Universidad de Valladolid

Facultad de Ciencias

Trabajo Fin de Grado

Grado en Estadística

Statistical Model Checking in Approximate Computing Systems

Autor: Sergio Pérez Hernández

Tutor: Luis Ángel García Escudero

Resumen

Este trabajo se centra en la prueba de sistemas computacionales aproximados, comprobando, entre otras cosas, su tasa de acierto y si son útiles. Para estas pruebas se utilizará UPPAAL SMC, una herramienta de comprobación de modelos estadísticos.

Abstract

This thesis is focused in testing approximate computing systems, checking, among other things, its hit rate and if they are useful. For these tests, UPPAAL SMC, a statistical model checking tool, will be used.

Acknowledgements

I would like to thank Mr. Josef Strnadel, teacher of the Information Technology Faculty at Brno University of Technology, for his effort and dedication in helping me throughout the proccess of creating my thesis.

Contents

Chapter 1

Introduction

In the last two or three dacades, information has became the most valuable thing in the world. With it, we can study what has happened, what is happening and what will happen. But have these data is not free. In a lot of cases, it is needed a lot resources to process and store it. This concepts introduce 'Big Data'. By definition, Big Data is a extremely large data sets that may be analysed computationally to reveal patterns, trends, and associations, especially relating to human behaviour and interactions. But sometimes, handle with this huge amount of information is not possible because there isn't any tool available or it is so expensive.
Here is where approximate computing appears. Admitting that some error could occur, we can improve the speed, complexity and size of our problem so we are able to have significative results.

Approximate computing is used in a lot of fields, such as machine learning, scientific computing, etc. To observe the behaviour of some systems that use this technology, Statistical Model Checking method will be used. It does simulations of a system in a stochastic way, that is, non-deterministic, with certain conditions. Then, the results will be studied. Quantitative properties of stochastic systems are usually specified in logic that allow one to compare the measure of executions satisfying certain temporal properties with thresholds. The model checking problem for stochastic systems with respect to such logic is typically solved by a numerical approach that iteratively computes (or approximates) the exact measure of paths satisfying relevant subformulas.([3])

In this bachelor's thesis, I will try to prove that approximate computing systems are useful in some specific fields. To do that, I will check the properties of some approximate computing circuits that I've design in my Information Technology thesis. The tool that will be used for the implementation will be UPPAAL, on version 4.1. It is an integrated tool environment for modelling, validation and verification of real-time systems modelled as networks of timed automata.([6])

Chapter 2

Background: Statistical Model Checking

Model Checking is a recognized approach to guarantee the correctness of a system (4). It is based on algorithms that check whether all executions of a system satisfy some properties stated in specification logic. If this happens, the system is correct. Else, a bug is reported. Model checking can detect all bugs of a system, but it is generally slower.
Classical model checking techniques are Boolean, but this view is now obsolete. This has motivated the development of a series of new techniques, based in probability.

Statistical Model Checking has recently been proposed as an alternative to avoid an exhaustive exploration of the state-space of the model. The main idea is to conduct some simulations of the system, monitor them, and then use results from statistic area to decide if the system satisfies the property or not with some degree of confidence. SMC is a compromise between testing and classical model checking techniques. Furthermore, it is very simple to implement, it doesn't require extra modelling or specification effort and it allows to model check properties that cannot be expressed in classical temporal logics.

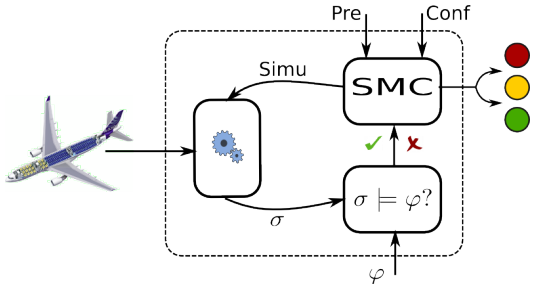


Figure 2.1: Scheme of Statistical Model Checking (4)

Chapter 3

Choice of Implementation Areas and Means

3.1 Implementation Areas

In this thesis, I will evaluate some models that I have designed and developed for my Information Technology thesis (5). It includes two models of an approximate logical multiplier, one based on gate networks and another one based one truth tables. These are the specifications of those models:

3.1.1 Approximate logical multiplier

A logical multiplier is a circuit that, from a series of inputs that represents a binary number, and another that represents another binary number, the multiplication of both numbers in the form of 0 and 1 is obtained (5). The length of the output will be the sum of the length of the two inputs. This, in case that the length of these numbers is very high, can lead to a high cost in the processing of the output. One solution can be the approximate logical multipliers.
The approximate logical multipliers are the same as the exact ones, except that they have fewer outputs. That means that we will have fewer costs but some multiplications won't be correct. This kind of multipliers are useful in systems that don't require a perfect computation, that is, they are tolerant of errors. Because of that, a certain amount of accuracy is sacrificed to reduce the area of the circuit and power consumption and increase the performance.(2)
To do the tests, an accurate multiplier and an approximate multiplier will be implemented. Both will have 4 inputs, where we want to multiply (B1, B0) with (A1, A0), obtaining 4 outputs: (Out3, Out2, Out1, Out0). This is the truth table for the exact logical multiplier:

B1	B0	A1	A0	Out3	Out2	Out1	Out0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

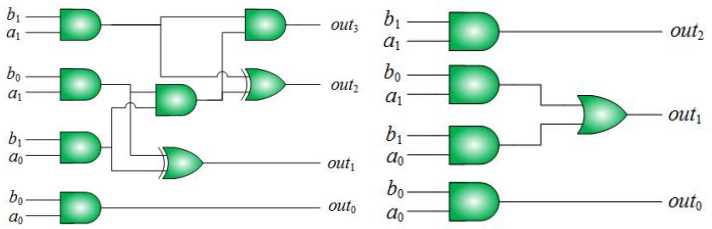
Table 3.1: Truth table of the accurate logical multiplier

