Master thesis project 2021-2022

Grenoble INP-PHELMA

Nanotech

Implementation and verification of an in-SRAM computing subarray in 65nm CMOS technology

---

Presented by : **Clément CHONE**
clement.chone@epfl.ch

Examining Committee :

PHELMA Tutor :
Prof. Lorena ANGHEL, lorena.anghel@phelma.grenoble-inp.fr

EPFL supervision team :
Prof. David ATIENZA, david.atienza@epfl.ch

Dr. Alexandre LEVISSE, alexandre.levisse@epfl.ch

Dr. Davide SCHIAVONE, davide.schiavone@epfl.ch

Mr. Marco RIOS, marco.rios@epfl.ch

Done at : **Embedded System Laboratory (ESL)**
Address : ELG building (EPFL), Route cantonale 1015 ECUBLENS (SUISSE)
Confidentiality : **No**

21 February 2022 — 13 August 2022

Acknowledgements

I would like to express my sincere gratitude to my research supervisors Dr. Alexandre LEVISSE, Dr. Davide SCHIAVONE and Marco RIOS for their help and advice during this internship as well as all the knowledge they provide me about Analog and Digital circuit design.

Besides my supervisors, I would like to thank Prof. David ATIENZA, director of the Embedded System Laboratory, who welcomed me in his laboratory for this internship.

Finally, I would like to thank all people who contributed to the success of this study in any way.

Abstract

With the growing interest of applications relying on Artificial Intelligence, Convolutional Neural Network gained in popularity because of their high potential in terms of performance, energy efficiency and security. However, because of their high energy consumption, their integration inside embedded systems remains difficult and, by consequences, many efforts are made to built low power accelerators. In that purpose, In-Memory Computing promises great improvements since it allows the acceleration of the processing speed by reducing the data transfer between the memory and CPU in addition to remarkable energy savings. The aim of this project is to implement and validate an improved version of a 2kB memory subarray for building an in-SRAM computing memory based on the EPFL BLADE architecture Bitline Accelerator for Device on the Edge, that will be implemented in an upcoming tapeout in 65nm. Throughout this study, we will first explore a new architecture based on the used of Embedded shift in order to accelerate up to 4 times the multiplication step. In addition, a power gating strategy is proposed to reduce the memory power consumption of idle blocks. Results show that the proposed implementations will enable us to save up to 30% of energy. Finally, an Analog-Mixed-Signal (AMS) design flow will be proposed so as to assure the proper integration of the full-custom memory with its digital, standard-cell based controller.

Résumé

Avec l'intérêt croissant des applications reposant sur l'utilisation de l'intelligence artificielle, les réseaux de neuronnes convolutifs ont gagné en popularité de par leur fort potentiel en termes de performances, d'efficacité énergétique et de sécurité. Cependant, leur intégration au coeur de systems embarqués reste difficile à cause de leur forte consommation énergétique et, par conséquence, de nombreux efforts sont réalisés afin de construire des accélérateurs basse consommation. Dans cet objectif, l'In-Memory Computing promet un progrès significatif puisqu'il permet une amélioration de la vitesse de calcul grâce à la réduction des movements de données entre le CPU et la mémoire ainsi que des gains énergétiques considérables. Le but de ce projet est donc d'implémenter et de valider une version améliorée d'un sous-réseau mémoire de 2 Ko afin de construire une mémoire SRAM calculatoire basée sur l'architecture BLADE (Bitline Accelerator for Device on the Edge) créée par l'EPFL et qui sera intégrée dans une prochaine puce. Au cours de cette étude, nous allons exploré en premier lieu une nouvelle architecture basée sur l'utilisation d'Embedded Shift dans le but d'accélérer jusqu'à 4 fois la multiplication. De plus, une stratégie de power-gating sera proposée afin de réduire la consommation d'énergie des parties inactives de la mémoire. Les résultats obtenus montreront que les implémentations proposées nous permettra d'économiser jusqu'à 30% d'énergie. Enfin, un flux de conception Analog Mixed-signal sera proposé afin d'assurer l'intégration de la mémoire avec son contrôleur digital.

