Ines Slimi

Nanotechnologies for ICTs
2019/2020

STMICROELECTRONICS
12 RUE JULES HOROWITZ, 38019 GRENOBLE, FRANCE

Verification of a non-volatile memory block (flash) using mixed simulation tools (analog + digital).

From 16/02/2020 to 31/06/2020

Confidentiality: YES

Under the supervision of
- Company supervisor: Thomas Jouanneau, thomas.jouanneau@st.com
- Phelma tutor : Lorena Anghel, lorena.anghel@grenoble-inp.fr

Present at the defense: YES

Ecole nationale
supérieure de physique,
électronique, matériaux
Phelma
Bât. Grenoble INP - Minatec
3 Parvis Louis Néel - CS 50257
F-38016 Grenoble Cedex 01
Tél +33 (0)4 56 52 91 00
Fax +33 (0)4 56 52 91 03
http://phelma.grenoble-inp.fr

”all models are wrong but some are useful”

Georges box

Contents

Glossary

eSTM : embedded Select in Trench Memory.
VXR2 : An output pin of the VXR charge pump.
VDD, VDDPUMP : The input pins of the VXR charge pump.

List of Figures

List of Tables

Abstract

The eSTM memory block is made of digital and analog sub blocks. Analog sub blocks may be described through models to speed up the simulation while keeping reasonable accuracy. This work aims at presenting the improvement done on existing charge pump models by adding information about their supply currents consumption in order to optimize the trade off between time and accuracy.
key words: charge pump - simulation.

Le bloc de mémoire eSTM est composé de sous-blocs numériques et analogiques. Des sous-blocs analogiques peuvent être décrits à travers des modèles pour accélérer la simulation tout en conservant une précision raisonnable. Ce travail vise à présenter les améliorations apportées aux modèles de pompe de charge existants en ajoutant des informations sur leur consommation de courants d'alimentation afin d'optimiser le compromis entre le temps et la précision.
mots clés: pompe de charge - simulation.

Il blocco di memoria eSTM è costituito da blocchi secondari digitali e analogici. I blocchi secondari analogici possono essere descritti attraverso modelli per accelerare la simulazione mantenendo una precisione ragionevole. Questo lavoro intende presentare i miglioramenti apportati ai modelli esistenti di pompe di carica aggiungendo informazioni sul loro consumo di correnti di alimentazione al fine di ottimizzare il compromesso tra tempo e precisione.
parole chiave: pompa di carica - simulazione

1 Introduction

1.1 Introduction

My internship was carried out within the MDG Division (Microcontrollers Digital IC Group) of STMicroelectronics which is an organisation mainly deployed in France and Italy. It mainly designs microcontrollers for various applications such as consumer electronics (STM32) and secure products (NFC, banking). I've joined the Central RD team eNVM (Embedded Non-Volatile Memory) within this MDG division. This team is in charge of producing non-volatile memory blocks (Flash, EEPROM) intended to be integrated into various microcontrollers.

A flash memory block is a complex system with various analog and digital components. Before going into production, this system is verified by mixed simulations. This allows to verify the interactions between the different sub-blocks and the proper functioning of the whole system. The effectiveness of these mixed simulations is measured on two criteria : accuracy of the simulation and speed of execution.

However, for certain simulations which target functionality rather than performance, a compromise could be found by intelligent modeling of different analog sub-blocks. This allows to greatly improve the speed of simulation while limiting the loss of precision.

This is the case of charge pumps which are analog sub-blocks of the flash memory block that are very heavy to simulate on full transistor level.

For this reason, this internship aims at participating in the improvement of existing models of charge pumps in order to increase their accuracy while minimizing the impact on the speed of the simulation. The main purpose of this thesis is to first develop models in hardware and behavioral description languages, then to check up the validity of these models compared to the original blocks, and finally to integrate the models into the whole circuit and perform a global simulation to check its functionality.

1.2 Host Company

Figure 1: ST Building

The training period extends over twenty-four weeks in STMicroelectronics Grenoble in France. STMicroelectronics is a global independent semiconductor company and a leader in developing and delivering semiconductor solutions across the spectrum of microelectronics applications.
The company was born from the 1987 merger of the Italian company SGS Società Generale Semiconduttori SpA and the French company Thomson Semiconducteurs a subsidiary of Thomson. It employs 46,000 creators and makers of semiconductor technologies that cover its 11 manufacturing sites.[1]

My internship was carried out in Grenoble site which now hosts many divisions (marketing, design, industrialization) and is an important RD center (design, software, research on manufacturing processes).

2 Theoretical background

2.1 Flash Memories

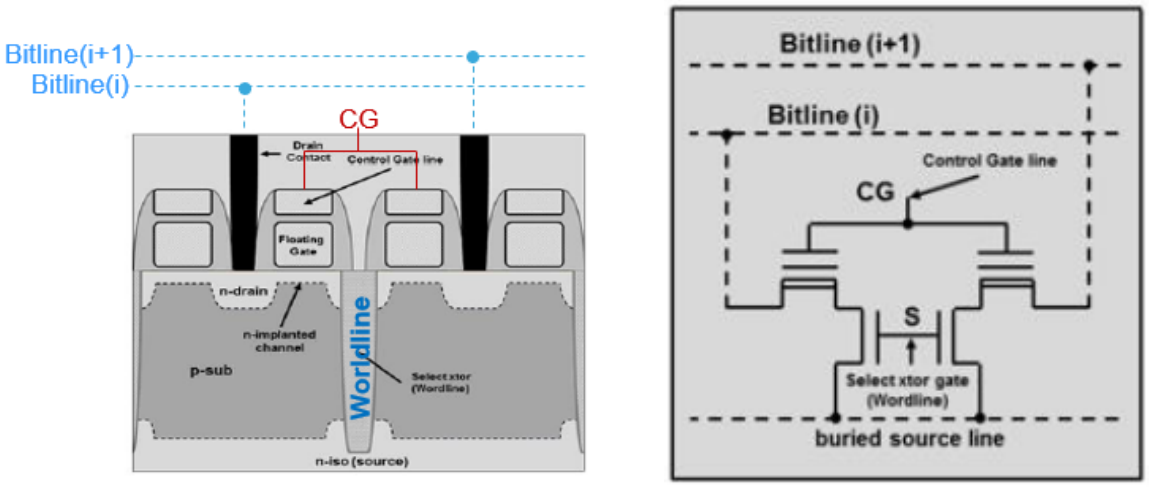
2.1.1 Overview

Flash memory is an electronic non volatile memory that was first developed by Toshiba in the early 1980 based on EEPROM (Electrically Erasable Programmable Read Only Memory). Tanks to its remarkable speed, it has evaded several devices ranging from smart phones to solid state drivers. It is based on a floating gate that tunes the switch on or off depending on the charge stored in it.

There are two main kinds of flash memory, NOR flash and NAND flash. The first one is based on a NOR logical gate for each bit cell of the memory array, this type of flash memory allows each single byte to be operated independently and shows a good performance during read operation with lower latency . The second one is based on a NAND logical gate for each bit cell, this type of memory may be operated in blocks or pages and has better storage density with lower price. NOR is used as a first support component in simple embedded systems while NAND plays the role of a second storage component in complex embedded systems such as SDD drivers.[5]

2.1.2 The eSTM memory

The eSTM is an innovative charge storage non-volatile memory cell conceived, developed and industrialized by STMicroelectronics that has allowed to achieve a breakthrough scaling.
It is based on flash NOR memory cells with a conventional polysilicon floating gate and a select transistor built vertically in the depth of the silicon and representing the wordline as shown in figure 2-a.



(a) Schematic cross-section of eSTM cell

(b) Device schematic of coupled cells arrangement

Figure 2: eSTM bitcell schematics

Each single cell is identified at the intersection of a BL and a WL. In fact, the structure of the vertical select transistor determines the vertical selection of the two rows of symmetrical cells simultaneously, and since each drain of these two cells is connected to a different bitline, the selection of the horizontal bitline would determine the selected cell as shown in figure 2-b.[8]

The eSTM cell gathers the advantages of a conventional 1T Flash Memory cell together with a more compact cellbit area outpacing state-of-the-art in term of bit-cell area.

The eSTM macrocell flash memory block is composed of two main sub-block: a digital controller and an analog core as sown in figure 3. It has two main supplies VDD ranging from 0.9 to 1.4V and VCC ranging from 1.5 to 5.5V.

The digital controller is composed of a micro core controller which executes a code stored in a ROM (Read Only Memory). It has access to registers that store the resources, configuration registers that store the trimmings and configurations needed for the different operations. It also has access to RAM memory in case the code needs to be patched. Several function for analog blocks managements are also implemented in the digital controller.

The hardmacro has two main sub blocks which are the eSTM memory array and the analog control circuits. The analog control circuit contains four charge pumps to generate supplies for other analog blocks, oscillators, current references and two HV (high voltage) blocks. Besides, the eSTM memory array is divided into two half arrays with a read circuitry in between.

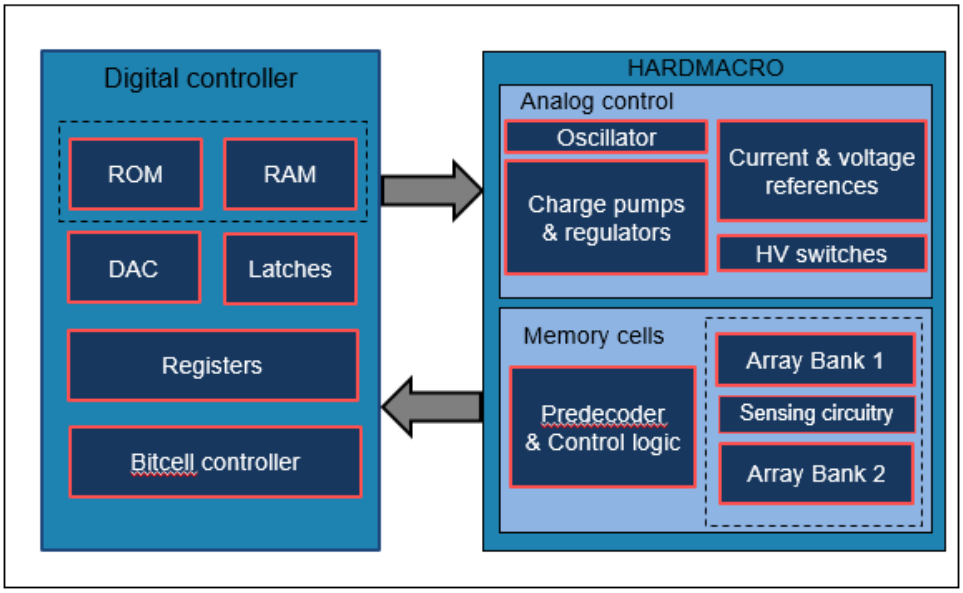


Figure 3: eSTM core organisation overview

The main operations of the cell are the read, program and erase operations.
The read operation is used to output the content of the cell. During this operation 2 words are read at a time by the sense amplifier which apply about 0.7V to the selected bitline of the cell while its worldline is biased to 3V, its control gate biased to 0V and its buried source is kept to GND potential. Besides, the measured drain current would characterize whether the cell is programmed if it is negative or erased if it's positive.

The Programming operation is equivalent to writing '0' in the array with a granularity down to 1 bit by introducing electrons into the floating gate.
During this operation the control gate is biased to a positive high voltage around 10V so that the left transistor according to figure 2 (b) is ON and that way, the potential on its source is equal to the potential at the drain of the right transistor which is equal to 4.5V and therefore the right transistor is in saturation. This condition generates hot electrons in its vertical drain region since they have enough kinetic energy to cross the thin oxide barrier and reach the gate. A vertical electric field is also generated thanks to the high control gate biasing and controls the injection of electrons to the floating gate.

The Erase operation is equivalent to writing '1' in the array with a granularity down to 1 page by subtracting electrons from the floating gate.
During this operation, the selected control gate line is biased to a high negative voltage equal to -10V. the unselected control gate world line is biased to the same positive high voltage 10V of the p-well and source in order to avoid any electrical noise from unselected pages. The physical mechanism involved is the Fowler-Nordheim effect.[8]

2.2 Charge Pumps

DC-DC voltage converters are very common in integrated circuits. LDO (Low Dropout Regulators) and SMPS (Switched Mode Power Supplies) are the main converters used for high to low conversion. On the one hand, LDO has a simple implementation technique, a low cost and low noise but it has a limited flexibility since it could only be used to step down voltages. On the other hand, SMPS has a better flexibility since it could step down or step up voltages with a high efficiency but it has a more complicated design with higher cost and noise.

In the case of low to high conversion, transformers followed by a filter have been used over the 100 past years. transformers were mainly used as AC-DC converters but being followed with a filter has allowed to use them as DC-DC converters. Although this technique was successful but with the continuous scaling down of circuits it becomes large and costly compared to the rest of the power conversion circuity it supports due to its inductive component.[6]

For this reason charge pumps which are compact inductor-less DC-DC converters are more suitable for IC design. A charge pump uses capacitors as an element that stores the energy.
There are many types of charge pumps, the main topologies are :[7]
• Two-stage charge pump with DC voltage supply and a pump control signal.
• Dickson charge pump with diodes.
• Dickson charge pump with MOSFETs
• PLL (Phase Locked Loop) charge pump.

Charge pumps are commonly used to provide a positive or negative voltage from low voltage power supplies which is the case in the eSTM memory for the VXR charge pump having VDD (0.9-1.4V) and VDDPUMP (1.5-2.1V) as the only two low power supplies and requiring voltages up to 20V for erasing aged cells. Therefore, the circuit is made of three charge pumps. The main one VXR pump which creates two high voltage supplies for analog blocks and generate the bitline biasing voltage for programming operations. Vyp that generates the positive supply voltage (10V) used for example to bias bitcell control gate during programming operations. And Vneg which is a negative charge pump that generates control gate negative voltages (-10V) for the erase operations.