We can see that, for Out3, all possible outputs are 0 except for the last one. What is done with the approximate multiplier is to delete this output. The last one will be transformed into (1,1,1), so, for these 16 possible outputs, only 1 will be incorrect:

B1	B0	A1	A0	Out2	Out1	Out0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	0	1	0	0	1
0	1	1	0	0	1	0
0	1	1	1	0	1	1
1	0	0	0	0	0	0
1	0	0	1	0	1	0
1	0	1	0	1	0	0
1	0	1	1	1	1	0
1	1	0	0	0	0	0
1	1	0	1	0	1	1
1	1	1	0	1	1	0
1	1	1	1	1	1	1

Table 3.2: Truth table of the approximate logical multiplier

Then, from these tables, the logic circuits can be created:



(a) Accurate multiplier circuit (2) (b) Approximate multiplier circuit (2)

As can be seen in the pictures above, the accurate multiplier has 8 logic gates (6 AND gates and 2 XOR gates), while the approximate one only has 5 (4 AND and 1 OR), being this last one the simplest.

3.1.2 DRAM memory

A DRAM (*Dynamic Random Access Memory*) memory is a kind of RAM memory based on capacitors, which lose their charge progressively, so they need a refresh dynamic circuit that, every certain period, check this charge and replenish it. Its principal advantage is the possibility of build memories of a high density of positions and that work at a high speed. Like the rest of the types of RAM memories, it is volatile. It means that if the electrical power is interrupted, the stored information will be lost. The DRAM is widely used in digital electronics where low-cost and high-capacity memory is required. At the moment, it is one of the most used memories.(9; 10)
Each memory cell is the basic unit of each memory, able to store a bit in its logic circuits. Each cell has a transistor and a capacitor. Cells are organized in two-dimensional matrices, which are accessed through rows and columns.

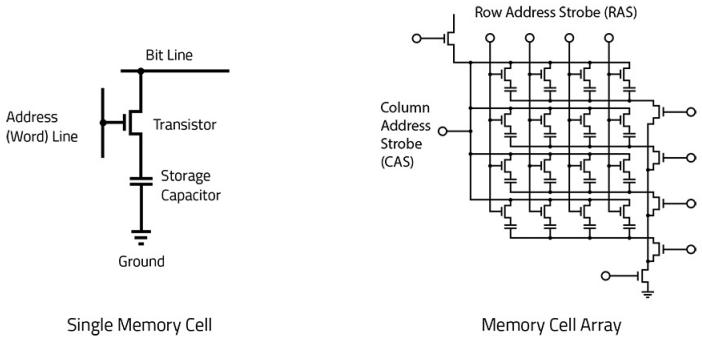


Figure 3.1: DRAM scheme (11)

If there is a charge inside the capacitor, it means that the logic value of the cell is 1. Otherwise, it is 0. The transistor connects or disconnects the capacitor. Over time, the capacitor will be discharging progressively. In the case that the voltage is below a threshold value, it will be assumed that the logic value of the cell is 0. That's why every so often it is necessary to recharge the capacitor. Therefore, the error rate of a DRAM memory will be checked depending on the time between the refresh periods.

3.2 Implementation Means

The tool that I have used to build the models and I will use to evaluate them is UPPAAL. It is a toolbox for validation and verification (via automatic model-checking) of real-time system (8).
The objective of this tool is to model a system using timed automata, simulate it and then verify properties on it. Timed automata are finite state machines with time (clocks). A system consists of a network of processes that are composed of locations. Transitions between these locations define how the system behaves. The simulation step consists of running the system interactively to check that it works as intended. Then, we can ask the verifier to check reachability properties.

This tool, at this moment, is in version 4.0, but there is also a version 4.1 that is in development, which includes an SMC extension. This last version is what will be used.

3.2.1 UPPAAL 4.0

UPPAAL is based on timed automata, that is a finite state machine with clocks. The clocks are the way to handle time. It is continuous and the clocks measure time progress. It is allowed to test the value of a clock or to reset it. Time will progress globally at the same pace for the whole system.

A system in UPPAAL is composed of concurrent processes, which are modelled as an automaton. This automaton has a set of locations. Transitions are used to change location. To control when to take a transition, it is possible to have a guard and a synchronization. A guard is a condition on the variables and the clocks saying when the transition is enabled. When a transition is taken, two actions are possible: assignment of variables or reset the clocks.

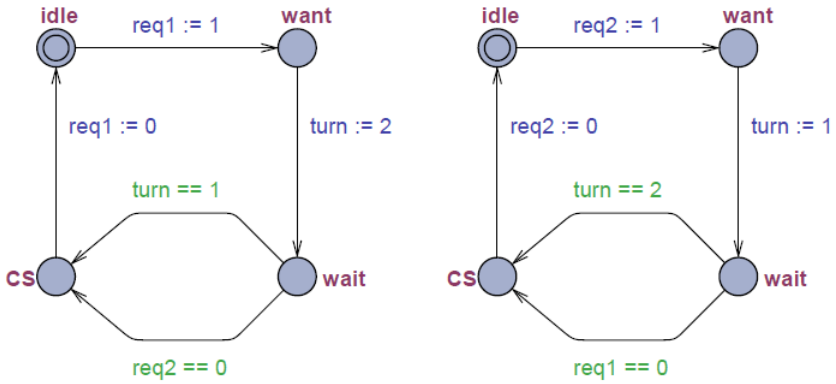


Figure 3.2: Example of a UPPAAL model ([8])

Locations

There are different kinds of locations of UPPAAL ([7]):

- **Normal locations** (with or without invariants).
- **Urgent locations:** This kind of locations freeze time. It means that time is not allowed to pass. They are marked by a U inside the circle.
- **Committed locations:** These locations also freeze time, but also, the next transition must involve an edge from one of the committed locations. They are useful for creating atomic sequences and for encoding synchronization between more than two components. They are marked by a C inside the circle.

