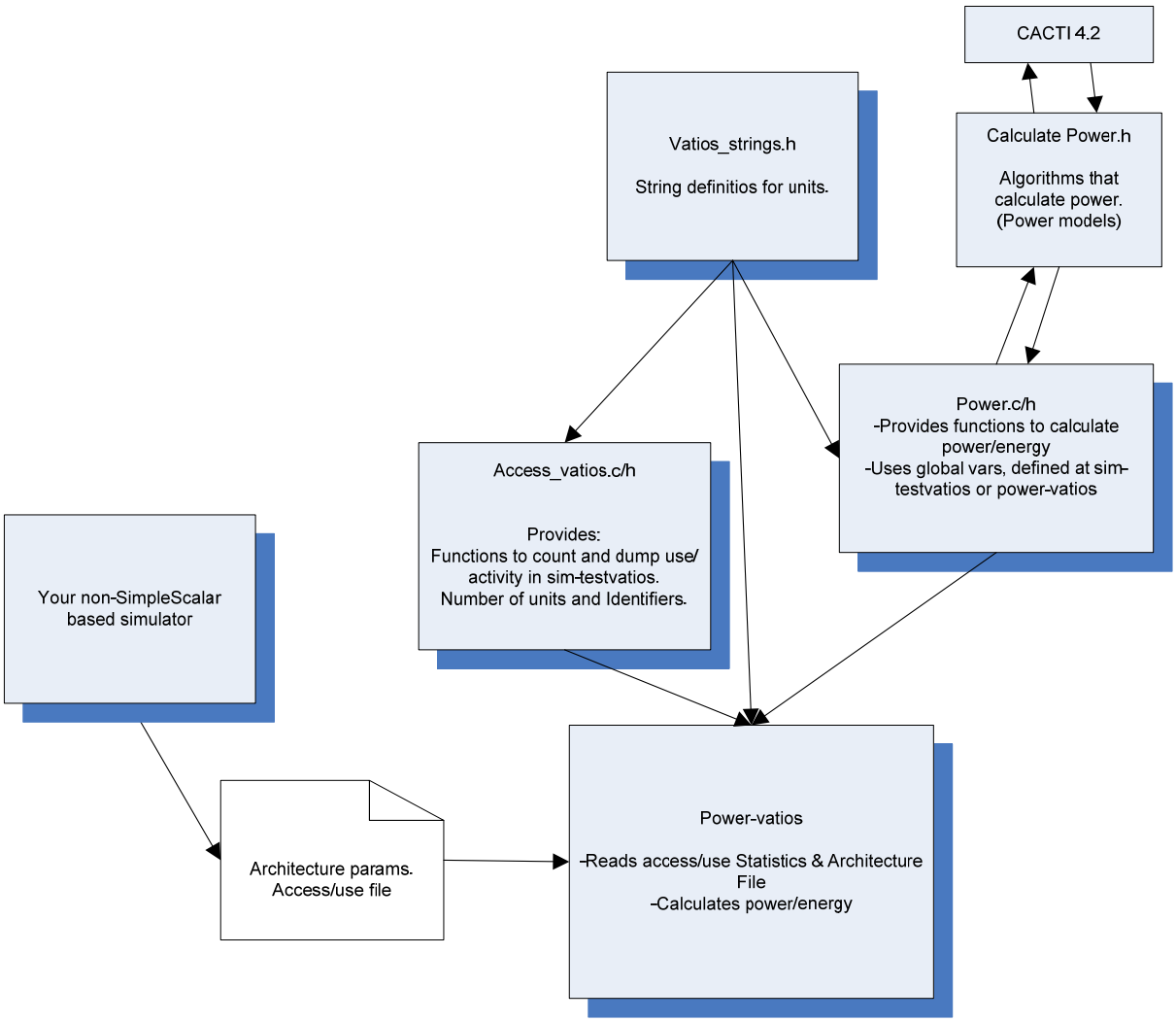
The rest of the work depends on the units in your simulator and your power models. It's recommended to understand the differences between `sim-vatios.c` and the original sim-outorder.c before including Vatios to an existing SimpleScalar based simulator.