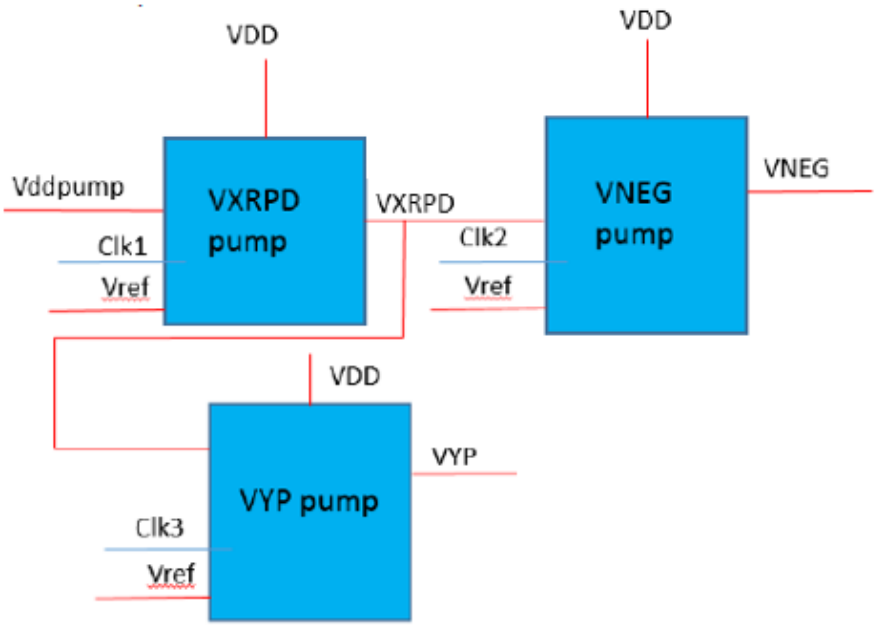


Figure 4: Power Tree

2.3 Circuit Verification

The previously presented block needs to be verified thereafter its design. This is done through mixed simulators.

2.3.1 General presentation

SPICE (Simulation Program With Integrated Circuit Emphasis) is the most important analog simulator in the word of microelectronics to check the integrity of circuit designs and to predict circuit behavior.
Although SPICE is a general purpose analog circuits simulator, it may has some limitations with large circuits as the run time exponentially increases with the circuit size. Besides, with the wide spread of mixed-signal designs, the need of simulating mixed analog-digital circuits

has arisen. Simulating the digital parts with full SPICE simulation is very time-consuming, that's why efficient event-driven techniques are used for simulating the digital parts. And mixed mode simulators glue together an accurate full SPICE analog simulator to an efficient digital simulator.[2][3]
The interaction between analog and digital parts is made through DAC (Digital Analog Converters) as well as ADC (Analog Digital Converters) that bridges the path between the modes and which are only virtual devices.

2.3.2 Existing models and simulations

Following the previous part, several blocks of the eSTM block are modeled and the figure 5 below points the modeled blocks in red while keeping the analog ones in blue.

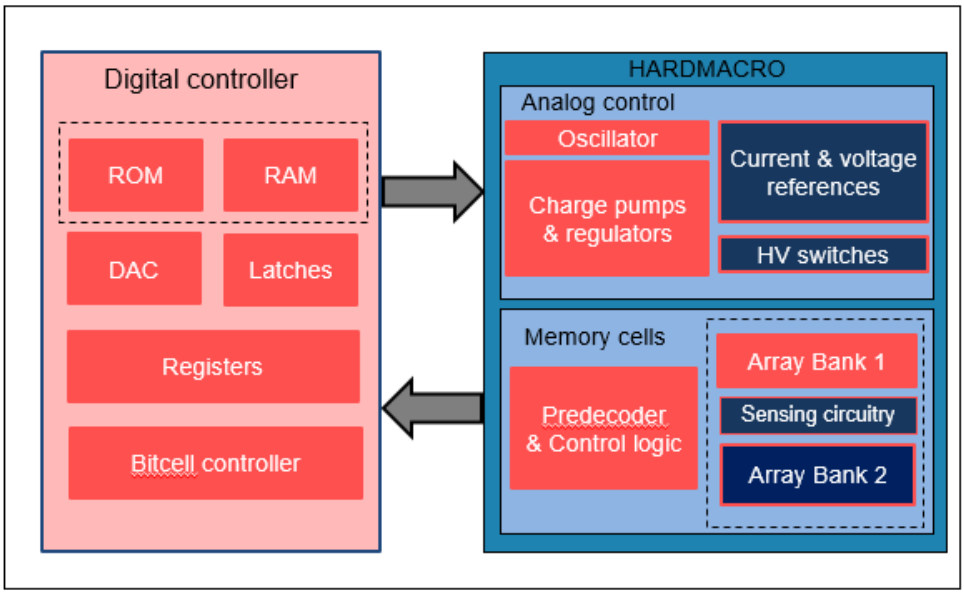


Figure 5: eSTM core organisation overview

The eSTM is simulated using this type of mixed simulation tools. Several existing simulations for the different operations of the memory already exist.
Full transistor level simulations are very long and could go up to 2 days for a "programming" simulation for example. For this reason a lot of sub parts are modeled which makes the simulation time shrink down to 20 min while limiting the loss of precision.
Some models already exist for this charge pump with short simulation times but these models don't provide any information about the currents consumption of the charge pump, and this is why this internship aims at figuring out a model with these missing information while keeping the same run time.

3 VXR Pump modelling

3.1 Theoretical presentation of the vxr pump

The VXR pump is made of three identical stages as the one shown in figure 6. One of these three stages can be by passed making the charge pump having two different operating modes according to the number of stages (either two or three).
The VXR pump is connected to a DCDC converter at its input and to two other charge pumps Vyp and Vneg at its output. It is also connected to the bitcell control blocks at its output since it would deliver different levels of supply voltages according to the type of operation done by the memory.

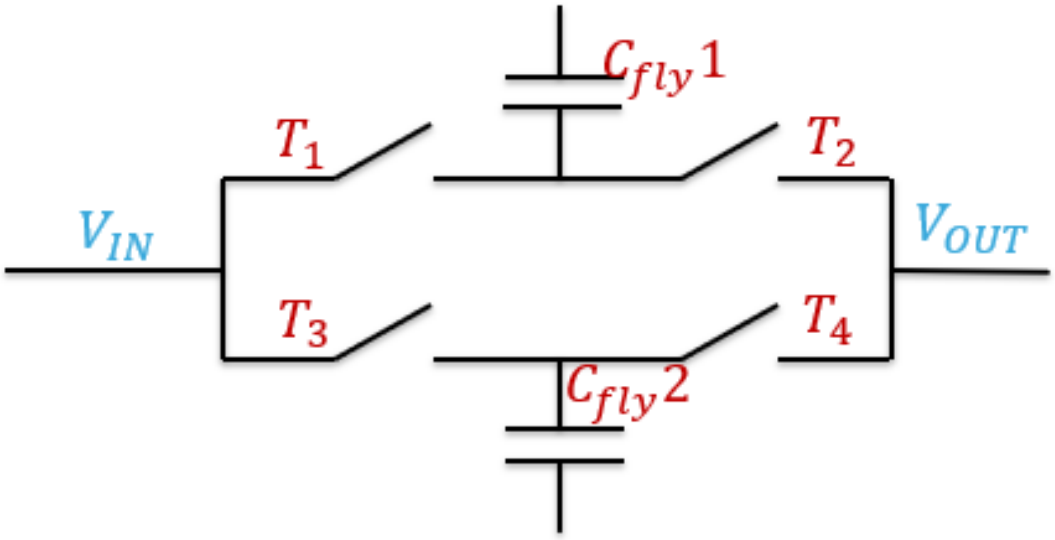


Figure 6: A simplified schematic of the VXR pump

The VXR charge-pump circuit uses two fly capacitors for storage in order to achieve higher voltages. The circuit has two states, which it continually switches between. The first state has two opposite switches on (T_1 and T_4 for example) and the two others off (T_2 and T_3 in this case), this is a charging state for the $C_{fly}1$. Then, in state 2 when T_1 and T_4 are now off and T_2 and T_3 on, the potential at the node between T_1 and T_2 becomes virtually equal to $2*V_{in}$ and is thereafter transferred to V_{out}. The same process occurs to $C_{fly}2$ but with phase opposition with respect to $C_{fly}1$ making V_{out} increase at each state.

3.2 Preliminary Study

The VXR Pump has two main voltage supplies V_{VDD} which ranges from 0.9 to 1.4V and $V_{VDDPUMP}$ which ranges from 1.4 to 2.1V. Added to an output voltage V_{VXR2} which is maintained constant at a certain level through a voltage regulator. Its value is triggered using a digital control bus (VxrCtl).
The whole purpose of this modelling is to estimate the two currents (I_{VDD} and $I_{VDDPUMP}$) delivered to the charge pump supplies. In order to do so all the parameters that may affect these two supply currents need to be determined first. 8 parameters are held as shown in table 1 below.

Voltage supplies V_{VDD} and $V_{VDDPUMP}$
Output current I_{VXR2}
Output voltage V_{VXR2} (Controlled by VxrCtl)
Temperature
Clock frequency
Number of stages
Process variation

Table 1: Different parameters affecting the two supply currents

The Output voltage V_{VXR2} is a discrete parameter that can take 47 different values and the number of stages is a discrete parameter as well that can be fixed rather to 2 or 3. for this reason these two parameters could be dropped of because a model will be found for each different value of them instead of being included into a general equation. Besides, the process variation is very difficult to model and the worst case (FFA) will be considered for the rest of the study. This makes the number of parameters shrink from 8 to 5.

3.2.1 Testbench and simulations

Modelling the behavior of this pump in term of current consumption by finding out an equation that relates 5 parameters is still very uncomfortable. For this reason, each parameter's impact on the input current needs to be studied independently and compared to other parameters' impacts in order to simplify the modelling by only keeping the critical parameters.

First of all, Cadence Virtuoso environment is used in order to create a testbench schematic for the existing instance of the VXR pump using some power supplies from the analog library added to an existing instance of current mirror. This schematic is shown in figure 7.
Then Eldo Premier engine from Mentor Graphics is used for simulations. Each parameter is fixed to an operating value except the one being studied, then a parametric sweep is applied to the studied parameter which will allow the two supply currents to be measured for different values of this parameter. And these two supply currents are extracted using additional commands.

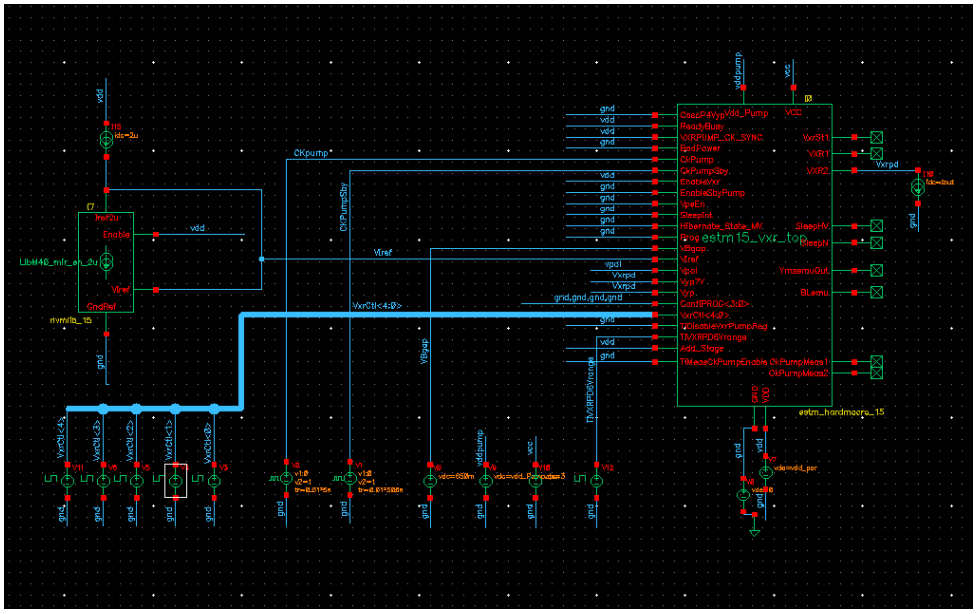


Figure 7: Testbench schematic of the VXR pump

3.2.2 Results

At nominal current values of the output, the clock frequency has no impact on the currents delivered by the two supply voltages V_{VDD} and $V_{VDDPUMP}$. This frequency gives the maximum pumping capacity, the higher the frequency is the higher the driving capability is.

The output current is the most critical parameter. However, according to its curve, the variation of the two supply currents with respect to it is linear with a steep slope so it could be added as a multiplying factor to the equation at the end. A constant could be added as well if the supply currents are significant when the output current is equal to zero.

The temperature and the two supply voltages are therefore the three main contributors. However, for the sake of simplicity, temperature is going to be dropped out in a first approximation taking the worst case corresponding to the case with the highest temperature. And the following parts are going to focus on the modelling of the currents variation with respect to V_{VDD} and $V_{VDDPUMP}$.

Parameter	$I_{VDDPUMP}$	I_{VDD}
V_{VDD}	7.6 %	26.14 %
$V_{VDDPUMP}$	7.3 %	29 %
Frequency	0 %	0 %
Temperature	3.18 %	14.97 %
I_{VXR2}	$\geqslant$ 100 %	$\geqslant$ 100 %

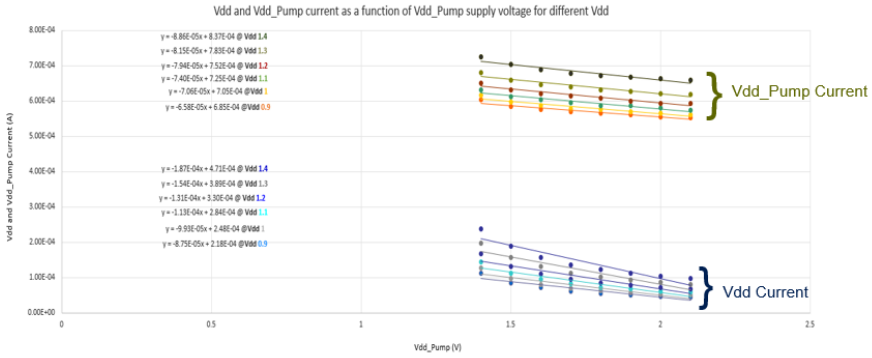
Table 2: Supply currents variation for each parameter

3.3 Different strategies

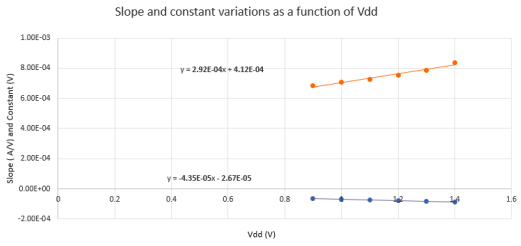
3.3.1 Explanation and development of two strategies

As explained in the previous section, this part is focused on the modelling of the two supply currents as a function of V_{VDD} and $V_{VDDPUMP}$.
The launched simulation aims at plotting I_{VDD} and $I_{VDDPUMP}$ as a function of V_{VDD} and $V_{VDDPUMP}$ each, and this simulation is done in the following conditions:
- V_{VXR2} = 3.26V - T=27∘C -f=200Mhz -I_{VXR2} =80μA.