There is also one (and only one) initial state. It is marked by a double circle.

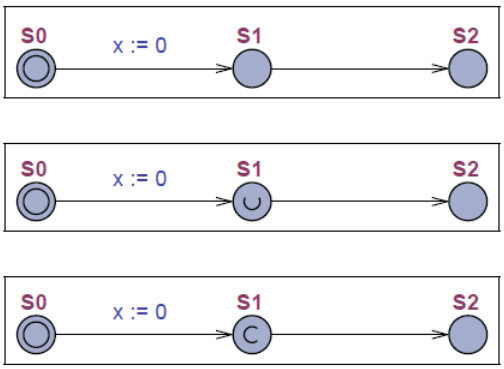


Figure 3.3: Example of a normal, urgent and committed states

Verifying properties

To check if a property of our model is correct, UPPAAL has a verifier tool. The queries available in the verifier are ([8]):

- **E<>** ***p***: There exists a path where p eventually holds.
- **A[ ]** ***p***: For all path p always holds.
- **E[ ]** ***p***: There exists a path where p always holds.
- **A<>** ***p***: For all paths p will eventually holds.
- ***p*** $--> $ ***q***: Whenever p holds, q will eventually hold.

p and q are state formulas.
There is also a special query: **A[ ] not deadlock**, that checks for deadlocks (more transitions are not possible).

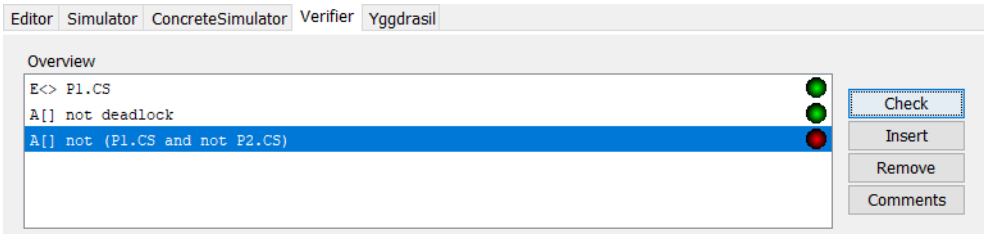


Figure 3.4: Verifier tool in UPPAAL

In this example we can see that the first and the second properties that we want to check are validated, but not the last one.

3.2.2 UPPAAL 4.1 (SMC)

The modelling formalism of UPPAAL SMC is based on a stochastic interpretation and extension of the timed automata formalism used in the classical model checking version of UPPAAL.([1]) For individual timed automata components, the stochastic interpretation replaces the non-deterministic choices between multiple enabled transitions by probabilistic choices. Similarly, the non-deterministic choices of time delays are defined by probability distributions, which at the component level are given either uniform distributions in cases with time-bounded delays or exponential distributions in cases of unbounded delays.

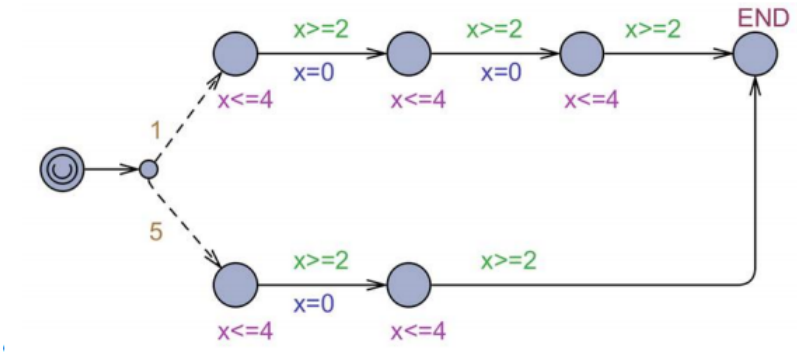


Figure 3.5: Example of a stochastic timed automata ([11])

A model in UPPAAL SMC consists of a network of interacting stochastic timed automata. It is assumed that these components are input-enabled, deterministic and non-zero. These components communicate via broadcast channels and shared variables to generate networks of stochastic timed automata. The communication is restricted to broadcast synchronizations to keep a clean semantics of only non-blocked components which are racing against each other with their corresponding local distribution.

Additional verifying properties

Simulation In addition to the standard model checking queries, UPPAL SMC provides a number of new queries related to the stochastic interpretation of timed automata. In particular, it allows the user to visualize the values of expressions along simulated runs (evaluating to integers or clocks), providing insight to the user on the behaviour of the system so more interesting properties can be asked to the model-checker.
The concrete syntax applied in UPPAAL SMC is as follows:

simulate $[<= \mathit{bound}]\{E_1,...,E_k\}$

where N is a natural number indicating the number of simulations to be performed, *bound* is the time bound on the simulations, and $E_1,...,E_k$ are the k expressions that are to be monitorized and visualized.

Probability estimation The probability estimation algorithm of UPPAAL SMC computes the number of runs needed to produces an aproximation interval $[p-\varepsilon, p+\varepsilon]$ for $p = Pr(\Psi)$ with a confidence $1-\alpha$. A frequentist interpretation of this result tells us that if we repeat the interval estimation N times, then the estimated confidence interval $(p \pm \varepsilon)$ contains the true probability at least $(1-\alpha)$ N in the long run $(N \to \infty)$

The syntax applied to check probability estimations is:

Pr $[bound](Expression)$

Also, UPPAAL gives you some plots, like the probability density distribution plot or the cumulative probability distribution plot.