Sommario

Con il crescente interesse per le applicazioni basate sull'intelligenza artificiale, le reti neurali convoluzionali hanno guadagnato popolarità grazie al loro elevato potenziale in termini di prestazioni, efficienza energetica e sicurezza. Tuttavia, a causa del loro consumo energetico, la loro integrazione in sistemi embedded rimane difficile e, di conseguenza, vengono fatti molti sforzi per costruire acceleratori a bassa potenza. A tal fine, l'In-Memory Computing promette grandi miglioramenti grazie all'accelerazione della fase di elaborazione, riducendo il trasferimento di dati tra memoria e processore, che risulta in notevoli risparmi energetici. L'obiettivo di questo progetto è implementare e convalidare una versione migliorata di un subarray di memoria da 2kB per la creazione di una memoria di calcolo in-SRAM basata sull'architettura BLADE, Bitline Accelerator for Device on the Edge, che sarà implementata in un prossimo tapeout. Durante questo studio, prima una nuova architettura basata sull'uso di Embedded shift per accelerare fino a 4 volte la fase di moltiplicazione verrà esplorata. Quindi, alcune ottimizzazioni nella funzionalità del subarray in modo da ridurne il consumo energetico oltre allo sviluppo di una strategia di power gating per ridurre la potenza consumate da elementi silenti verranno proposte ed implementate. I risultati mostrano che le implementazioni proposte consentono di risparmiare fino al 30% di energia. Infine, verrà proposto un flusso di progettazione Analog-Mixed-Signal (AMS) in modo da assicurare la corretta integrazione della memoria full-custom con il suo controllore basato su standard-cell nel prossimo tapeout.

Contents

List of Figures

List of Tables

1 Introduction

With the intensive development of applications relying on Artificial Intelligence such as self-driving cars, image recognition and surveillance systems, a growing interest has emerged for Convolutional Neural Network (CNN) over the past few years since they provide many advantages in terms of performance, energy efficiency and security [1]. However, due to their high energy consumption and large resources requirement, their integration in edge devices for embedded systems remains the main issue. As a solution, engineers developed new architectures based on In-Memory Computing (IMC) to reduce data movement between the CPU and the memory, which is generally the main source of energy consumption for such applications. Thus, intensive data workload can be achieved directly on IoT edge devices, improving the security and the energy efficiency of the system by suppressing the tranfer of a huge quantity of data towards other networks (Fog, Cloud...) [2] [3].

Within this context, a new architecture using In-Memory Computing called BLADE is being developed by the ESL laboratory in EPFL since 2019. Relying on the use of Local Groups (LG) and Bitline computing, it boosts the performances of a traditional memory computing architecture in terms of area, frequency of operation and computational complexity [4] [5]. A first version was implemented inside Rosetta [6] in 2019, a chip co-designed with the Integrated System Laboratory in ETH Zurich. Two years later, an improved version of BLADE was implemented in a second collaborative chip named Darkside [7]. Currently, the ESL Laboratory is working on a third tapeout named HEEPpocrates in which further optimizations regarding the power consumption of the array proposed in this Thesis, as well as the computing process, will end with a promising accelerator architecture for the integration of CNN in IoT edge devices.

The goal of the proposed Thesis is to improve the power and energy efficiency of a 2KB SRAM-based in-memory computing subarray implemented in TSMC 65nm based on the EPFL Blade architecture. Such macro is then instantiated 16 times and interfaces with a digital controller to build a 32KB memory block. Furthermore, an extensive, multiple-steps, and multiple EDA tools Validation flow is implemented to verify the logical and electrical connectivity, as well as the timing constraints between the digital controller and the subarrays. The flow relies on multiple EDA tools and checks the functionalities at several levels of the implementations as:

- the HDL description (SystemVerilog) of the digital controller and the behavioral model of the subarrays;
- the synthesized digital controller (Verilog model of the standard cells) and the behavioral model of the subarrays;
- the synthesized digital controller (Verilog model of the standard cells) and the extracted-from-layout spice model of the subarrays;
- the spice model digital controller (spice model of the standard cells) and the extracted-from-layout spice model of the subarrays.