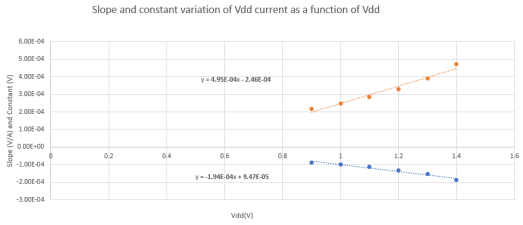
So the first step is to report the data of the simulation results to excel for observation. And by plotting point by point the supply currents as a function of $V_{VDDPUMP}$ for different values of V_{VDD}, one could clearly notice a linear behavior which leads to make a linear approximation for each curve as shown in figure 8-a. This leads to several different linear equations for different values of V_{VDD} whose slopes and constants depend on V_{VDD}. As a consequence, the variation of the slope as well as the constant as a function of V_{VDD} is plotted in figures 8-b and 8-c in order to find a general equation that relates each supply current with the two supply voltages and this results in equations (3.3.1.1) and (3.3.1.2).



(a) Linearization



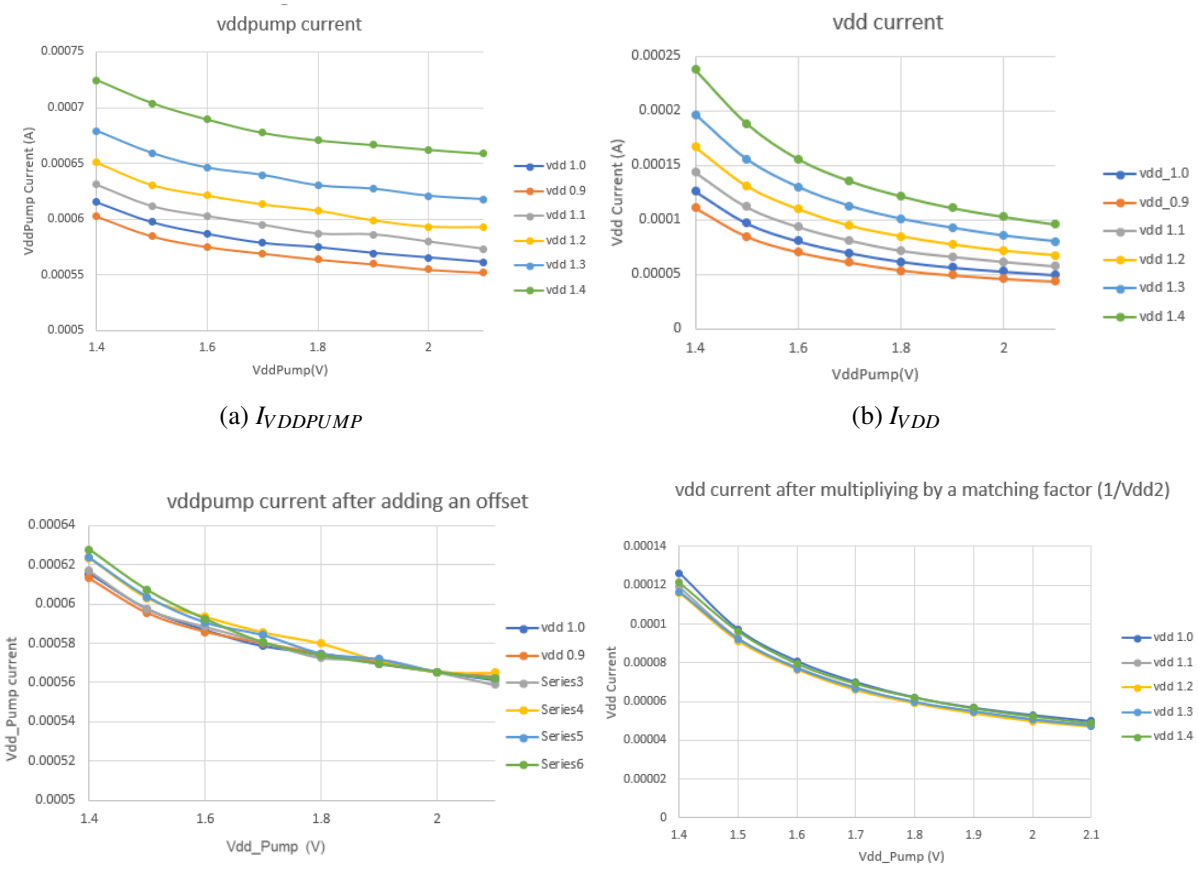
(b) Slope and Constant for $I_{VDDPUMP}$



(c) Slope and Constant for V_{VDD}

Figure 8: First strategy

One can notice that there is also the possibility of doing the same previous steps but by starting with plotting the currents as a function of V_{VDD} instead of $V_{VDDPUMP}$ then getting the expression of the slope and constant as a function of $V_{VDDPUMP}$. This possibility has been tested but it was much more complex to find an easy and representative modelling strategy in that case.



(a) $I_{VDDPUMP}$

(b) I_{VDD}

(c) $I_{VDDPUMP}$ after adding an offset

(d) I_{VDD} after adding an offset

Figure 9: Second strategy

So far, a first strategy has been explained and for which the results are going to be exposed in the next subsection. However, this isn't the only strategy that has been explored. In fact, coming back to plotting point by point the supply currents as a function $V_{VDDPUMP}$ for different values of V_{VDD}, the linear approximation of the curves could be pushed further to the second order resulting in more accurate curves as shown in figure 9.

However in that case plotting up the three constants as a function of V_{VDD} would result in a very rough equation, for that reason the strategy here is to find a way to "get from one equation to another".
In the case of the current delivered by V_{VDD}, according to the figure 9-b the curves are characterized by an hyperbolic trend and adding a multiplying factor would lead to a general equation. This factor is equal to ${V_{VDD}}^2$ because in order to get from any curve of figure 9-b to the curve at which V_{VDD} is equal to 1V, one needs to multiply by $\frac{1}{{V_{VDD}}^2}$, so after finding a second order linearization to the curve at which V_{VDD} is equal to 1V multiplying by ${V_{VDD}}^2$ would result in the general equation (3.3.1.4).
In the case of the current delivered by $V_{VDDPUMP}$, the curves are parallel as shown in figure 9-a and therefore adding an offset would generalize the equation. The expression of the offset is found by plotting the variation of the supply current offset of each curve of figure 9-a with respect to the one at which $V_{VDDPUMP}$ is equal to 2V as a function of V_{VDD}. Thus, this generates

the equation (3.3.1.3).

First strategy

$$I_{VDD} = -1.94*10^{-4}*V_{VDD}*V_{VDDPUMP} + 9.47*10^{-5}*V_{VDDPUMP} + 4.95*10^{-4}*V_{VDD} - 2.46*10^{-4} \quad (3.3.1.1)$$

$$I_{VDDPUMP} = -4.35*10^{-5}*V_{VDD}*V_{VDDPUMP} - 2.67*10^{-5}*V_{VDDPUMP} + 2.92*10^{-4}*V_{VDD} + 4.12*10^{-4} \quad (3.3.1.2)$$

Second strategy

$$I_{VDD} = V_{VDD}^2 * (1.267*10^{-4}*V_{VDDPUMP}^2 - 5.313*10^{-4}*V_{VDDPUMP} + 6.078*10^{-4}) \quad (3.3.1.3)$$

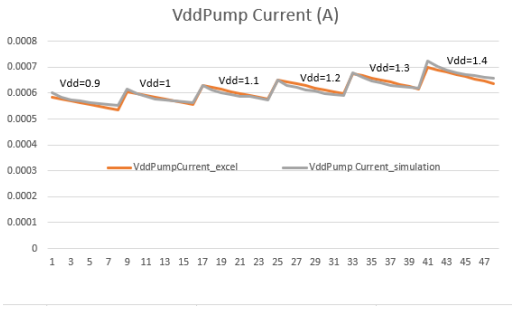
$$I_{VDDPUMP} = 9.78*10^{-5}*V_{VDDPUMP}^2 - 4.13*10^{-4}*V_{VDDPUMP} + 9.99*10^{-4} + 4.89*10^{-4}*V_{VDD}^2 - 9.38*10^{-4}*V_{VDD} + 4.51*10^{-4} \quad (3.3.1.4)$$

The previous equations for an output current I_{VXR2} equal to 80μA and therefore still need to be normalized in the following parts.

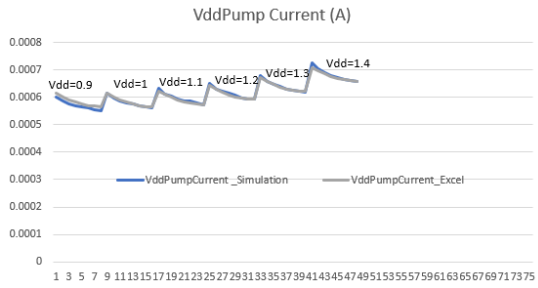
3.3.2 Results comparison of the two strategy

According to the previous section, two strategies were developed leading to two different equations for each supply current. In order to decide which strategy is better for the modelling, there is a need to compare the resulting currents from each equation to the ones resulting directly from simulations.

According to figure 10, the $I_{VDDPUMP}$ error of the second strategy is below 3% while the $I_{VDDPUMP}$ error of the first one is below 5 %. So both strategies are acceptable for $I_{VDDPUMP}$ modelling.
However, according to figure 11, I_{VDD} errors of the second strategy are below 12 % while the ones of the first strategy go up to 500 % . So the first strategy failed to model I_{VDD} consumption and therefore the second strategy is going to be kept to further the study.

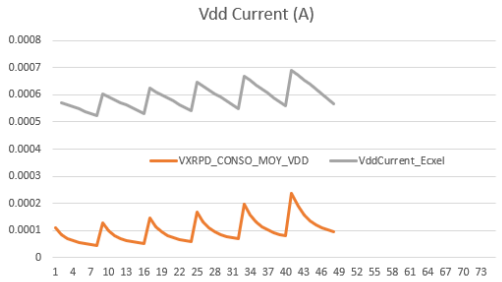


(a) First strategy

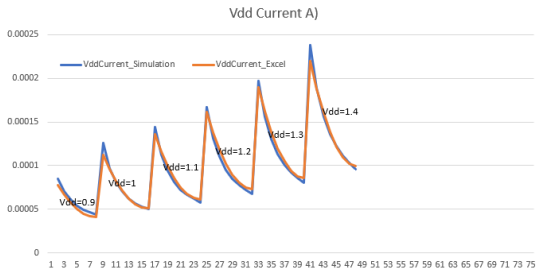


(b) Second strategy

Figure 10: $I_{VDDPUMP}$ comparison



(a) First strategy



(b) Second strategy

Figure 11: I_{VDD} comparison

3.4 Development of the chosen strategy

Up to here, a preliminary study was held to determine which parameters to focus on for the study leading to consider the two supply voltages as a first approximation.Then, two strategies were developed leading to choose the second one because it matches better with simulations. So, this last results in two equations of the two supply currents as a function of the two supply voltages. However, these equations are determined for a specific value of the output voltage V_{VXR2} and in the case of using the pump with three stages. Since there is 47 different trims leading to 47 different output voltages and the number of used stages may be 2 or 3, there is 94 different equations for each supply current.
Determining each equation separately using excel as done in the previous case is very time consuming. For this reason, using tcl scripts for automatizing has become essential.

3.4.1 Detailed script

The detailed script is in annexes but here the main steps of it.

The first step consists in putting the supply currents data when V_{VDD} is equal to 1 into a list. In order then to find each current as a function of $V_{VDDPUMP}$, a second order polynomial approximation is done. The three coefficients of this second order equation are found using Gaussian process for regression.

The second step consists in generalizing the equation to all values of V_{VDD}. In the case of I_{VDD}, the second order equation that corresponds to I_{VDD}(V_{VDD}=1V) is multiplied by the square of V_{VDD} to find a general one.
In the case of $I_{VDDPUMP}$ equation, a list of current offsets as a function of V_{VDD} when $V_{VDDPUMP}$ is equal to 2V is built. The offset is the difference of the current $I_{VDDPUMP}$ at a specific value of V_{VDD} with respect to the current $I_{VDDPUMP}$(V_{VDD}=1V). The same method of polynomial resolution is then used to find a second order equation of this offset as a function of V_{VDD}, and this equation is added to the previous one in order to obtain a general equation of V_{VDD} variation as a function of the two supply voltages.

The third step consists in applying the same previous whole process on different values of the output voltage V_{VXR2} (triggered the by digital control bus VxrCtl) leading to different equations. These equation are then gathered into an output text file.

The fourth step consists in verifying if the resulting equations of the tcl script match with simulation results. To do so, for each value of V_{VDD}, $V_{VDDPUMP}$ and V_{VXR2} the two supply currents are computed using the resulting equations from the scripts. These results are gathered into an output text file behind the simulation results and then the error between simulation and model results is computed and displayed as well in the output file.

An input text file needs to be filled before executing the script with the following informations:
• Number of rows of the csv file minus one.
• Number of columns of the csv file.
• Number of trims.
• Number of different values of $V_{VDDPUMP}$.
• Number of different values of V_{VDD}.
• The index of the first column of I_{VDD}.
• The index of the first column of $I_{VDDPUMP}$.
• Number of extractions of average values of current (In additional spice command tab while launching the simulation) .
• The index of the row of the first value of VxrCtl minus one.

This input file will allow to execute the script for the two configurations of the VXR pump (2 or 3 stages).
Before the execution of the script, the right name of the csv file needs to be put into the script. besides, all the commas in the csv file need to be replaced by a space.
Added to that, the csv file is generated right after the simulation by converting the Eldo simulation output file (aex file) into csv one.
It is also important to mention that this script is valid only when the different trims of the VxrCtl are extracted in rising order. For example, 11000 could only be followed by 11001.