To compile the project, you will have to modify the Makefile. You can take the `Makefile` distributed with Vatios as an example.

7. Adding Vatios to a non-SimpleScalar based simulator.

If you are using a simulator that is not based on SimpleScalar, you won't have a sim-outorder file. Despite that, you could use the Vatios library and `power-vatios`. You only have to modify your simulator to dump some stats and architectural parameters to a file with a certain format.

This would be the simulator architecture:



It's highly recommended to be familiar with Vatios and how to add new power models/units. First of all, you should generate a file, readable by the SimpleScalar `options.h` library, with all the information needed by `power-vatios.`

Any option should look like:

```
-optionName value
```

Lines that begin with '#' are considered as comments.

For example:

```
# instruction decode B/W (insts/cycle)
-decode:width                    4
# instruction issue B/W (insts/cycle)
-issue:width                     4
# run pipeline with in-order issue
-issue:inorder                false
# issue instructions down wrong execution paths
-issue:wrongpath               true
# instruction commit B/W (insts/cycle)
-commit:width                    4
# register update unit (RUU) size
-ruu:size                       16
# load/store queue (LSQ) size
-lsq:size                        8
# l1 data cache config, i.e., {<config>|none}
-cache:dl1             dl1:128:32:4:l
```

For every unit you are going to calculate energy predictions, you will have to write access and use information using the function `void dump_access_vectors(FILE * stream, access_data_t * access_data)` included in access_vatios.h, or with this format:

```
-unitName:accesses       [Cycles0,Cycles1,Cycles2,…,CyclesN]
```

Where Cycles0 is the number of cycles when this unit has been accessed 0 times, Cycles1 is the number of cycles when this unit has been accessed 1 time, and so on…

For units that depends on an Activity factor (that can vary between 0 and 1), you will have to write the Activity factor information with this format:

```
-unitName_af         [term0,term1,term2,…,termN]
```

Where termX is the addition of all the activity factors resulting from cycles where the number of accesses to this unit have been X.

The activity factor can be calculated using `pop_count` function. You have predefined functions to do all of this in the `acces_vatios.h` library.

An example of the access and use information is:

```
#=========================================#
#               ACCESS VECTORS            #
#=========================================#

-rename:accesses         [34095,2756,3321,2438,9505]
-bpred:accesses          [45568,5738,780,29]
-icache:accesses         [31714,20401]
-dcache:accesses         [41927,7021,2854,285,28]
-dcache2:accesses        [49057,3032,23,2,1]
…
…
-lsq_af [0.0,2546.171875,494.617188,326.916667,42.101562]
-resultbus_af [0.0,4727.468750,5078.125,5485.276042,3266.125]
```

Of course in the dumped file, you will have to include all the information needed by your power models. `power-vatios` depends on all this libraries, so you will have to include them from the Vatios tar.gz file from the Web. Don't forget to include the .c files as well.

```
access_vatios.h
vatios_strings.h
options.h
power.h
cache.h
misc.h
```

Note: the function `dump_access_vectors(FILE * stream, access_data_t * access_data)` depends on a extern variable declarated as:

```
extern char * dumpConfigFileName;
```

You have to make sure that this variable has the name of the file where all other information has been dumped when you call the function.

Now, you only have to modify the files you have included to map the units in your simulator. The guide to this process can be found at previous sections of this manual.

Note:

There are some units whose power depends on an activity factor. These units should be treated specially, because not only the number of accesses should be count, but also the activity factor of the results.

Regfile
Resultbus
Window
LSQ

If you are going to add a new unit and you want to measure the activity factor, please, take a look at how these units are modelled. The process is similar, but you will have to add extra code in some points.