The aforementioned steps were performed with both Cadence and Synopsys EDA tools to further increase the validation coverage.
After a brief introduction around the BLADE architecture in Section 2, Section 3 details the exploration and integration at schematic level of new hardware implementations imagined by our research group in order to improve the computation efficiency of BLADE in

the scope of building a powerful CNN accelerator. Then, Section 4 explains the different methods investigated and implemented for reducing the energy consumption of the subarray. Because of some issues related to the testing step of Darkside which came back from fabrication at that time of the project, we decided to suspend temporarily this step and concentrate our efforts on the verification of Darkside. Thus, Section 5 details the implementation of an Analog Mixed-Signal flow of verification in order to test the interface linking the analog and digital world in Darkside.

2 Background

2.1 In-Memory Computing for edge devices

With the continue increased amount of data that have to be processed in a shorter time, the data access time of memories remains the main barrier for processing data quickly and efficiently and impedes the use of CPUs to their 100% capacity. In order to assess this issue commonly named CPU-memory bottleneck or "Von Neumann bottleneck", scientist have started to develop architectures enabling them to process the data directly inside the memory [8]. For Edge devices and time critical applications, this idea promises breathtaking performances since it enables a huge reduction in terms of data movement resulting in a faster and more power efficient data treatment. Indeed, Edge devices often lack of resources available for computation and therefore require the use of external servers for processing data creating latency and power hungry applications. The insertion of computation capabilities closer to the memory addresses the main constraints of edge devices (latency and power consumption) and may open the path toward bigger data workload such as AI-related computations[9].

Thus, many data-centric architectures such as In-Memory-Computing (IMC) and Near-Memory Computing (NMC) have been developed so as to bring the computation unit closer to the memory array in order to increase the performances of modern architectures limited by memory access, power and delay (also called "memory wall") [10]. The data computation in such architectures is usually performed in the periphery of the memory and the results are then written back directly inside the array from there without the use of the processor. This feature makes them usually less dense than classical Read/Write memories but offers a great opportunity to run memory intensive applications at the edge.

2.2 BLADE : a smart In-SRAM Computing architecture

BLADE, standing for BitLine Accelerator for Device on the Edge, is a 6T in-SRAM computing architecture developed for low-power devices like embedded systems which are often powered by batteries. Relying on IMC, it performs some bitwise and arithmetic operations without the use of an external Arithmetical Logic Unit (ALU) in order to reduce data movement between the memory and the processor. BLADE is a 32kB memory composed of 16 subarrays. Each subarray is composed of 4 different blocks having different duties as shown by Figure 1 :

- Array : composed of bitcells storing the data and organized in words of 32 bits in addition to some logic enabling their access (precharge, read port ...);
- Decoder (DEC) : allows to decode the type of operation to execute (Read, Write, IMC) and the address;
- Controller (CTRL) : handles the control signals and their activation during a clock cycle for the proper operation of the memory;
- IO logic : allows to compute the result of the operation and output it towards registers. The result can be then written back in the memory at the next clock cycle.