3.4.2 Simulation and Results

The previous script was executed with all the possible cases of the VXR pump functioning including all the possible values of V_{VXR2} as well as all the possible values of the number of stages resulting in 94 different equations.

Three main simulations are held in order to obtain all the equations. The three simulation has the same worst case temperature equal to 105°, the same worst case of the process variation (so called FFA) as well as the same range of V_{VDD} [0.9 - 1.4]. Although two parameters differ between the three simulations and are summarized in table 3 below corresponding to the actual use cases of the VXR pump.

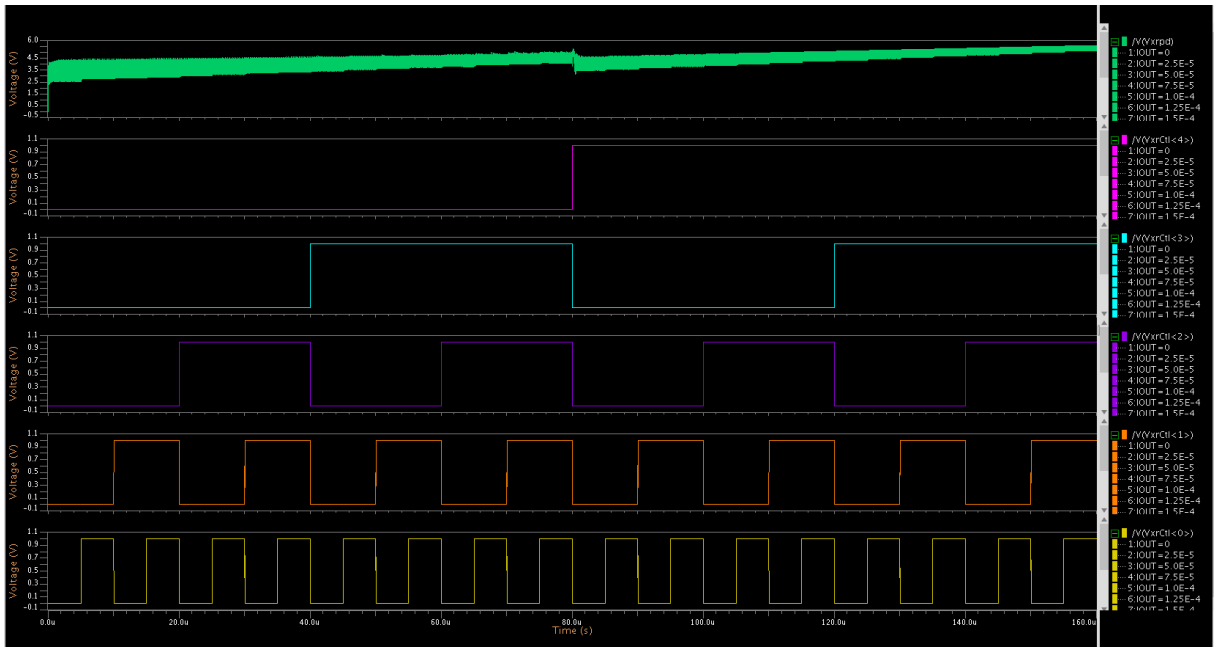


Figure 12: Transient results of simulation 1

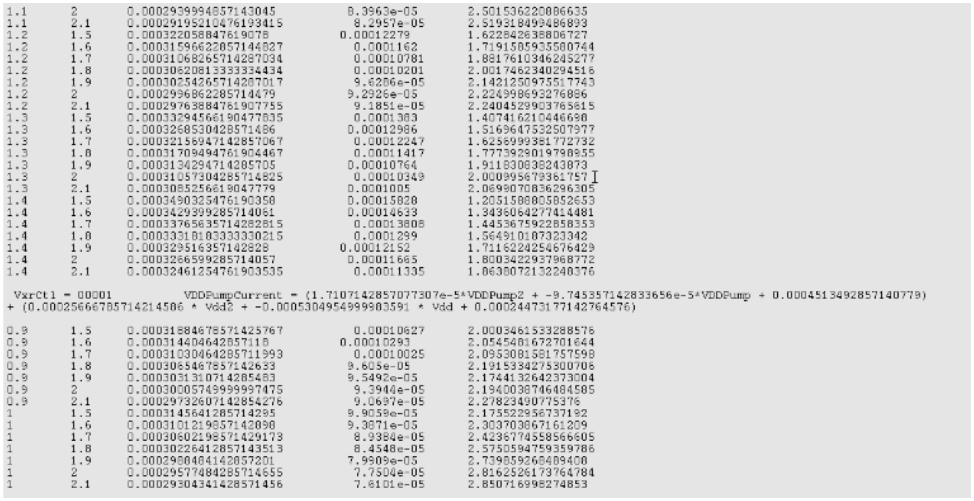
The transient analysis of simulation 1 of figure 12 lasts 160 μS making the simulation time of each value of V_{VXR2} 5 μS which is more than enough since the stabilization time is about 1 μS and therefore the supply currents are extracted 1 μS after the change of the value of V_{VXR2} each time. The first curve which is the green one of figure 12 shows the evolution of the output voltage V_{VXR2} depending on the value of VxrCtl control bus monitored by five bits as shown by the remaining curves.

Simulation	Simulation 1 (High performance mode)	Simulation 2 (Low perf read mode)	Simulation 3 (Low perf modify mode)
Number of stages	3	2	2
Range of $V_{VDDPUMP}$	[1.4 - 2.1]	[1.4 - 2.1]	[1.75 - 2.1]
V_{VXR2}	$\forall$	$\leqslant$ 4.47V VxrCtl(4)=0	$\geqslant$ 4.47V VxrCtl(4)=1

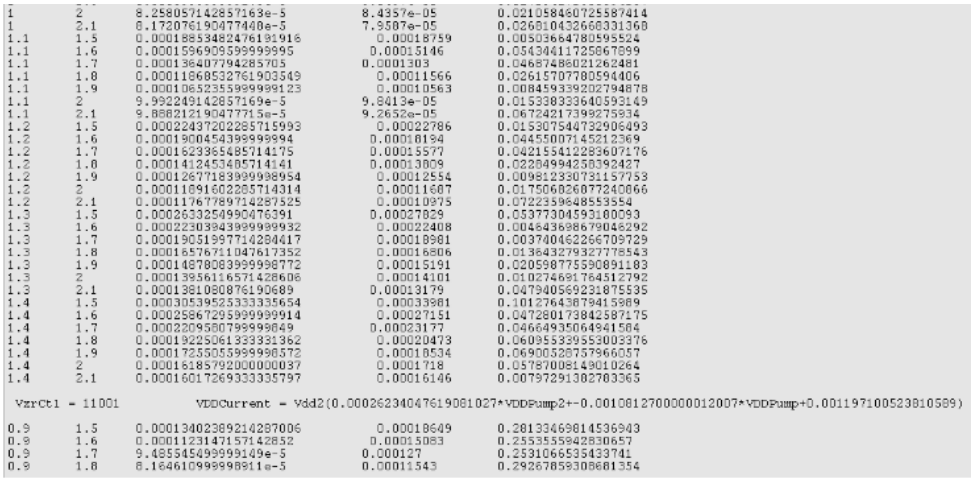
Table 3: Parameter variation for the different simulations

The execution of the script results in two text output files containing 5 columns with the tcl and simulation supply current values and the error between them according to the values of

V_{VDD} and $V_{VDDPUMP}$. One output file is for the supply current I_{VDD} and the other output file is for the supply current $I_{VDDPUMP}$. Each output file contains as well the Current consumption equation for each value of V_{VXR2}. Finally, it contains maximum errors which are summarized in table 4. According to this table, the maximum error for $I_{VDDPUMP}$ is equal to 3.982 % and the maximum error for I_{VDD} is equal to 0.972 % which is quite tolerable. So the tcl script current values are similar to the simulation ones and the model precision is satisfactory.



(a) $V_{VDDPUMP}$



(b) V_{VDD}

Figure 13: Extracts of the output text file

	$I_{VDDPUMP}$-3stages	I_{VDD}-3stages	$I_{VDDPUMP}$-2stages	I_{VDD}-2stages
Maximum Error	3.910 %	0.425 %	3.982 %	0.972 %

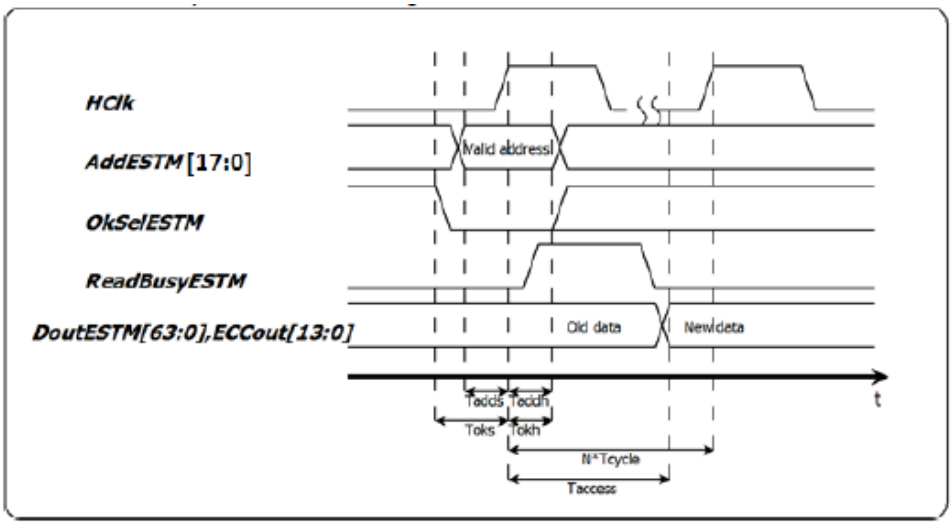
Table 4: Maximum Error

4 Integration of the model in mixed simulation environment

In the previous section, the supply currents consumption of the VXR pump were modelled by finding an equation that relates each current with the other parameters of the pump.
This section aims at integrating these equations into existing models of the pump using VerilogAMS description language into mixed simulation environment.

4.1 Testbench and simulations

Several simulations are already available for different purposes. And each simulation has a testbench in SystemVerilog description language.
Simulations that perform the ”erase” operation on full transistor level are very long (Up to 2 weeks) and are therefore not being to be performed since they don’t have a specific interest for this work. Besides, the ”read” operation is included in most of the ”erase” and ”prog” simulations, so there is no interest of simulating it independantly.
For this reason, only simulations that perform the ”prog” operation are going to be done in this part.



(a) Chronogram

```
task read_nvm ;
input [17:0] add;
input [63:0] expected;
begin
 AddESTM = add;
 OkSelESTM = 0 ;
#2   HClkESTM = 1;
#22   HClkESTM = 0;
#4  OkSelESTM = 1 ;
#34 OkSelESTM = 1 ;//waiting time to be sure doutESTM is properly updated (corresponds to 60ns delay for low cons read)
if (DoutESTM == expected )
 $display("<INFO ENVM>: OK read at address %h with  %h , as expected", add, DoutESTM);
else
begin
 $display("<INFO ENVM>: ERROR read at address %h with  %h , not as expected", add, DoutESTM);
 $stop;
end

end
endtask
```

(b) Code

Figure 14: ”Read” task

A simple ”prog” simulation without any additional functional constraints is picked up. Its

testbench is mainly made of a first part that declares the module of the memory block by declaring all its inputs and outputs as registers and wires. A digital sub block that contain the sequence of operations performed on the memory which are read, prog then read operation. Each operation takes in argument the address of the bitcell on which the operation is made and an expected value for auto-check. And they are all defined in another verilog file. Finally it instantiate the top level module using ”named” association.

Going back to the sequence of operations performed during the testench, one could check in details one of these operations.
For example the ”read” task has two inputs according to figure 14-b : ”add” which is the address of the read bitcell and ”expected” which is the expected read value from this bitcell. The reading starts on the positive edge of HClkESTM clock when OkSelESTM is active low that’s why it’s assigned to zero before HClkESTM is assigned to 1 after two sequences of time according to the first lines of the code.
Besides, when reading is finished ReadBusyESTM is reset and the new data is sampled in DoutESTM.
The testbench code then checks if this new data DoutESTM is equal to the expected one and displays an external message.

4.2 Scripts and simulations

This subsection aims at modifying the VerilogAMS script that describes the VXR pump model in order to integrate the new model exposed in previous sections adding information about the currents consumption I_{VDD} and $I_{VDDPUMP}$.

4.2.1 Code of the VXR pump existing model

The model of the VXR pump is described in VerilogAMS language which is a language that brings analog and digital modeling together. The potentials and flows for example are calculated in the continuous domain (analog) while other values like the register contents are calculated in the discrete domain (digital).

The code is made from several blocks such as ”initial” and ”always” blocks that describe a digital behavior and ”analog” block that describes the analog one. The interaction between the two previous domains (digital and analog) is done through variables and nets. In fact, analog variables and nets could be read from digital blocks and vice versa.[10]
The digital part of the code of the VXR pump model transforms the binary input (VxrCtl) into a decimal one and deduces the assigned value of the output voltage according to the value of VxrCtl and puts it into a variable. It also checks if the pump is ON or OFF based on its digital inputs.

The analog block starts by checking analog inputs such as the voltage value of VBgap, vdd and gnd. It is then composed of two main parts. The first one treats the case of the pump in ON mode, in that case, it assigns the values of the output voltages according to a trim table that has been hard coded according to the corresponding mode of functioning. The second part

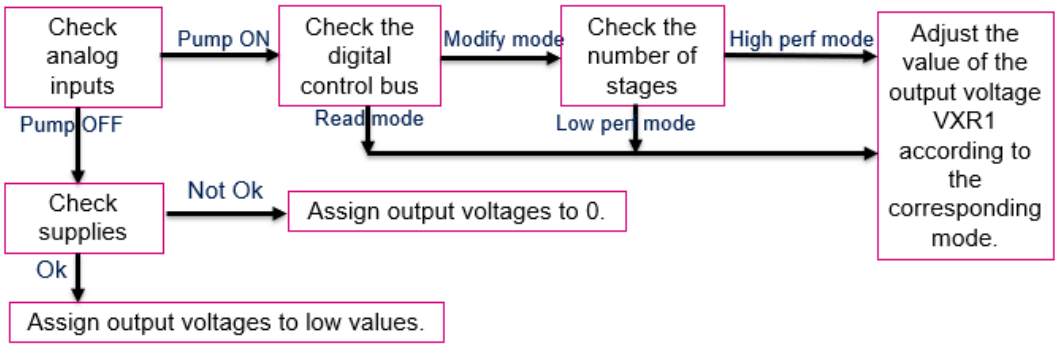


Figure 15: Existing model

treats the case of the pump in OFF mode and assign the values of the output voltages to zero when supplies are not ok or to other low values when the pump is simply OFF. Precision is not critical in OFF mode because this model is mainly intended to be used in ON mode. This is better represented in figure 15.