Figure 3.6: Queries about probability estimation and simulation

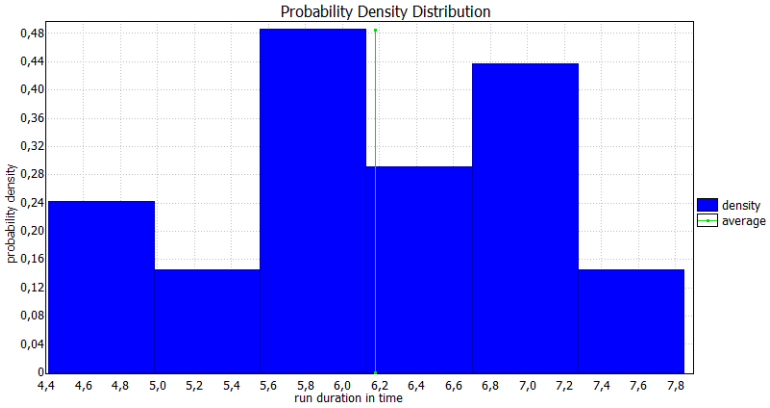


Figure 3.7: Probability density distribution plot

Hypothesis testing This approach reduces the qualitative question to test the null-hypothesis $H_0 : p \geq p_0$ against the alternative hypothesis $H_1 : p < p_0$. In UPPAAL SMC, we use the following query:

Pr $[bound](Expression) >= p_0$

Probability comparison The algorithm use the Wald test to compare probabilities, with the following query:

$\mathbf{Pr}[bound_1](Expression_1) = \mathbf{Pr}[bound_2](Expression_2)$

Expected values UPPAAL SMC also supports the evaluation of expected values of min or max of an expression that evaluates a clock or an integer value. The syntax is:

$\mathbf{E}[bound;N](\mathbf{min}{:}Expression)$ or $\mathbf{E}[bound;N](\mathbf{max}{:}Expression)$

Chapter 4

Evaluation

For the different tests that will be done on the models described before, the verifier tool of UPPAAL SMC will be used. With it, we can obtain various plots, confidence intervals, results of simulations using diagrams, etc.

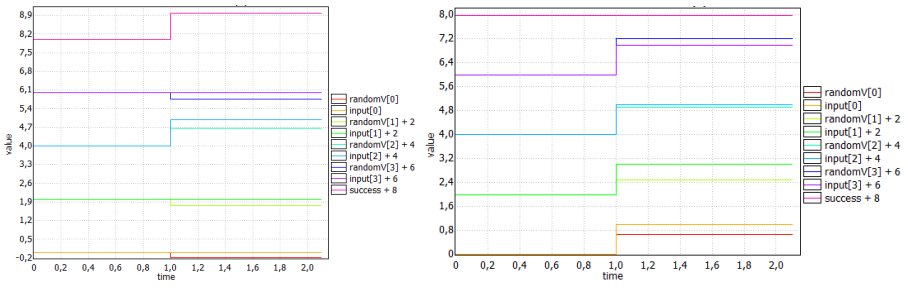
4.1 Tests on multiplier based on gate networks

Verification of the model

First of all, it will be checked that the system behaves correctly. To do that, the parameter of the template `Main` will be set on 1, so only one iteration will be done. The following plot has been obtained with the command:

```
simulate[<=2] {randomV[0], input[0],
randomV[1]+2,input[1]+2,randomV[2]+4, input[2]+4, randomV[3]+6,
input[3]+6, success+8}.
```

On it, there will be represented, starting from below, the four random values obtained using a normal distribution. With them, if the value is positive, the input will be 1, and if it is negative, the input will stay in 0. At the top, the success variable is shown. These are two possible results:



(a) (b)

In figure 4.1a, we can see that, for the inputs 0, 1 and 3, the random values obtained are negative, and for the input 2 it has been positive, so the corresponding input is `0010`. With this combination of inputs, the approximate and the accurate multipliers have the same outputs, so the variable `success` has increased one unit.
In the simulation that is represented in the second plot 4.1b, all random values have been positive, so the inputs of the system were `1111`. That is the only possible input which the outputs of the multipliers are different. That means that there has been a failure, so the variable `success` doesn't increase.

Number of successes

In this test, I will see how many successes are obtained depending on the number of iterations that the system does. It is known that, theoretically, there is only 1 chance of 16 that the system fails (0.0625%). For the simulations, the next command has been used:

`simulate[<=`N`] {success}`

where N represents the number of iterations done. These are the results:

N	Successes	Fails
10	10	0
50	46	4
100	94	6
200	183	17
300	278	22
400	376	24
500	471	29
750	710	40
1000	922	88
2000	1871	129

These results seem similar to the theoretical value, but it is needed to check it. To do it, for each number of iterations, I will do a hypothesis test for proportions , testing $H_0 : p = 0.0625$. I use R to do it, obtaining a p-value with the function:

```
prop.test(Fails,N,p=0.0625,alternative="two.sided")
```

The results are:

N	p-value
50	0.8266
100	1
200	0.2426
300	0.5118
400	0.9177
500	0.7465
750	0.3362
1000	0.00109
2000	0.7465

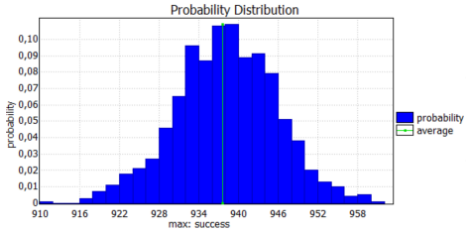
With these results it can be said that it is not possible to reject H_0 for all N tested except for N=1000.

Behaviour of the number of successes

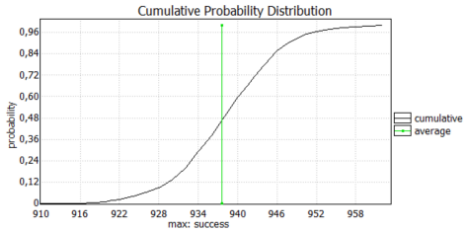
With this test, I will check what is the statistical behaviour of the number of successes. To do that, I calculate the expected values of the maximum of the variable `success` and then check the plots. I've used the next command in the verifier tool of UPPAAL:

`E[<=`*bound*`;`*N*`](max:success)`

where *bound* and N are numbers big enough to have the significant results, and *bound* need to be at least as big as the parameter `loops` of the template `Main`. For the experiment, I've chosen 1000 for both numbers. These are some results:



(c) Probability Distribution Plot



(d) Cumulative Probability Distribution Plot

As numerical results, this simulation has been obtained as mean $\mu = 937.6$ and a confidence interval of 95% $[937.129, 938.071]$.