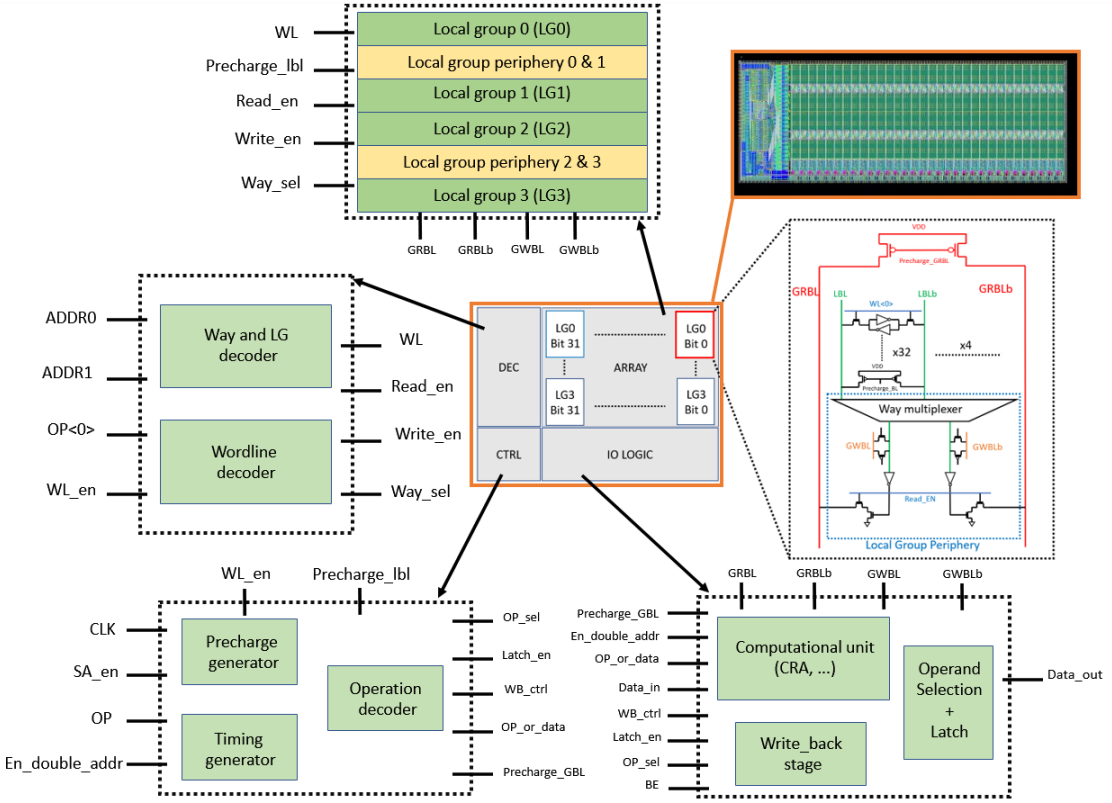


Figure 1: Organisation of the different blocks inside a 2kB subarray

To perform arithmetic computations, the BLADE architecture relies on Bitline Computing which consists in activating two wordlines at the same time. This results in the discharge of two pairs of local bitlines which then activate the discharge of the Global Read Bitlines (GRBLs) thanks to the Local Read Port. The GRBLs being bitwise shared by all the bitcells, the data is merged on these latter so as to compute a bitwise NOR and AND operation between the values read from the two activated bitcells as explained by Figure 2. Based on these two signals, additional logic has been added underneath the bitcells array in order to perform a XOR bitwise operation as well as more complex operations like Shift, ADD, ADD+SHIFT, subtraction or greater/less than. The XOR operation is done with a NOR gate between the 2 GRBLs whereas a carry ripple adder (CRA) is used for the addition and shifts. The same hardware can be also used to perform the subtraction since this latter can be achieved by the addition of the first operand with the 2's complement version of the second operand. The former is obtained by taking the values on the LBLb to which we add the binary value 1 thanks to the carry chain (carry_in equal to 1). Finally, Greater/less than operation can be achieved by subtracting two operands and checking the value of the MSB.

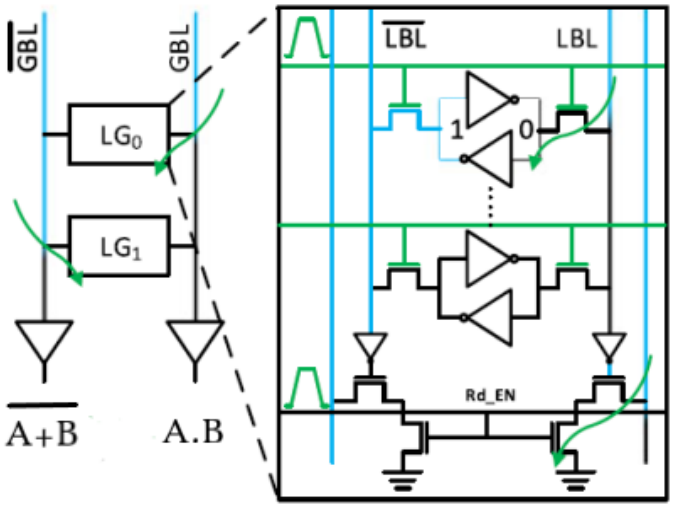


Figure 2: Architecture assuring the bitwise NOR and AND operation between the activated bitcells [5]

The specificity of BLADE compared to other in-SRAM Computing memory architectures relies in the use of Local Groups and Local Bitlines. These characteristics enable to both limit the risk of data corruption due to the simultaneous activation of two wordlines and to keep a good density of bitcells inside the array. Indeed, if two activated bitcells belong to the same bitlines, it may happen that the content of one bitcell flips the other one because of process variation during fabrication or bitcell degradation. However, another study conducted by K.C Akyel and al. shows the possibility of in-SRAM computing without the use of additional bitlines to prevent data corruption. It has been done through the addition of two read ports in each bitcell. This feature enables to secure the data stored inside the bitcells but they have to pay the price of some area overhead [11]. Nevertheless, it has to be mentioned that the use of local bitlines and local groups inside BLADE induce a constraint in data placement since data have to belong to different local groups in order to perform operations.