4.2.2 Current measurement schematic

The first step was to figure out a way to measure the output current flowing through the VXR2 pin of the VXR pump with VerilogAMS description language. To do so, an ideal internal node named VXR2_ideal is created (as shown in figure 16) on which the target V_{VXR2} voltage is applied (based on VxrCtl values). Then a command line 1 of table 5 is added in order to impose a certain current between VXR2_ideal internal node and VXR2 output port so that their potential difference is equal to zero. Doing so enables to indirectly control VXR2 output port voltage and measure the current flowing from it to VXR2_ideal. So command 2 measure this current and put its value into a variable.
Besides command 3 is a timing statement that allows to smooth variations of the VXR2_r variable with a certain rate (analog solvers do not work with instantaneous variations).
This also include adding some required probes in the .cir file in order to be able to plot the corresponding waves at the end of the simulation.

1- ”I(VXR2,VXR2_ideal):V(VXR2,VXR2_ideal) == 0.0;”
2- ”I_VXR2 = I(VXR2,VXR2_ideal);”
3- ”V(VXR2_ideal) <+ transition(VXR2_r, 2e-7, 2e-7);”

Table 5: Added command lines for current measurement

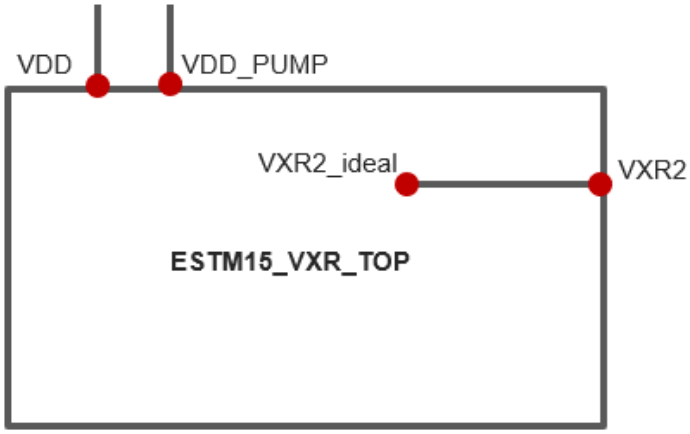


Figure 16: Current measurement schematic

4.2.3 Model integration

The second step was to integrate the model that was previously found of the input supply currents consumption and to check thereafter the VXR pump behavior through simulation results.

The first method of integration consists on manually integrating the related equations of the two supply currents according to the value of the VxrCtl. These two equations were added in the sub-part of the analog block were the pump is on working conditions.
This method was functional but poor in efficiency since it requires to check the output voltage value V_{VXR2} and then to look for the corresponding equation in the output text file.

For this reason the second method aimed at automatically calling the output files and deducing the corresponding equations according to the value of V_{VXR2}. VerilogAMS can't handle character chains so it was not possible to extract the corresponding coefficients of each equation from the existing text output file.

For this reason another output text file that contains only the coefficients of each equation was created as shown in figure 17 so that VerilogAMS could manipulate integers and registers only.

```
 1 4.0452380952594006e-5 -0.0001863285714293392 0.0005322619047625879 0.0002600037142785303 -0.0005411407857126283 0.0002523574285620795
 2 1.7107142857077307e-5 -9.7453571428333656e-5 0.0004513492857140779 0.0002566678571421458 -0.0005304954999983591 0.0002447317714276457
 3 1.0571428571902753e-5 -7.32142857159963e-5 0.0004308014285729522 7 0.0003061428571420255 -0.0006376779999980816 0.0003012047714274894
 4 -1.6488095237852424e-5 1.6389285713408497e-5 0.0003587716666674492 4 0.0003119035714277098 -0.0006495230714265838 0.0003058723714274502 7
 5 1.3654761904835684e-5 -8.386071428598052e-5 0.0004429959523811896 0.0002491214285707692 5 -0.0005010109999984791 0.0002179506857134277 6
 6 -1.0154761904950564e-5 2.8107142863972174e-6 0.0003658397619041516 0.0002843232142849303 6 -0.0005869179642839055 0.0002686206142846937 6
 7 -1.1488095238307291e-5 7.575000000763977e-6 0.0003643445238088441 0.0002988374999991767 -0.0006139282499981006 0.0002793572999989285
 8 -1.7880952381029154e-5 3.127142857170579e-5 0.0003442109523807052 4 0.0003486249999990471 -0.0007323333785712087 5 0.0003475612285701886
 9 1.6083333333558975e-5 -8.512500000081523e-5 0.0004443816666673939 6 0.0003157732142848617 6 -0.0006616715357123197 0.0003084124714274621
10 -5.500000000463286e-6 -1.1871428569758882e-5 0.0003834871428556571 6 0.0002744910714278290 6 -0.0005558431785697159 0.0002405475571418909 4
11 2.3059523809432733e-5 -0.0001085249999996728 1 0.0004668811904759002 3 0.0002453178571421865 4 -0.0004783364999984527 4 0.0001882611714276985 4
12 2.309523809533433e-5 -0.0001120428571432044 5 0.0004733333333336429 0.0002885607142849272 -0.0005770819285696128 0.0002422064857132613 5
13 4.4333333333265e-5 -0.0001965357142854678 6 0.0005575538095235901 0.0002774035714278017 -0.0005490359285696526 0.0002213406571418553
14 6.020238095224345e-5 -0.0002500107142852181 6 0.0006028554761900341 0.0002857999999923386 -0.0005656442857125181 6 0.0002260164285704314 2
15 6.3154761904395354e-5 -0.0002632249999986782 0.0006189488095226325 0.0002513303571421534 4 -0.0004832326785698049 0.0001732622142847982 8
16 6.240476190462258e-5 -0.0002659999999994953 0.0006272152380947866 0.0003850232142845781 -0.0008107993928545211 0.0003672007571414137 8 0
17 3.250000000022095e-6 -5.058214285722302 4e-5 0.0004829107142857861 0.0003400142857133684 -0.0006688317142835979 0.0002779138285702346 3
18 6.285714285877102e-6 -6.212857142916133e-5 0.0004966357142862421 0.0003590464285704655 4 -0.0007081159285692067 0.0002955732285701751 4
19 3.472619047592003 6e-5 -0.0001650892857133104 5 0.0005909288095229411 0.0004280517857131356 -0.0008555508214259118 0.0003690136142842175
20 2.3142857143173746e-5 -0.0001239357142868 56 0.0005581385714295881 0.0004368410714273684 -0.0008719173214257959 0.0003723270714270057
21 4.102380952361885e-5 -0.0001924499999993125 0.0006255147619041498 0.0004323821428559792 5 -0.0008646512142830297 0.0003644611999984856 6
22 7.272619047625018e-5 -0.0003054178571430715 0.0007274230952382853 0.0004401732142845172 -0.0008774121071400953 0.0003632701285690706 3
23 6.528571428583695e-5 -0.0002863285714290132 4 0.0007212728571432501 0.0004625214285701797 -0.0009206404285685471 0.0003758470285698029 4
24 7.082142857112248e-5 -0.0003121535714274669 0.0007526792857133016 0.0004758482142844232 -0.0009475123214255925 0.0003806514999983196
25 0.0001081309523806779 1 -0.0004495464285704373 0.0008820411904753064 0.0004525714285701861 5 -0.0008940908571399907 0.0003403327714269543
26 0.0001278809523813365 3 -0.0005269785714299545 0.0009600761904774198 0.0004928249999986001 -0.0009854120714253415 0.0003800822571410349 3
27 0.0001581190476191926 74 -0.0006495214285717119 0.0010085212380952631 0.0005107874999985213 -0.0010258072499996588 0.0003876788999980751
28 0.0001885238095243215 3 -0.0007692642857161313 0.0012058004761921187 0.0005256535714271435 -0.0010540614999996705 0.0003823623142838551
29 0.0002580476190478534 -0.0010365000000000842 0.0014653852380959848 0.0005212446428556403 -0.0010398929642822477 0.0003465118999980445 4
30 0.0003126428571433365 -0.0012551285714302966 0.0016850571428586675 0.0005534696428555414 -0.0011104033214248766 0.0003570251857122006 3
31 0.0004897738095241851 -0.0019259750000013465 0.0023214011904773824 0.0005426964628556546 -0.0010798671071394214 0.0002883684142837759
32 0.0006784523809535667 -0.0026562071428614083 0.0030270676190514074 0.0005431749999982034 -0.0010755670714244251 0.0002152425714051718
```

Figure 17: Output text file containing only the equations' coefficients for $I_{VDDPUMP}$

Besides, the detailed code of the second method is in annexes with the added part encircled in red but here are its main three steps.
The first step starts with creating six empty lists for the coefficients of $I_{VDDPUMP}$ and three other additional empty lists for the coefficients of I_{VDD}. It continue with opening the text file of figure 17 and checking if its empty. If not, it extract each line independently and puts its coefficients in the corresponding lists. And finally it closes the text file.
The second step consists in checking continuously during the simulation the value of VxrCtl and extracting each time its corresponding coefficients from the previously created lists to put them into final variables.
The third step consists in using the final variables to assign the values of I_{VDD} and $I_{VDDPUMP}$ in the sub-part of the analog block.

The whole code is in annexes 3 but figure 18 shows the differences between the manual (a) and automatic (b) integration in the analog sub part.

```
                I_VDDPump = -((0.00073182394924074 97*VDD_PUMP_r*VDD_PUMP_r-0.0029792740681771623*VDD_PUMP_r+
0.003344868608225653+0.0005160854166653283*VDD_r*VDD_r-0.0010466059583303699*VDD_r+0.0005233964999983842)*(I
_VXR2/0.00008));
                I_VDD = - ((VDD_r*VDD_r*
(0.0013530417129173668*VDD_PUMP_r*VDD_PUMP_r-0.0055159201982210 83*VDD_PUMP_r+0.005695330221616578))*(I_VXR2/
0.00008));
```

(a) Manual integration

```
                I_VDDPump = -((coeff_final_1*VDD_PUMP_r*VDD_PUMP_r-coeff_final_2*VDD_PUMP_r+
coeff_final_3+coeff_final_4*VDD_r*VDD_r-coeff_final_5*VDD_r+coeff_final_6)*(I_VXR2/0.00008));
                I_VDD = - ((VDD_r*VDD_r*
(coeff_final_VDD_1*VDD_PUMP_r*VDD_PUMP_r-coeff_final_VDD_2*VDD_PUMP_r+coeff_final_VDD_3))*(I_VXR2/0.00008));
```

(b) Automatic integration

Figure 18

4.3 Results

4.3.1 Functional Check

A first verification of the new model could be done through the messages displayed in the transcript of the output window of the simulator that are the following.
”OK read at address 14000 with ffffffffffffffff , as expected”
”OK read at address 14000 with ffffffff00000000 , as expected”
14000 being the address of the read bitcell and ffffffffffffffff/ffffffff00000000 the expected value each time. These messages indicate that the first and second ”read” operations were done successfully and that the programation of the bitceel from ffffffffffffffff to ffffffff00000000 was successful.
Besides, a functional check needs to be done by verifying the voltage levels of the bit cell. This could be done by plotting the control gate, bitline and worldline voltages as shown in figure 19.

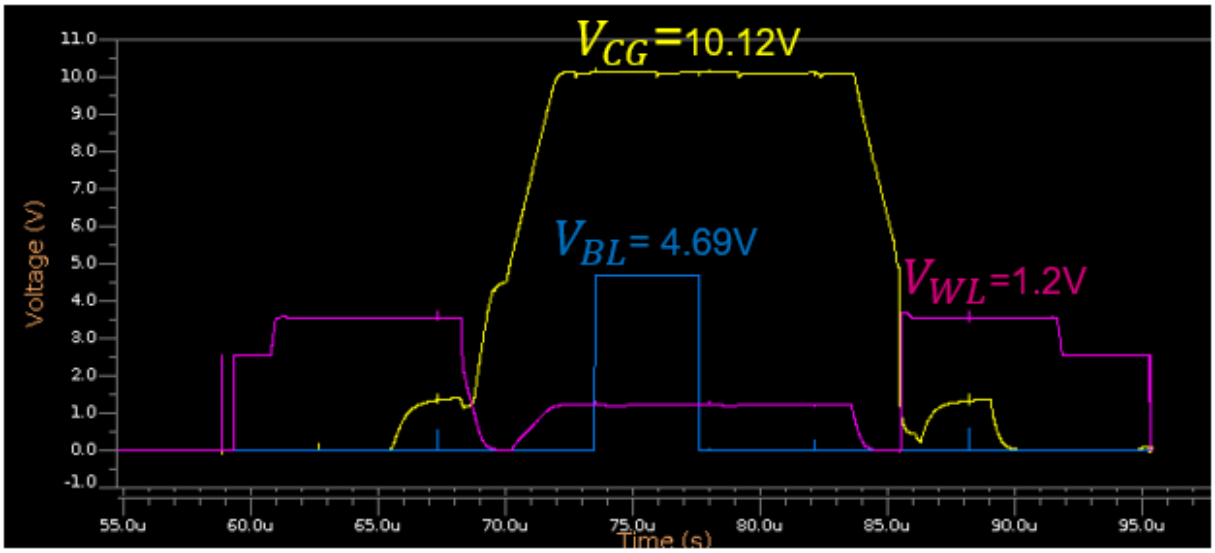


Figure 19: Functional check of the model

These previous voltages values as well as the bitline current are then compared to the expected ones for a programming operation. According to table 6 simulation values matches the expected theoretical ones and therefore the model is functionally correct.