Observing the plots 4.1c and 4.1d, we can see clearly that the variable `success` follows a normal distribution.
Transforming now these values to error rate (doing the operation $1 - (\frac{x}{1000})$), we obtain that the average error rate is 0.0624, and the confidence interval in 95% is $[0.061929, 0.062871]$. With this interval, we can see that the theoretical value is in it, and the mean is also really proximate to that number.

Comparison of the resources used by both multipliers

In this test, I will compare the time needed by the system to calculate N outputs of the accurate and the approximate multiplier separately. This time is given by UPPAAL at the end of the calculation. I've used the same command of the first test but varying the *bound*. These are the results:

N	Accurate multiplier	Approximate multiplier
100	0.032	0.016
1000	0.235	0.125
2000	0.36	0.281
3000	0.594	0.359
4000	0.75	0.563
5000	0.969	0.64
6000	1.093	0.75
7000	1.328	0.86
8000	1.485	1.015
9000	1.703	1.125
10000	1.969	1.344
11000	2.047	1.375
12000	2.266	1.5
15000	2.891	1.828

Table 4.1: Time used by the multipliers in seconds depending on the number of iterations

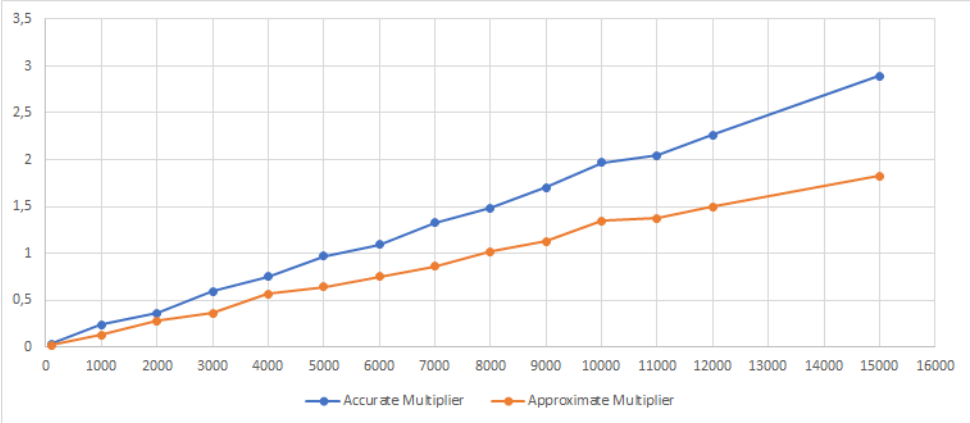


Figure 4.1: Plot of times of both multipliers

With this table and this plot, we can see that the time used by the accurate multiplier is higher than the used by the approximate multiplier in every point. But first, I will check the relation between both lines, checking if they are related.

For that I will do a paired t-test with $H_0 : \mu_D < 0$ and $H_1 : \mu_D > 0$. I've used the function:

`t.test(`*acc_mult*,*appr_mult*,`alternative="greater",paired=TRUE)`

obtaining the following results:

```
data:  acc_mult and appr_mult
t = 4.0768, df = 13, p-value = 0.0006545
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.2183836       Inf
sample estimates:
mean of the differences
              0.3861071
```

Figure 4.2: Paired t-test for times in gate network approach

The p-value obtained is 0.0006545, so H_0 can be rejected in favor of H_1, showing differences between both models.
Now I will do an coincidence test to check if there exists any differences between the two regression lines. I unify all the times in a unique column and I create two new columns: Z, which is 0 in case the observation is from the accurate multiplier and 1 if it is from the approximate one; and W=Z*N to deal with different slopes for the regression lines:

N	Time	Z	W
100	0.032	0	0
1000	0.235	0	0
2000	0.36	0	0
...	...	...	...
100	0.016	1	100
1000	0.125	1	1000
2000	0.281	1	2000
...	...	...	...

I consider two different models:

- Model 1: $Time = \beta_0 + \beta_1 N$
- Model 2: $Time = \beta_0 + \beta_1 N + \beta_2 Z + \beta_3 W$.

First, I see if there is any gain adding the information of W and Y codifying the categorical one. I use the function `anova(Model_1,Model_2)`:

```
  Res.Df     RSS Df Sum of Sq      F    Pr(>F)
1     26 1.85137
2     24 0.03045  2    1.8209 717.72 < 2.2e-16 ***
```

Figure 4.3: Anova test for Model 1 and Model 2

The p-value obtained is practically 0, so it can be said that the second model gives more information than he first one. Using `summary(`*Model_2*`)` we can see more information:

```
Call:
lm(formula = Tiempo ~ N + Z + W, data = X)

Residuals:
      Min        1Q    Median        3Q       Max
-0.049512 -0.019564 -0.005595  0.014678  0.088501

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.946e-03  1.760e-02   0.224    0.825
N            1.898e-04  2.227e-06  85.228   <2e-16 ***
Z            1.574e-02  2.489e-02   0.632    0.533
W           -6.618e-05  3.149e-06 -21.018   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.03562 on 24 degrees of freedom
Multiple R-squared:  0.9979,    Adjusted R-squared:  0.9976
F-statistic:  3779 on 3 and 24 DF,  p-value: < 2.2e-16
```

Figure 4.4: Summary of *Model_2*

Observing the p-values of the coefficients, it makes sense that the intercept and the Z can be set on 0, so the can be deleted from the model. The final model is:

$$Time = \beta_1 N + \beta_3 Z \cdot N$$

So the levels of the Z variables changes the slope of the fitted regression lines. Since W has not been rejected, we can say that there are coincidences, so there is not any parallelism between the accurate model and the approximate one.