2.3 PULP, Rosetta and Darkside

In 2019, BLADE was first integrated inside Rosetta, a low-power 65nm chip designed in common by the Integrated System Laboratory (ISS ETH Zurich), the Telecommunication Systems laboratory (EPFL) and the Embedded System Laboratory (EPFL) [6].
Rosetta is based on the single core platform Pulpissimo which is able to perform several RISC-V ISA extensions such as Single Instruction Multiple Data (SIMD) operations, hardware loops, MAC and fixed point operations thanks to its vector processing core RI5CY [12]. The chip is constituted of 512kB of L2 memory in order to process traditional computations requiring high memory capacity as well as a 64kB xSRAM. This latter is an improved version of SRAM able to perform in-memory vector boolean operations (NAND, NOR, IMP and XOR) and used for simultaneous program execution [13]. Finally, a 32kB version of BLADE was implemented inside Rosetta as a Tightly-Coupled-Data-Memory (TCDM) in order to accelerate data-intensive applications. These memories use a protocol based on a simple request-acknowledgement process. Regarding BLADE, a request signal (req) is sent from the processor to BLADE's controller and a grant signal (gnt) is set to

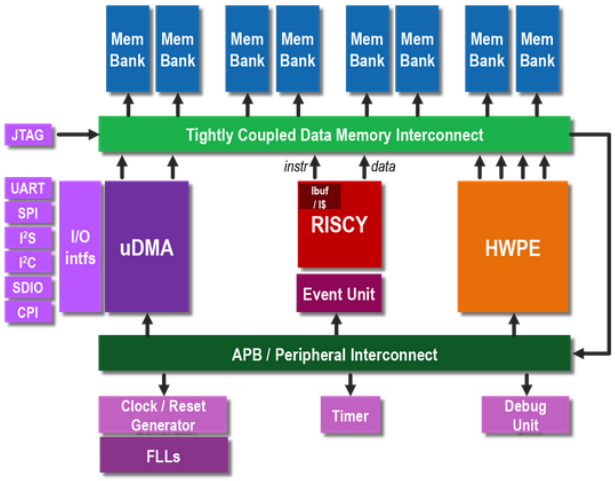


Figure 3: Pulpissimo architecture [12]

1 in the next cycle. This confirms that the request was indeed taken into consideration. Then the controller decodes the information it received from the processor and sends to the memory the required signals to perform the operation (address (add), byte enable (be), wordline enable (wen), opcode "op" ....). These later are then latched by the memory subarray in order to be properly set at the next rising edge of the clock. The operation is processed by BLADE during a certain number of clock cycles depending on the type of operation to execute given by the opcode. After the computation, the result is saved in the memory and outputted via a signal r_data. Meanwhile, a r_valid signal is asserted by the controller in order to tell the processor that the operation is done and the result is available. A timing diagram can be seen on Figure 4.

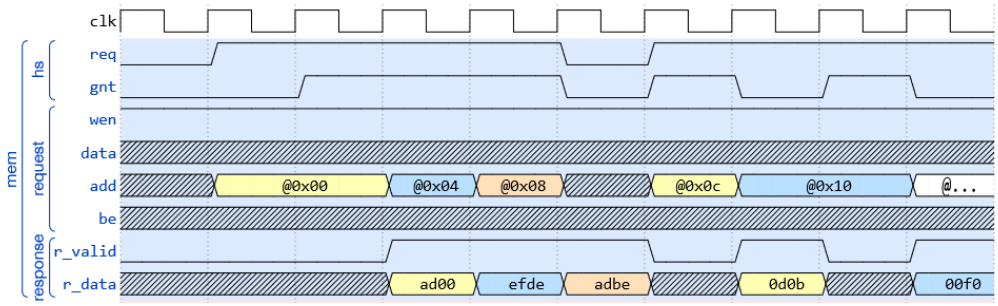


Figure 4: Timing diagram of successive read operations [14]