	Control Gate voltage	Bitline Voltage	Worldline voltage	Bitline Current
Simulation value	10.12 V	4.69 V	1.2 V	5.12 μA
Theoretical value	10 V	4.7 V	1.3 V	5 μA

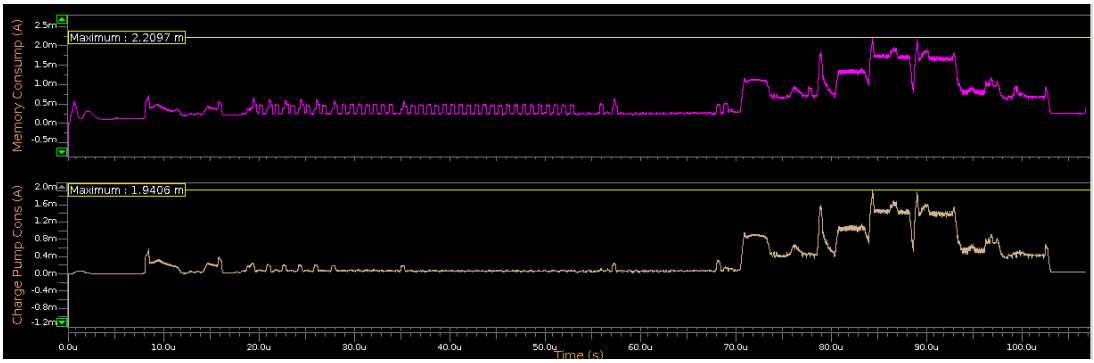
Table 6: Bitcell voltages check

4.3.2 Currents consumption

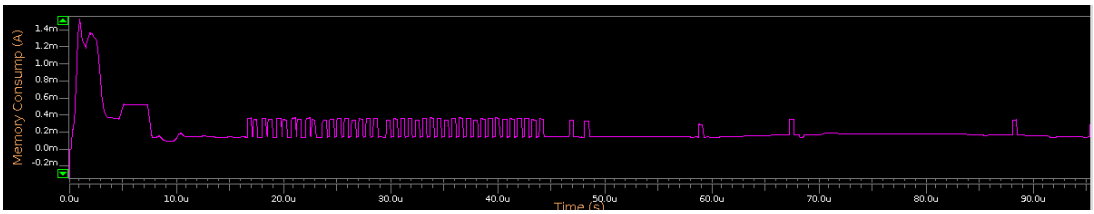
Besides, currents consumption of the VXR pump as well as of the whole memory need to be checked because they need to be inferior to 2.2mA according to the specifications of the eSTM macrocell.

To start with, the curves of figure 20-a are plotted in the case of full transistor simulation and are therefore a reference to which the old as well as new models are going to be compared. Besides, one may notice that programming operations start after about 60 μ from the beginning of the simulation and the first part corresponds to the ”power on” phase.

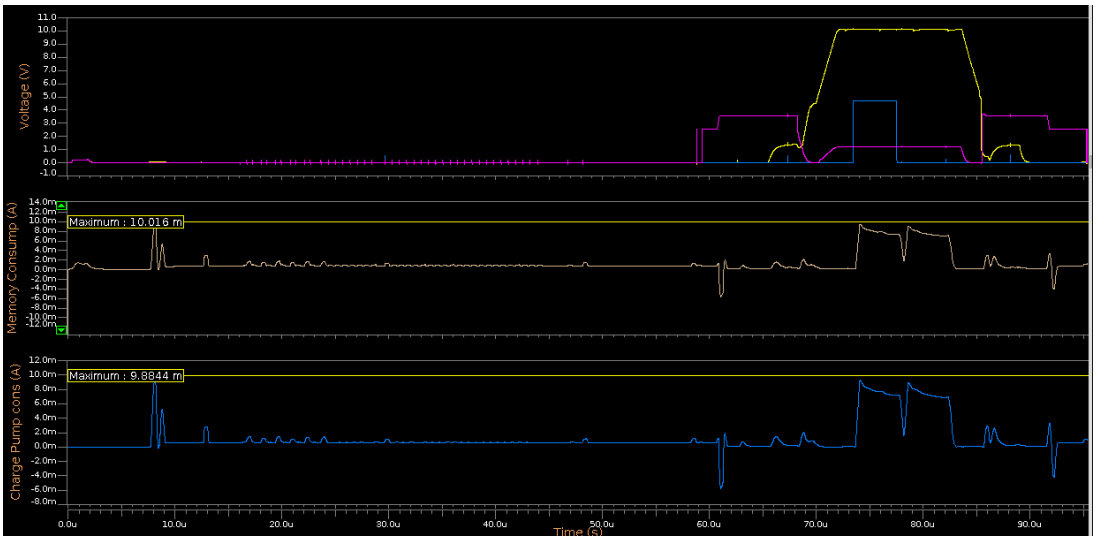
In the case of the old model simulation, the VXR charge pump model didn't take into account supply currents consumption by assuming ideal voltage sources and the values of the input currents of the VXR charge pump I_{VDD} and $I_{VDDPUMP}$ are therefore equal to zero in that case. This also affect the whole memory current consumption. In fact according to figure 20-b, the curve that plot the sum of input current of the eSTM block (I_{VCC} and I_{VDD}) doesn't match the reference curve and held lower current values with only pulses during the power on phase and not any consumption during the operations.



(a) Full Transistor Level Simulation



(b) Old Model Simulation



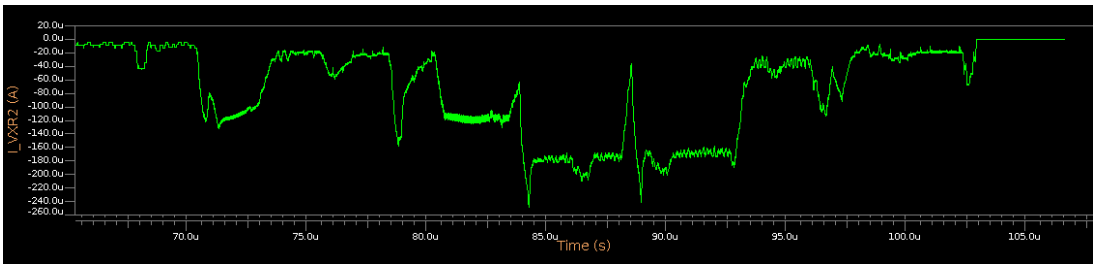
(c) New Model Simulation

Figure 20: Input Current consumption check (I_{VDD} and $I_{VDDPUMP}$)

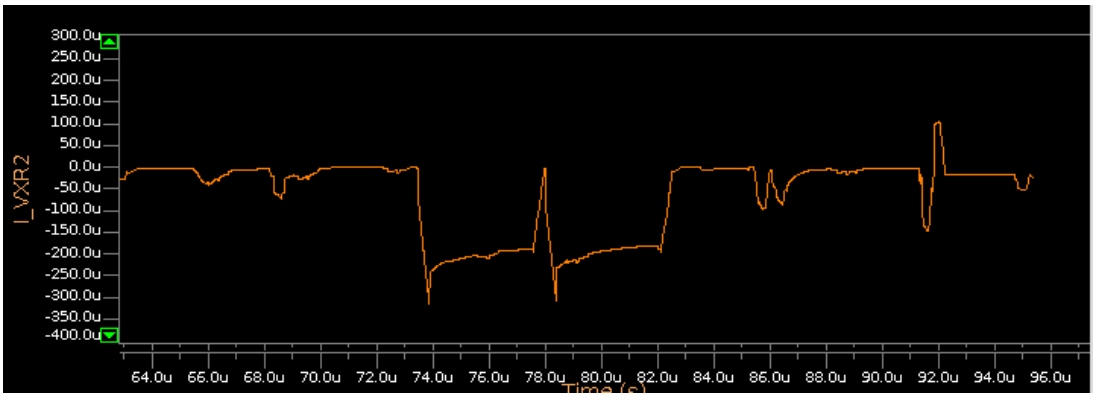
In the case of the new model simulation current consumption of the VXR pump as well as the eSTM block are shape alike the reference. however, they are equal to 10.02μA and 9.88μA which is quite above the maximum value equal to 2.2V. This is due to the fact that the value of I_{VXR2} in the new model is higher than the real one.

So far, only input currents consumption were checked. The output current consumption of the new model are plotted in figure 21-b just below the reference curve. In the case of the full transistor level simulation, the value of this current goes down to about 180μA during ”prog” operation in the case of high performance mode (using the 3 stages of the pump). The VYP charge pump (fed by the VXR one) only needs 60μA during its operation. Then 20 cells are programmed consuming 5μA each on VXR2 so that makes the output current consumption

goes up to 160 μA. And the remaining 20 μA are consumed by the other devices connected to the VXR charge pump.
In the case of new model simulation, the value of the output current consumption is about 205 μA during ”prog” operation. This values difference may be due to the fact that despite the VXR charge pump other blocks are modelled giving rise to additional errors. The old model holds zero as a value of the input current as explained in the previous part and is therefore not plotted.



(a) Full Transistor Level Simulation



(b) New Model Simulation

Figure 21: Output current consumption I_{VXR2}

4.3.3 Simulation time

Although the simulation time of the new model is some minutes longer than the simulation time of the old one, it still in the same order of magnitude and by far manageable compared to the speed of the full transistor level simulation.
Therefore, the new model has succeeded in keeping a reasonable simulation time.

Simulation	Full Transistor Level	With the old model	With the new model
Simulation time	30h 49mn	49mn	1h 23mn

Table 7: Simulation time comparison for the same CPU frequency 3 GHz

5 Conclusion

First, a strategy that models the currents consumption of the charge pump (I_{VDD} and $I_{VDDPUMP}$) was developed leading to 94 different equations (according to the value of V_{VXR2} and number of stages) for each current as a function of three parameter: the two supply voltages (V_{VDD} and $V_{VDDPUMP}$) as well as the output current I_{VXR2}. This model was done in the worst case in term of temperature and process variation.

Then, The previous model was integrated in mixed simulation environment in an already existing old model using VerilogAMS description language. Along with, a measure of the output current of the charge pump was also added to the model in order to be used in the equations and allow the computing of the the input currents.
This leads according to the simulation results to a correct functioning of the new model. Nevertheless, the values of the input current consumption were larger than the expected ones.
However, this model allowed keeping a similar simulation speed while improving the charge pump model.

Besides, only one single charge pump was modeled over three, so for a further study, this model could be adapted to the two other charge pumps (Vyp and Vneg). It may also be adapted to other charge pumps in other circuits.

6 Personal comments

6.1 Personal conclusion

Although this internship was not my first working experience related to my master degree, it was the first in microelectronic field.
I was surprised and challenged at the same time to see that my acquired knowledge in microelectronics during my master was a drop in the ocean. However, my academic training especially during EPFL semester was an essential basis that has laid the ground for this job.

There is a great cohesion within STMicroelectronics based on a hierarchical structure that guarantee the organization and progress of the different projects. Good communication and mutual share and help between members of each team as well as as between different teams working on different projects are keys to maintaining ST a leader in semi-conductor industry. And working with people on other sites abroad doesn't make a barrier to this group working. Weekly meetings as well as update and work presentation meetings are also constantly held to contribute to the well functioning of this structure.

Considering the special circumstances that we've been facing, ST was very reactive to the constant change of events advocating the safety of its employees first and the company interests second. Besides, I was able to carry on in the best conditions my internship after the lockdown through teleworking thanks to the provided screen and computer as well as the constant exchange with my supervisor.

6.2 Acknowledgments

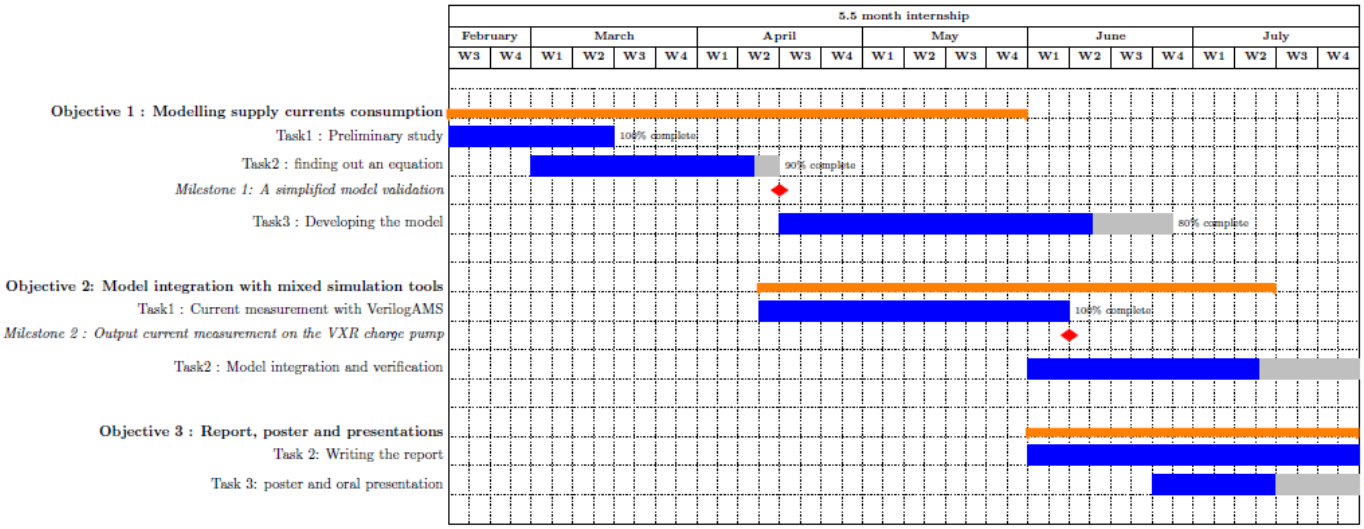
First of all, I would like to thank my supervisor Thomas Jouanneau as well as my manager Christophe Forel for welcoming me within their team and giving me the chance and opportunity to be part of this project. I also would like to thank again my supervisor for his constant technical help and advice as well as more general exchange which gave me a clearer and more global vision of the profession.
Then, I would like to thank other team members for their kindness and availability whenever I needed help.
Finally, I'm grateful for the financial and material means that were provided by ST and that were essential for carrying on this internship.

7 Bibliography

1. https://fr.wikipedia.org/wiki/STMicroelectronicsGrenoble_et_Crolles,_France
2. https://en.wikipedia.org/wiki/SPICECommercial_versions_and_spinoffs
3. Electronic Design Automation for Integrated Circuits Handbook - Louis Scheffer, Luciano Lavagno, and Grant Martin - EDA for IC Implementation, Circuit Design, and Process Technology.
4. https://www.sciencedirect.com/topics/computer-science/flash-memory
5. https://www.powerelectronictips.com/faq-what-is-a-charge-pump-and-why-is-it-useful-part-1-faq
6. https://en.wikipedia.org/wiki/Charge_pump
7. 40nm embedded Select in Trench Memory (eSTM) Technology Overview - F. La Rosa1, S. Niel2, A. Regnier1 , F. Maugain1, M. Mantelli1 and A. Conte3
8. E40 eSTM Macrocell 648KB for K470 Implementation Specification and Handbook.
9. http://www2.ece.ohio-state.edu/ bibyk/ece822/verilogamsref.pdf.