Then, observing that the model of the approximate multiplier has less slope than the accurate one, it has been demonstrated that **the approximate multiplier is more efficient than the accurate multiplier in terms of time.**

4.2 Tests on multiplier based on truth tables

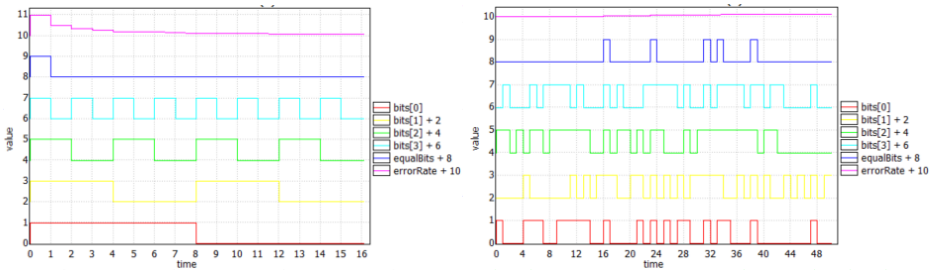
Tests on this model will be similar to the ones done in the previous model, so we can check that the results are correct.

Verification of the model

The first test will check if the system works correctly. For that, I will use both templates that generate inputs. The two next plots are generated in the verifier tool on UPPAAL with the next command:

`simulate[<=`*bound*`]{bits[0],bits[1]+2,bits[2]+4,bits[3]+6,equalBits+8, errorRate+10}`

bound is set in 16 for the first simulation and in 50 for the second.



(a) Simulation using `set_inputs`

(b) Simulation using `set_inputs_random`

Starting from below, the variables shown are the 4 inputs, `equalBits`, that says if there has been a failure in the iteration, and the error rate.
In figure 4.5a we can see that all possible inputs have been generated. The first one is the only one that fails, so the error rate is 1 at that moment. Then, it decreases in every iteration, reaching 0.0625 at the end of the simulation, the theoretical error rate value.
In figure 4.5b the inputs have a random distribution, so the error rate change depending on them.
With this information, it can be concluded that **the system is correct**.

Number of successes

The template that generates random inputs will be used. With it, I will generate N random inputs, and count the number of successes. Finally I will do, with the values obtained, the same hypothesis test that I did for the gate network model: $H_0 : p = 0.0625$ It will be used the same command of the first test, changing the *bound* depending on N. These are values obtained:

N	Successes	Fails
50	49	1
100	94	6
200	184	16
300	278	22
400	375	25
500	462	38
750	710	40
1000	933	77
2000	1862	138

Using the same R command as in the first model, The p-values obtained for each N are:

N	p-value
50	0.3424
100	1
200	0.3808
300	0.5119
400	1
500	0.2482
750	0.3362
1000	0.06741
2000	0.2482

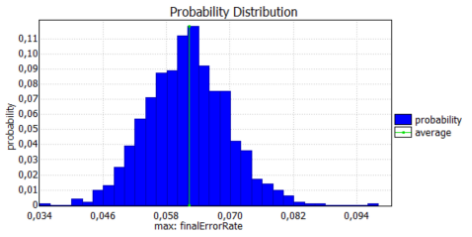
In this case, we can see, considering $\alpha = 0.05$, that all p-values are higher enough to accept H_0

Behaviour of the error rate

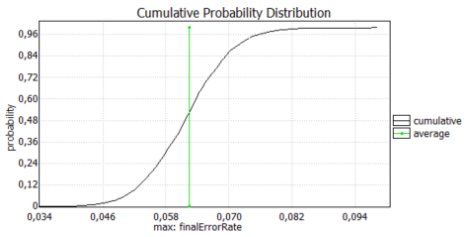
In this test I will obtain the statistical behaviour of the error rate, calculating the expected value of the maximum of the variable `finalErrorRate`. It will be done with the next command:

```
E[<=1000;1000](max:finalErrorRate)
```

Some of the plots obtained are:



(c) Probability Distribution Plot



(d) Cumulative Probability Distribution Plot

As numerical values, the mean of the 1000 simulations is a error rate of 0.0623, with a confidence interval ($\alpha = 0.05$) of $[0.061827, 0.062773]$, which include the theoretical error rate.
Observing the plots, we can conclude that the error rate follows a normal distribution with mean 0.0623.

Comparison of the resources used by both multipliers

In this last test, I will check the time that the verifier tool last to complete N iterations of each multiplier separately. I will disable `outputs_acc` or `outputs_aprox`,

depending on the multiplier which will be tested. The command used on the verification of the model will be used, varying the *bound*. These are the times obtained, in seconds:

N	Accurate multiplier	Approximate multiplier
100	0.016	0.015
1000	0.046	0.031
2000	0.094	0.078
3000	0.125	0.093
4000	0.156	0.125
5000	0.172	0.156
6000	0.188	0.171
7000	0.219	0.203
8000	0.266	0.219
9000	0.297	0.266
10000	0.359	0.281
11000	0.391	0.296
12000	0.422	0.313
15000	0.453	0.375