Unfortunately, this first version of BLADE turned out to be defective. In 2021, an improved version of BLADE based on Rosetta design was inserted inside Darkside, the second collaborative chip between ETH Zurich and EPFL. This chip integrates 8 different RISCV cores and new IPs in order to accelerate the computation time for Deep Neural Network. The integration of BLADE inside that chip will enable in-situ additions that can be chained to perform multiply-accumulate operations, achieving up to 64 1-byte (or 16 4-bytes) multiplications in parallel. [7]. The fabricated chip should come back in mid-2022 for testing.

Figure 5: Rosetta and Darkside chip [6] [7]

3 BLADE characterization and computation optimization

As mentioned in the previous section, the concept of In-Memory Computing consists in performing some arithmetic operations directly inside the memory so as to reduce data movement between the memory and the processor. Thanks to the logic embedded inside the memory periphery, we are able to perform arithmetic operations directly inside BLADE. To increase its computation efficiency in the scope of convolutional neural networks (CNN), a recent structure named "Embedded Shifts" has been designed by a PhD student of the ESL Laboratory. This optimization is mostly related to the multiplication step since it is the most used arithmetic operation in such applications. Thus, I will detail in this first section the exploration of the design space through the implementation and verification at schematic level of this new structure that i realized with the help of his inventor.

3.1 Design exploration for calculation optimization

3.1.1 Embedded shifts : a multiplication accelerator

Being a complex operation, multiplication is usually decomposed at hardware level in a succession of sub-operations like additions and shifts. This solution is compliant with heterogeneous bitwidth and usually easier to implement physically since a multiplier is a complex and big structure. The choice of sub-operation to execute is done by checking successively the bits of the multiplier starting from the most significant bit (MSB). In case the binary value '1' is read on the bit checked, an addition followed by a shift is performed at the result address, otherwise the result address is only shifted (logic "0"). An example of multiplication is shown on Figure 6.

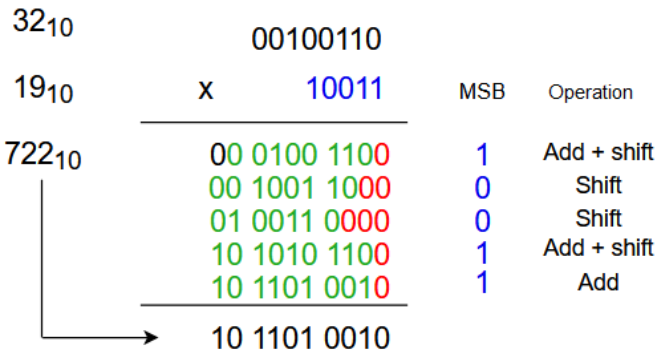


Figure 6: Example of multiplication performed by BLADE between the binary numbers $a = 00100110_2 = 38_{10}$ and $b = 10011_2 = 19_{10}$

The multiplication implemented in BLADE is done in the following way. On one side, the multiplier is stored in a 32 bits register file where its MSB is checked progressively while being shifted. On the other side, the multiplicand is stored in the array as well as the accumulator (result) which has been initialized to the value 0. Those latter need to be stored in two different LGs for the good operation of the memory.

However, the data used in CNN often contain several series of 0s and therefore the current implementation of BLADE is using many clock cycles to shift the data which is inefficient. That's why we would like to find a way to reduce the number of cycles used by BLADE to perform the multiplication. The solution found by our research group

[15] consists in checking several bits of the multiplier at the same time so as to perform several shifts in one cycle in case a pattern containing several 0s is detected. For the implementation done in this work, we decided to check the first four MSB of the register file since it has been proved from a previous study to be the best trade-off between energy, area overhead and performances [15]. A schematic of the structure controlling the multiplication steps is shown on Figure 7.