8 Annexes

8.1 Gantt Chart



8.2 Tcl code

```
#File generating different equations for different VxrCtl

set fp [open "/prj/mms_e40/K470/dev/users/slimiine/scripts/Vxrpd_bis.csv" r]
package require math::linearalgebra
set In [open "/prj/mms_e40/K470/dev/users/slimiine/scripts/Inputs.txt" r]

#First replace all the "," of the csv file with " "
#This works when we have only two parametric sweeps VDD and VDDPump supply voltages and when the current is as a function of
VDDPump for different values of VDD

#Creating an empty list that will contain the input parameters
set List_inputs { }
for {set i 0} {$i<9} {incr i} {
        gets $In data
        lappend List_inputs [lindex $data 1]
        }
set N1  [lindex $List_inputs 0]
set N2  [lindex $List_inputs 1]
set N3  [lindex $List_inputs 2]
set N4  [lindex $List_inputs 3]
set N5  [lindex $List_inputs 4]
set N6  [lindex $List_inputs 5]
set N7  [lindex $List_inputs 6]
set N8  [lindex $List_inputs 7]
set N9  [lindex $List_inputs 8]

#Building an empty array and putting the data of the .csv file inside
for {set j 0} {$j<$N1} {incr j} {
        for {set i 0} {$i<$N2} {incr i} {
                set array($j,$i) 0
                }       }
gets $fp data
for {set i 0} {$i<$N1} {incr i} {
        gets $fp data
        set j 0
        foreach chiffre $data {
        set array($i,$j) $chiffre
        incr j}   }


#Building all the necesary lists for VDDCurrent. This builds the two vectors X and Y for which we will do a second order linearizat
set list_X { }
set list_Y { }
set j $N6
while {$j<$N2} {
        for {set i $N4} {$i< [expr $N4 + 2 ]} {incr i} {
                lappend list_X $array($i,2)
                lappend list_Y $array($i,$j)}
         set j [expr $j + $N8 ]
}
```

```
#Building all the necesary lists for VDDPumpCurrent
set list_X1 { }
set list_Y1 { }
set list_X2 { }
set list_Y2 { }
set j $N7
while {$j<$N2} {
        for {set i $N4} {$i< [expr $N4 * 2 ]} {incr i} {
                lappend list_X1 $array($i,2)
                lappend list_Y1 $array($i,$j)}
         set j [expr $j + $N8 ]
}
set j 3
while {$j<$N2} {
        set i [expr $N4 - 1 ]
        while {$i<$N1} {
                lappend list_X2 $array($i,1)
                lappend list_Y2 [expr $array($i,$j)- $array(14,$j)]
                set i [expr $i + $N4 ]
                }
         set j [expr $j + $N8 ]
}

#Polynomial resolution of the second order. This is a general polynmial resolution that could be used several times

proc build.matrix {xvec degree} {
    set sums [llength $xvec]
    for {set i 1} {$i <= 2*$degree} {incr i} {
        set sum 0
        foreach x $xvec {
            set sum [expr {$sum + pow($x,$i)}]
        }
        lappend sums $sum
        #puts $sum
    }
    set order [expr {$degree + 1}]
    set A [math::linearalgebra::mkMatrix $order $order 0]
    for {set i 0} {$i <= $degree} {incr i} {
        set A [math::linearalgebra::setrow A $i [lrange $sums $i $i+$degree]]
    }
    return $A
}

proc build.vector {xvec yvec degree} {
    set sums [list]
    for {set i 0} {$i <= $degree} {incr i} {
        set sum 0
        foreach x $xvec y $yvec {
            set sum [expr {$sum + $y * pow($x,$i)}]
        }
        lappend sums $sum
    }
```

```
    set x [math::linearalgebra::mkVector [expr {$degree + 1}] 0]
    for {set i 0} {$i <= $degree} {incr i} {
        set x [math::linearalgebra::setelem x $i [lindex $sums $i]]
    }
    return $x
}


#Equation generation for a particular value of Vxrpd. This is a general fonction that generate the second order coefficients for a
proc generate.equation {list_X list_Y} {
        set degree 2
        set A [build.matrix $list_X $degree]
        set b [build.vector $list_X $list_Y $degree]
        set coeffs [math::linearalgebra::solveGauss $A $b]
        return $coeffs
        }


#Creating an empty array of the results first column VxrCtl second VDD third VDDPump fourth EqustionVDD fifth EqautionVDDPump
for {set j 0} {$j<$N3} {incr j} {
        for {set i 0} {$i<4} {incr i} {
                set array_equation($j,$i) 0
                }       }

#Generating all the equations of VDDCurrent for different values of VxrCtl and putting them in the array named array_equation that
set l [llength $list_Y]
set degree 2
set list_labels {00000 00001 00010 00011 00100 00101 00110 00111 01000 01001 01010 01011 01100 01101 01110 01111 10000 10001 10010
set k 0
set i 0
set lab $N9
while {$i< $l} {
        set j [expr $i + [expr $N4 - 1 ] ]
        #6 is N4-1
        set list_X_bis [lrange $list_X $i $j]
        set list_Y_bis [lrange $list_Y $i $j]
        set coeff [generate.equation $list_X_bis $list_Y_bis]
        set VxrCtl [lindex $list_labels $lab]
        set array_equation($k,0) $VxrCtl
        set array_equation($k,1) $coeff
        #puts $array_equation($k,0)
        #puts $array_equation($k,1)
        #puts [lindex $array_equation($k,1) 0]
        #puts "VDDCurrent_VxrCtl_$VxrCtl = VDD^2([lindex $coeff 2]*VDDPump^2+ [lindex $coeff 1]*VDDPump + [lindex $coeff 0])"
#Check the order of the coeffs
        set i [expr $i + $N4 ]
        set k [expr $k + 1 ]
        incr lab
        }
```

```
#Generating all the equations of VDDPumpCurrent which means two second order polynomial resolution for each value of VxrCtl and
putting them in the same results array named array_equation.
set l1 [llength $list_Y1]
set l2 [llength $list_Y2]
set degree 2
set k1 0
set i1 0
set k2 0
set i2 0
while {$i1< $l1} {
        set j1 [expr $i1 + [expr $N4 - 1 ] ]
        set list_X1_bis [lrange $list_X1 $i1 $j1]
        set list_Y1_bis [lrange $list_Y1 $i1 $j1]
        set coeffs1 [generate.equation $list_X1_bis $list_Y1_bis]
        set array_equation($k1,2) $coeffs1
        set j2 [expr $i2 + $N5 ]
        set list_X2_bis [lrange $list_X2 $i2 $j2]
        set list_Y2_bis [lrange $list_Y2 $i2 $j2]
        set coeffs2 [generate.equation $list_X2_bis $list_Y2_bis]
        set array_equation($k2,3) $coeffs2
        set i2 [expr $i2 + [expr $N4 - 1 ] ]
        set k2 [expr $k2 + 1 ]
        set i1 [expr $i1 + $N4 ]
        set k1 [expr $k1 + 1 ]
        }

# The following procedure gives the absolute value of a any number.
proc v_abs {num} {
    if {$num > 0} {
        return $num
    }
    set num [expr $num * (-1)]
    return $num
}

######## VDDCurrent Results ########

# Creating an output txt file that would contain all the results of the VDD Current consumption.
set tclresult [open "tcl_result_for_VDDCurrent.txt" w+]
puts $tclresult "VDD(V) \t VDDPump(V) \t Tcl_Value(A) \t Simulation_Value(A) \t Error(%) "
# Creating another output file that contains only the coefficients
set tcl_coeffs_result [open "tcl_coeffs_result_for_VDDCurrent.txt" w+]

#Creating an array named array_comparision that would contain the simulation data values for the supply currents
for {set j 0} {$j<$N1} {incr j} {
        for {set i 0} {$i<[expr $N3 +  2]} {incr i} {
                set array_comparision($j,$i) 0
                if  {$i==0 || $i==1} {
                        set h [expr $i + 1 ]
                        set array_comparision($j,$i) $array($j,$h) }
                }
```

```
#Computing the supply currents values using the equations found by tcl and then putting them in the output .txt file. Also putting
#Computing the error between tcl and simulation value for each data and adding it to the .txt file as well.

set k 3
set max_vdd 0
for {set i 0} {$i<$N3} {incr i} {
        set h [expr $i +  2]
        puts $tcl_coeffs_result "[lindex $array_equation($i,1) 2] [lindex $array_equation($i,1) 1]
         [lindex $array_equation($i,1) 0] \n"
        puts $tclresult "\n VxrCtl = [lindex $array_equation($i,0)] \t VDDCurrent = Vdd2([lindex $array_equation($i,1) 2]*
        VDDPump2+[lindex $array_equation($i,1) 1]*VDDPump+[lindex $array_equation($i,1) 0]) \n"
        for {set j 0} {$j<$N1} {incr j} {
                set factor [expr $array_comparision($j,0)*$array_comparision($j,0)]
                #puts $array_comparision($i,0)
                set a  [expr $array_comparision($j,1)*$array_comparision($j,1) * [lindex $array_equation($i,1) 2] ]
                #Need to check if it is 0 or 2
                set b  [expr $array_comparision($j,1) * [lindex $array_equation($i,1) 1] ]
                set c  [lindex $array_equation($i,1) 0]
                set theovalue  [expr $factor * [expr $a + [expr $b + $c ] ] ]
                set array_comparision($j,$h) $theovalue
                set e [expr $array_comparision($j,$h) - $array($j,$k) ]
                set Err  [v_abs [expr $e / $array($j,$k) ]]
                if  {$max_vdd < $Err} {
                        set max_vdd $Err }
                puts $tclresult "$array_comparision($j,0) \t $array_comparision($j,1) \t $array_comparision($j,$h)
                   $array($j,$k) \t $Err "
        }
        set k [expr $k + 3 ]
}
#This adds the value of the maximum error to the .txt file.
puts $tclresult " \n The maximun error is equal to $max_vdd \n "



######## VDDPUMPCurrent Results ########


# Creating an output txt file that would contain all the results of the VDDPUMP Current consumption.

set tclresultVDDPump [open "tcl_result_for_VDDPumpCurrent.txt" w+]
puts $tclresultVDDPump "VDD(V) \t VDDPump(V) \t Tcl_Value(A) \t Simulation_Value(A) \t Error(%) "
# Creating another output file that contains only the coefficients
set tcl_coeffs_resultVDDPUMP [open "tcl_coeffs_result_for_VDDPumpCurrent.txt" w+]

#Creating an array named array_comparision that would contain the simulation data values for the supply currents

for {set j 0} {$j<$N1} {incr j} {
        for {set i 0} {$i<[expr $N3 +  2]} {incr i} {
                set array_comparision_VDDPump($j,$i) 0
                if  {$i==0 || $i==1} {
                        set h [expr $i + 1 ]
                        set array_comparision_VDDPump($j,$i) $array($j,$h) }
                }
```

```
#Computing the supply currents values using the equations found by tcl and then putting them in the output .txt file. Also putting
#Computing the error between tcl and simulation value for each data and adding it to the .txt file as well.

set k 3
set max_vddpump 0
for {set i 0} {$i<$N3} {incr i} {
        set h [expr $i +  2]
        puts $tcl_coeffs_resultVDDPUMP "[lindex $array_equation($i,2) 2] [lindex $array_equation($i,2) 1]
        [lindex $array_equation($i,2) 0] [lindex $array_equation($i,3) 2] [lindex $array_equation($i,3) 1]
        [lindex $array_equation($i,3) 0]"
        puts $tclresultVDDPump "\n VxrCtl = [lindex $array_equation($i,0)] \t VDDPumpCurrent = ([lindex $array_equation($i,2) 2]*
        VDDPump2 + [lindex $array_equation($i,2) 1]*VDDPump + [lindex $array_equation($i,2) 0])
+ ([lindex $array_equation($i,3) 2] * Vdd2 + [lindex $array_equation($i,3) 1] * Vdd + [lindex $array_equation($i,3) 0]) \n "
        for {set j 0} {$j<$N1} {incr j} {
                set a  [expr $array_comparision_VDDPump($j,1) * $array_comparision_VDDPump($j,1) *
                [lindex $array_equation($i,2) 2] ]
                set b  [expr $array_comparision_VDDPump($j,1) * [lindex $array_equation($i,2) 1] ]
                set c  [lindex $array_equation($i,2) 0]
                set eq1  [expr $a + [expr $b + $c ] ]
                set a1  [expr $array_comparision_VDDPump($j,0)*$array_comparision_VDDPump($j,0) * [lindex $array_equation($i,3) 2]
                set b1  [expr $array_comparision_VDDPump($j,0) * [lindex $array_equation($i,3) 1] ]
                set c1  [lindex $array_equation($i,3) 0]
                set eq2  [expr $a1 + [expr $b1 + $c1 ] ]
                set theovalue [expr $eq1 + $eq2 ]
                set array_comparision_VDDPump($j,$h) $theovalue
                set e [expr $array_comparision_VDDPump($j,$h) - $array($j,$k) ]
                set Err [v_abs [expr $e / $array($j,$k) ]]
                if  {$max_vddpump < $Err} {
                        set max_vddpump $Err }
                puts $tclresultVDDPump "$array_comparision_VDDPump($j,0) \t $array_comparision_VDDPump($j,1) \t
                $array_comparision_VDDPump($j,$h)          $array($j,$k) \t $Err "
        }
        set k [expr $k + 3 ]
}

#This adds the value of the maximum error to the .txt file.
puts $tclresultVDDPump " \n The maximun error is equal to $max_vddpump \n "
```