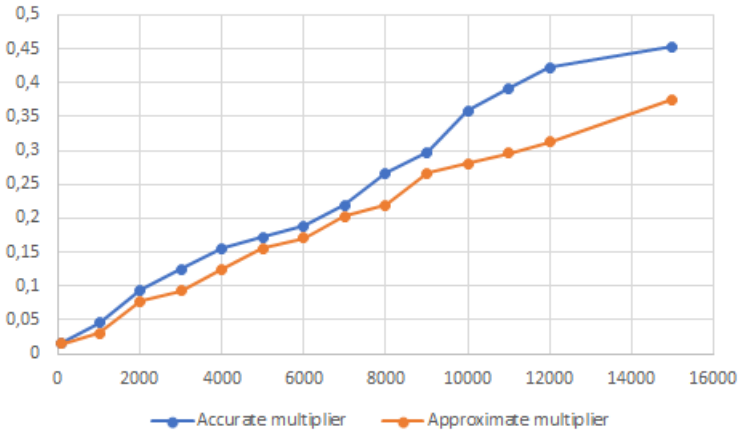


Figure 4.5: Plot of times of both multipliers

It seems that there are no such big differences between both times. First, I do a paired t-test checking $H_0 : \mu_D < 0$ an $H_1 : \mu_D > 0$ to see the relation between the lines as I did with the first model, using the same function. The results are:

```
data:  acc_mult and appr_mult
t = 4.434, df = 13, p-value = 0.000337
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.02471061        Inf
sample estimates:
mean of the differences
             0.04114286
```

Figure 4.6: Paired t-test for times in truth table approach

A low p-value has been obtained, so differences can be considered between both lines.
Then, I will check the possible coincidences that can exist. I've built the same table as I did in with the gate approach but with the corresponding values:

N	Time	Z	W
100	0.016	0	0
1000	0.046	0	0
2000	0.094	0	0
...	...	...	...
100	0.015	1	100
1000	0.031	1	1000
2000	0.078	1	2000
...	...	...	...

I use the same models that I used in the other test: I consider two different models:

- Model 1: $Time = \beta_0 + \beta_1 N$
- Model 2: $Time = \beta_0 + \beta_1 N + \beta_2 Z + \beta_3 W$.

With the R function `anova(Model_1,Model_2)`, I'll check if W and Y give any information:

```
  Res.Df       RSS Df Sum of Sq      F    Pr(>F)
1     26 0.0237233
2     24 0.0063416  2  0.017382 32.891 1.331e-07 ***
```

Figure 4.7: Variance analysis to Model 1 and Model 2

The p-value obtained is very low, so the second model gives more information than the first. Now, with `summary(`*Model_2*`)` I'll check the significance of the variables:

```
Call:
lm(formula = Tiempo ~ N + Z + W, data = X)

Residuals:
      Min        1Q    Median        3Q       Max
-0.037579 -0.007732 -0.000973  0.010619  0.026002

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.965e-02  8.034e-03   2.446 0.022154 *
N            3.140e-05  1.016e-06  30.896  < 2e-16 ***
Z            2.586e-03  1.136e-02   0.228 0.821882
W           -6.576e-06  1.437e-06  -4.576 0.000122 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.01626 on 24 degrees of freedom
Multiple R-squared:  0.9852,    Adjusted R-squared:  0.9833
F-statistic:   532 on 3 and 24 DF,  p-value: < 2.2e-16
```

Figure 4.8: Summary of *Model_2*

In this case, the only variable that can be deleted from the model with $\alpha = 0.05$ is Z. This results are different from the test done for the gate approach, where the intercept was also deleted. The final model is:

$$Time = \beta_0 + \beta_1 N + \beta_3 Z \cdot N$$

In this case, W gives information to the model, which means that the slope of the fitted lines are significatively different depending on whether Z=0 or Z=1. Finally, observing that the approximate multiplier has less slope than the accurate one, it can be concluded that **the approximate multiplier is more efficient than the accurate multiplier in terms of time**.

4.3 Comparison of the two implementations

In both implementation have been tested similar characteristics, obtaining similar results in all of them. In the test done on the gate network model, there was a notable difference between both multipliers, but that behaviour doesn't appear in the truth table model. In my opinion, this difference is caused because of the number of states that are in the first model. The accurate multiplier has more gates than the approximate one, so more transition needs to be done and more time is used. The second model has the same number of iterations in both multipliers, but there is 1 calculation less on the approximate one because it just needs to find 3 outputs and not 4 as the accurate multiplier does. This could cause the insignificant difference between them.

In conclusion, observing all the results, it has been proved that **the truth table implementation is more efficient than the gate network one**.

4.4 Tests on DRAM model

For all the tests, the dimension of the DRAM will be 1 row and 8 columns, but it can be increased as much as the user wants.

Verification of the system

As it has been done before, I will check that the model works correctly. I will write in some registers with different times of refresh and see what are the output. These are two plots with different time of refresh:

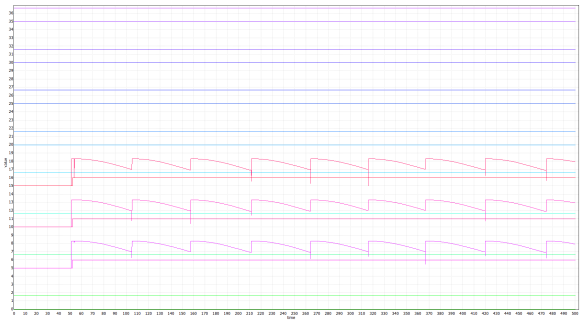


Figure 4.9: Simulation with `tRefresh` = 50

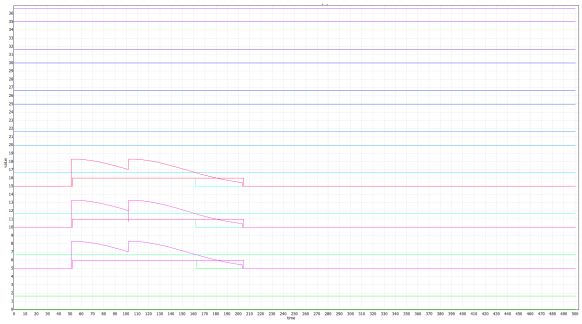


Figure 4.10: Simulation with `tRefresh` = 100

Each plot is divided into 8 parts as if they were the 8 columns of the DRAM. Each part has 4 variables on it: `cVolt` representing the voltage, `V_TRESH` representing the threshold, `eBit` representing the expected logical value and `cBit` representing the real logical value.
In both figures, there is a writing on the cells 1,2 and 3 because the data introduced was 2,4 and 8. The rest of the cells stays doing nothing.
In figure 4.9, the time of refresh is enough so the voltage never gets the threshold, so the real value is equal to the expected value all the time.