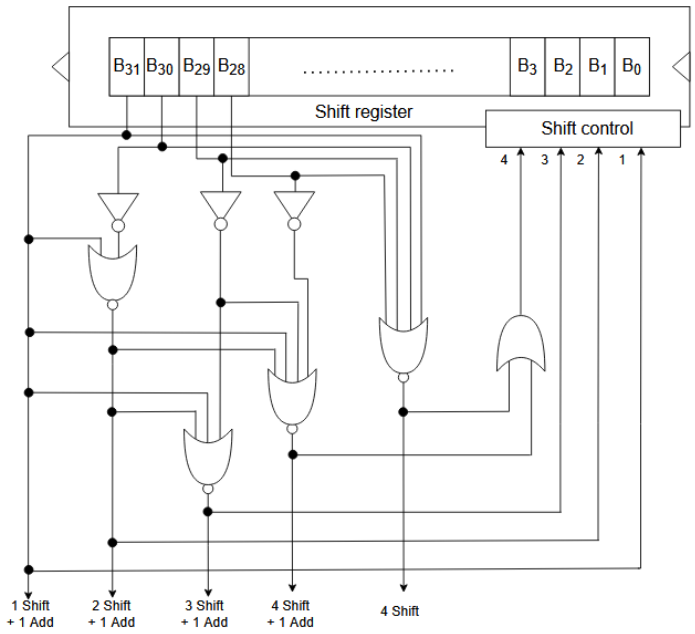


Figure 7: Schematic of the control structure for the multiplication [15]

In order to shift the data read from the LBL and LBLb, we need to include some additional logic structure named "Embedded shift" in the local group periphery (LGP) of BLADE. An embedded shift is a simple structure composed of two NMOS transistor in series as shown by Figure 8. The transistor M28 (respectively M26, M24 and M22) is activated whenever a logic 0 is read on the bitlines and allows us to discharge the Global Read BitLine through the ground if the transistor M29 (respectively M27, M25 or M23) has been activated by its respective control signal coming from the control structure (Figure 7). For instance, signal SH<1> corresponds to an assignment of "1 Shift + 1 Add", signal SH<2> to "2 Shift + 1 Add" etc...

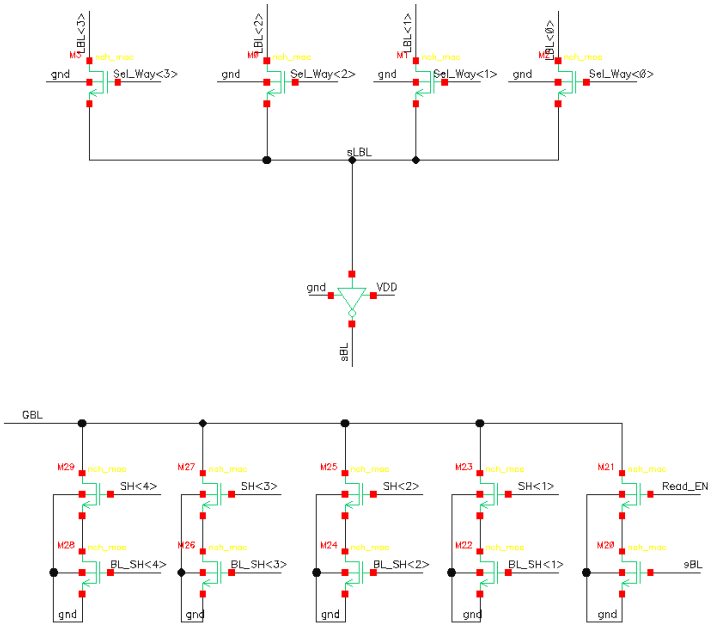


Figure 8: Schematic of 4 Embedded Shift inserted inside a LGP (only one side)

3.1.2 Embedded negation

Although the previous implementation will allow us to save many clock cycles while performing multiplications, its main issue lies in the accuracy of the result we will get. Indeed, as we are dealing with words of 32 bits, we theoretically need 64 bits to avoid some overflow issues. Therefore, with only 32 bits, we can only perform the multiplication on 16 bits if we don't want to lose accuracy. To use our memory at its full potential, we need to perform the multiplication in 2's complement and fixed point format notation as proposed by our research group [16]. This feature enables us to deal with numbers ranging from -1 to 1 which will suppress any risk of overflow while keeping a good accuracy. In that purpose, we need to perform a new operation on the data which consists in producing a negated version of those latter as shown by Figure 9.