8.3 VerilogAMS code

```
`timescale 1ns/1ns
`include "disciplines.vams"
`include "constants.vams"

module ESTM15_VXR_TOP (
// intput
ADD_STAGE, VXRCTL, PROG,
// unused
CKPUMPSBY, HIBERNATE_STATE_MV, ENABLESBYPUMP, CONFIPROG,
// powers
CASCP4VYP, GND, VDD, VBGAP, VIREF, VDD_PUMP, VPOL, VCC, VYP, VYP7V,
// inputs
BADPOWER, ENABLEVXR, SLEEPINT, VPSEN, TLDISABLEVXRPUMPREG, TLVXRPD6VRANGE,
// clock
CKPUMP,
// output
VXR2, BLEMU, VXRST1, VXR1, SLEEPHV, SLEEPN, YMSEMUOUT,
);
input ENABLESBYPUMP, PROG, CKPUMPSBY;
input VIREF, VPOL, GND, VDD, VBGAP, VDD_PUMP, VCC, CASCP4VYP, VYP, VYP7V;
input BADPOWER, CKPUMP, ENABLEVXR, HIBERNATE_STATE_MV, SLEEPINT, TLDISABLEVXRPUMPREG, VPSEN, TLVXRPD6VRANGE;
input [4:0] VXRCTL;
input [3:0] CONFIPROG;
input ADD_STAGE;
output BLEMU, SLEEPHV, SLEEPN, VXRST1, VXR1, VXR2, YMSEMUOUT;

real VDD_PUMP_r, VOUT, pump_dv, VDDCurrent;
logic clkpump_ok; //flag which is =1 if clock is between min_freq & max freq
integer refresh, refresh_vdd_pump, vxrctl_add;
logic functional_ok;

integer VBgap_ok;
integer VIref_ok;
integer Vpol_ok;
integer VxrPump_ok;
integer GndRef_ok;
integer vdd_ok;
integer gnd_ok;
integer pump_ok;
integer pump_on;

integer analog_ok;

electrical VIREF, VPOL, GND, VDD, VBGAP, VDD_PUMP, VCC, CASCP4VYP, VYP, VYP7V, HIBERNATE_STATE_MV, BLEMU, SLEEPHV,
SLEEPN, VXRST1, VXR1, VXR2, YMSEMUOUT, VXR2_ideal;
real BLEMU_r, SLEEPHV_r, SLEEPN_r, VXRST1_r, VXR1_r, VXR2_r, YMSEMUOUT_r, VCC_r, VDD_r, VDD_PUMP_CALC, VCCmVDDP,
I_VDDPump, VDDPump_out, I_VXR2, I_VDD;

parameter real VBgap_min = 0.53; //low limit of BandGap voltage
parameter real VBgap_max = 0.74; //high limit of Bandgap voltage

parameter real VIref_min = 0.5; //low limit of VIref voltage
```

```
parameter real VIref_min = 0.5; //low limit of VIref voltage
parameter real VIref_max = 0.8; //high limit of VIref voltage

parameter real Vpol_min = 1.4; //low limit of Vpol cascode voltage
parameter real Vpol_max = 2.5; //high limit of Vpol cascode voltage

parameter real vdd_min = 0.85; //low limit of digital vdd voltage
parameter real vdd_max = 1.45; //high limit of digital vdd voltage

parameter real gnd_min = -0.1; //low limit of gnd voltage
parameter real gnd_max = 0.1; //high limit of gnd voltage

//parameter tcl_result_for_VDDCurrent = "~/bit_sequence.txt";
//integer fp;

real vxr2_modify[0:15] ='{3.91, 4.01, 4.10, 4.19, 4.29, 4.39, 4.49, 4.58, 4.68, 4.78, 4.88, 4.97, 5.07, 5.17, 5.27,
5.36};
real vxr2_6V[0:15] ='{5.21, 5.34, 5.46, 5.59, 5.72, 5.85, 5.98, 6.11, 6.24, 6.37, 6.50, 6.63, 6.76, 6.89, 7.02, 7.15};
real vxr2_read[0:15] ='{3.26, 3.34, 3.41, 3.49, 3.58, 3.66, 3.74, 3.82, 3.90, 3.98, 4.06, 4.14, 4.23, 4.31, 4.39,
4.47};
real vxr1_read[0:15] ='{2.28, 2.34, 2.39, 2.45, 2.50, 2.56, 2.62, 2.67, 2.73, 2.79, 2.84, 2.90, 2.96, 3.01, 3.07,
3.13};
real coeff_final_1;
real coeff_final_2;
real coeff_final_3;
real coeff_final_4;
real coeff_final_5;
real coeff_final_6;
integer k;
real coeff_final_VDD_1;
real coeff_final_VDD_2;
real coeff_final_VDD_3;

//clock_checker250 clkP\\mpchk (CKPUMP, 10e6, 280e6, clkpump_ok);
assign functional_ok = (ENABLEVXR==1'b1 && SLEEPINT==1'b0 && BADPOWER==1'b0 && TLDISABLEVXRPUMPREG==1'b0 &&
ENABLESBYPUMP==1'b0);
always@(VXRCTL) begin
        vxrctl_add = 8*VXRCTL[3] + 4*VXRCTL[2] + 2*VXRCTL[1] + 1*VXRCTL[0];
        k = vxrctl_add;
        $display("here j", k );
end
always@(TLVXRPD6VRANGE,VXRCTL) begin
        vxrctl_add = VXRCTL[3:0];
        VOUT = (TLVXRPD6VRANGE)? vxr2_6V[vxrctl_add] :(VXRCTL[4])? vxr2_modify[vxrctl_add] : vxr2_read[vxrctl_add];
end

real coeffs1[0:31] ;
real coeffs2[0:31] ;
real coeffs3[0:31] ;
real coeffs4[0:31] ;
real coeffs5[0:31] ;
real coeffs6[0:31] ;
real coeff1_VDD[0:31];
real coeff2_VDD[0:31];
```

```
real coeff3_VDD[0:31];
integer file_results_VDDPUMP;
integer file_results_VDD;
integer scan_results_VDDPUMP;
integer scan_results_VDD;
integer scan_string_VDDPUMP;
integer scan_string_VDD;
integer i = 0 ;
integer j = 0 ;
reg [4095:0] data_VDDPUMP;
reg [4095:0] data_VDD;

initial begin
     file_results_VDDPUMP = $fopen("tcl_coeffs_result_for_VDDPumpCurrent_3stages.txt", "r");
     file_results_VDD = $fopen("tcl_coeffs_result_for_VDDCurrent_3stages.txt", "r");
     if (file_results_VDDPUMP == 0) begin
       $display("Results VDDPUMP is empty");
       $finish;
     end
     if (file_results_VDD == 0) begin
       $display("Results VDD is empty");
       $finish;
     end
     while (!$feof(file_results_VDDPUMP)) begin
        scan_results_VDDPUMP = $fgets(data_VDDPUMP, file_results_VDDPUMP);
        scan_string_VDDPUMP = $sscanf(data_VDDPUMP, "%g %g %g %g %g %g /n", coeffs1[i], coeffs2[i], coeffs3[i],
coeffs4[i], coeffs5[i], coeffs6[i]);
        $display("scan_results_VDDPUMP: %s  data_VDDPUMP: %s", scan_results_VDDPUMP, data_VDDPUMP);
        $display("coeffs are displayed heree");
        i = i + 1;
     end
     $fclose(file_results_VDDPUMP);
   //  $finish;
     while (!$feof(file_results_VDD)) begin
        scan_results_VDD = $fgets(data_VDD, file_results_VDD);
        scan_string_VDD = $sscanf(data_VDD, "%g %g %g /n", coeff1_VDD[j], coeff2_VDD[j], coeff3_VDD[j]);
        $display("scan_results_VDD: %s  data_VDD: %s", scan_results_VDD, data_VDD);
        $display("coeffs VDD are displayed heree");
        j = j + 1;
     end
     $fclose(file_results_VDD);
end

analog begin
        I_VXR2 = I(VXR2,VXR2_ideal);

        VBgap_ok = (((V(VBGAP) >= VBgap_min) && (V(VBGAP) <= VBgap_max))? 1 : 0);
        VIref_ok = (((V(VIREF) >= VIref_min) && (V(VIREF) <= VIref_max))? 1 : 0);
        Vpol_ok = (((V(VPOL) >= Vpol_min) && (V(VPOL) <= Vpol_max))? 1 : 0);
        vdd_ok = (((V(VDD) >= vdd_min) && (V(VDD) <= vdd_max))? 1 : 0);
        gnd_ok = (((V(GND) >= gnd_min) && (V(GND) <= gnd_max))? 1 : 0);
        analog_ok = (((VBgap_ok + VIref_ok + Vpol_ok + vdd_ok + gnd_ok) == 5)? 1 : 0);
        pump_on = (((1*functional_ok + analog_ok) == 2)? 1 : 0);

```

```
153         VDD_PUMP_r = V(VDD_PUMP);
154         VCC_r = V(VCC);
155         VDD_r = V(VDD);
156         YMSEMUOUT_r = 0;
157
158         //handling the full functional case of the pump
159         if (pump_on==1) begin
160                 //I_VDDPump = - 1e-6;
161                 //I_VDD = - 1e-6;
162
163                 VDD_PUMP_CALC = 3*VDD_PUMP_r-0.3;
164
165                 //different trim table depending if the pump is in modify mode or not
166                 BLEMU_r = (ADD_STAGE===1'b0 && VOUT>VDD_PUMP_CALC) ? VDD_PUMP_CALC : VOUT ;
167
168 //              VXR2_r = (VPSEN===1'b0) ? BLEMU_r : VCC_r ;
169                 if (VPSEN===1'b0) begin
170                         VXR2_r = BLEMU_r;
171                 end else begin
172                         VXR2_r = VCC_r ;
173                 end
174
175                 pump_dv = VOUT - VDD_PUMP_r;
176
177
178                 I_VDDPump = -((coeff_final_1*VDD_PUMP_r*VDD_PUMP_r-coeff_final_2*VDD_PUMP_r+
    coeff_final_3+coeff_final_4*VDD_r*VDD_r-coeff_final_5*VDD_r+coeff_final_6)*(I_VXR2/0.00008));
179                 I_VDD = - ((VDD_r*VDD_r*
    (coeff_final_VDD_1*VDD_PUMP_r*VDD_PUMP_r-coeff_final_VDD_2*VDD_PUMP_r+coeff_final_VDD_3))*(I_VXR2/0.00008));
180
181                 //I_VDDPump = -((0.0007318239492407497*VDD_PUMP_r*VDD_PUMP_r-0.002979274068177162 3*VDD_PUMP_r+
    0.003344868608225653+0.0005160854166653283*VDD_r*VDD_r-0.0010466059583303699*VDD_r+0.000523396499998384 2)*(I_VXR2/0.000
    08));
182                 //I_VDD = - ((VDD_r*VDD_r*
    (0.0013530417129173668*VDD_PUMP_r*VDD_PUMP_r-0.005515920198221083*VDD_PUMP_r+0.005695330221616578))*(I_VXR2/0.00008));
183
184                 //taking different values for VXR1 and VXRST1 depending on read/modify mode, and lowcons (2 stages) /
    highperf (3 stages)
185                 //modify mode activated by VXRCTL[4] bit, readmode if '0'
186                 //VXR1 & VXRST1 are juste intermediate voltages, not regulated
187                 if ((VXRCTL[4])||(TLVXRPD6VRANGE)) begin
188                         //high perf mode triggered by ADD_STAGE (VXRPD3STAGEON)
189                         if (ADD_STAGE === 1'b1) begin
190                                 VXR1_r= VDD_PUMP_r + pump_dv*0.67 ;
191                                 VXRST1_r= VDD_PUMP_r + pump_dv*0.45 ;
192                         //in low cons mode, VXRST1 = VXR1
193                         end else begin
194                                 VXR1_r= VDD_PUMP_r + pump_dv*0.52 ;
195                                 VXRST1_r= VXR1_r;
196                         end
197                 // in read mode, VXR1 is regulated (same value for low cons & high perf), while VXRST1 is not
198                 end else begin
199                         VXR1_r = vxr1_read[vxrctl_add];
200                         VXRST1_r = (ADD_STAGE===1'b1)? (VDD_PUMP_r + 0.66*(VXR1_r - VDD_PUMP_r)) : VXR1_r;
```

```
201             end
202         //handling sleep mode
203         end else begin
204                 I_VDDPump = 0;
205                 I_VDD = 0;
206                 if ((BADPOWER === 1'b1)||(Vpol_ok == 0)) begin
207                         VXR2_r = 0;
208                         BLEMU_r = 0;
209                         VXR1_r = 0;
210                         VXRST1_r = 0;
211                 end else if (SLEEPINT===1'b1) begin
212                         VCCmVDDP = VCC_r-VDD_PUMP_r;
213                         VXR2_r = VCC_r;
214                         BLEMU_r = VCC_r-1;
215                         VXRST1_r = ((ADD_STAGE===1'b1)? VDD_PUMP_r + 0.45*VCCmVDDP : VDD_PUMP_r + 0.52*VCCmVDDP);
216                         VXR1_r = ((ADD_STAGE===1'b1)? VDD_PUMP_r + 0.67*VCCmVDDP : VDD_PUMP_r + 0.52*VCCmVDDP);
217                 end else begin
218                         VXR2_r = VDD_PUMP_r;
219                         BLEMU_r = VDD_PUMP_r;
220                         VXR1_r = VDD_PUMP_r;
221                         VXRST1_r = VDD_PUMP_r;
222                 end
223         end
224
225         SLEEPHV_r = (SLEEPINT===1'b1) ? VCC_r : 0;
226         SLEEPN_r = (SLEEPINT===1'b0) ? VDD_r : 0;
227         //assign YMSEMUOUT = 0 ;
228
229         V(BLEMU) <+ transition(BLEMU_r, 2e-7, 2e-7);
230         V(SLEEPHV) <+ transition(SLEEPHV_r, 1e-9, 1e-9);
231         V(SLEEPN) <+ transition(SLEEPN_r, 1e-9, 1e-9);
232         V(VXRST1) <+ transition(VXRST1_r, 2e-7, 2e-7);
233         V(VXR1) <+ transition(VXR1_r, 2e-7, 2e-7);
234         V(VXR2_ideal) <+ transition(VXR2_r, 2e-7, 2e-7);
235         V(YMSEMUOUT) <+ transition(YMSEMUOUT_r, 2e-7, 2e-7);
236         I(VDD_PUMP) <+ transition(I_VDDPump,2e-7, 2e-7);
237         I(VDD) <+ transition(I_VDD,2e-7, 2e-7);
238         I(VXR2,VXR2_ideal):V(VXR2,VXR2_ideal) == 0.0;
239 end
240
241 endmodule
```