In figure 4.10, the voltage exceeds the threshold before the second refresh, so the real value is not equal to the expected value anymore. When a refresh is done, the real value is 0, so it doesn't recharge again.
Therefore, we can conclude that **the model works correctly**.

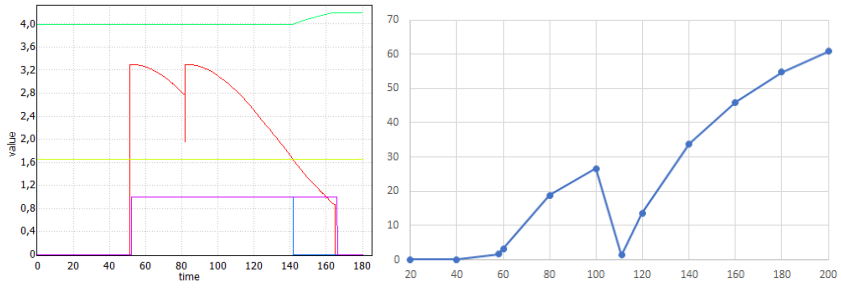
Error rate depending on the time of refresh

With this test, it is wanted to be known what is the error rate of the system varying the time of refresh of the system. Only 1 cell will be used because a writing in several cells at the same time returns the same error rate on all of them. This template simulates a writing on cell 0 on the time unit 50. The command used in UPPAAL verifier is:

```
simulate[<=bound]{cVolt[0][0],eBit[0][0],
  cBit[0][0],V_TRESH,errorRate[0][0]+4}
```

bound should be as big as necessary to see the correct error rate.
These are the results:

Time of refresh	Number of refreshes before a failure	Error rate
20	-	0%
40	-	0%
58	1	1.59%
60	1	3.2%
80	1	18.8%
100	1	26.57%
111	0	1.41%
120	0	13.63%
140	0	33.82%
160	0	46.05%
180	0	54.72%
200	0	60.83%



(a) Simulation with `tRefresh` = 80 (b) Error rate depending on the time of refresh

In plot 4.11a the writing is correct till the time 140 approximately, where the voltage starts to be lower than the threshold and the expected value is set on 0. There is a refresh on time 80, that can be seen because the voltage value is again the maximum possible.
In plot 4.11b we can see that the error rate always increases with the time of refresh, except on one point, on 111. This is because on that point there is a refresh just after the threshold has been reached, so the system updates its real value when very little time has passed since the expected value was updated.
In this case, if the system allows committing some failures, then **the time of refresh that should be selected is 111** because it is the time which gets a lower error rate and without doing any refreshes. If there would be an error rate even lower but doing one refresh , then the system manager must decide what is the best for him, get less error rate but doing a refresh, which has a cost, or gets a little more error rate and doesn't do any refreshes.
This results will change depending on the experiment done. This is given because the writing has been done on the time unit 50, but if this operation is done at another moment, the optimal time of refresh will change. This is caused because the time of refresh is done all times at the same moment, doesn't matter when an operation happens.

Chapter 5

Conclusions

The aim of this thesis was to prove that approximate computing systems are useful if some errors can be accepted in the results obtained. For that, two types of models have been tested: Two models of accurate and approximate logical multiplier and a model of a dynamic random access memory (DRAM). These tests have been done using UPPAAL SMC.

In my opinion, the final results obtained on the different tests performed confirm that the approximate computing systems have a lot of utilities in different fields, such as in image and sound processing, machine learning or artificial intelligence, where some bits can be lost without having serious problems.

Some other tests could be done on these systems but are not implemented in this thesis could be amplify the number of inputs and outputs of the multipliers and see if the error rate is still as low as it is with 4 inputs, or check the behaviour of the error rates with more writings on different moments on the DRAM model.

Bibliography

[1] David, A.; Larsen, K. G.; Legay, A.; et al.: *UPPAAL SMC tutorial*. 2015. [Online; visited 09.05.2019].
Retrieved from: https://doi.org/10.1007/s10009-014-0361-y

[2] Emerging Computing Technology Laboratory at SJTU: *Approximate Computing*. [Online; visited 09.05.2019].
Retrieved from: http://umji.sjtu.edu.cn/~wkqian/research.html

[3] Legay, A.; Delahaye, B.; Bensalem, S.: *Statistical Model Checking: An Overview*. 2010. [Online; visited 09.05.2019].
Retrieved from: https://arxiv.org/pdf/1005.1327.pdf

[4] Plasma Lab: *Statistical Model Checking*. [Online; visited 09.05.2019].
Retrieved from: https://project.inria.fr/plasma-lab/statistical-model-checking

[5] Pérez, S.: *Design and Implementation of Approximate Computing Systems*. 2020.

[6] UPPAAL: [Online; visited 09.05.2019].
Retrieved from: www.uppaal.org

[7] UPPAAL: *Locations*. [Online; visited 09.05.2019].
Retrieved from: http://www.it.uu.se/research/group/darts/uppaal/help.php?file=System_Descriptions/Locations.shtml

[8] UPPAAL: *UPPAAL 4.0: Small Tutorial*. 2009. [Online; visited 09.05.2019].
Retrieved from: http://www.it.uu.se/research/group/darts/uppaal/small_tutorial.pdf

[9] Wikipedia: *DRAM*. [Online; visited 09.05.2019].
Retrieved from: https://es.wikipedia.org/wiki/DRAM

[10] Wikipedia: *Dynamic Random Access Memory*. [Online; visited 09.05.2019].
Retrieved from: https://en.wikipedia.org/wiki/Dynamic_random-access_memory

[11] Yoon, A.: *Understanding Memory*. 2018. [Online; visited 09.05.2019]. Retrieved from: https://semiengineering.com/whats-really-happening-inside-memory/