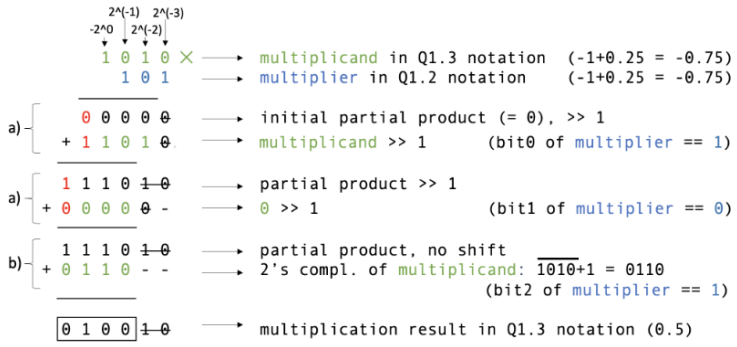


Figure 9: Example of multiplication between 2's complement number in fixed point format [16]

As during a read operation we are reading both the data and its complement, we can simply add a second multiplexer so as to chose the value either on the LBL or LBLb after

the multiplexer choosing the way to be read. A schematic view of the implementation (ES + in-situ negation) is shown on Figure 10.

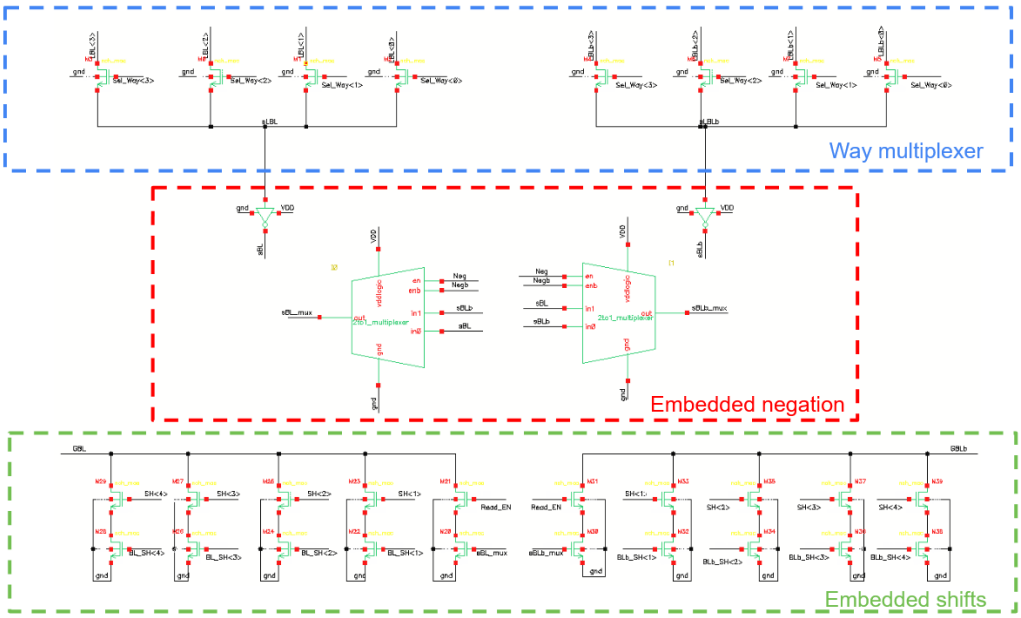


Figure 10: Schematic view of the new LGP with 4 Embedded Shift and embedded negation

Contrary to the multiplication performed in section 3.1.1, the bits have to be right shifted instead of left shifted. This change is done thanks to a proper wiring on the Embedded Shifts. Moreover, in order to keep the versatility of BLADE, we need to include other multiplexers in order to perform multiplications on 32, 16 and 8 bits. Indeed, when right shifting the data, the MSB of each word needs to be replicated for the correctness of the operation. The number of multiplexer added will depend on the bit position inside the memory.

Taking a word of 8 bits as example :
Bit 7 can be shifted 4 times at maximum and needs to be replicated for every type of shifts (1, 2, 3 or 4 times). Therefore, we will need 4 pairs of multiplexers for those 4 conditions. Bit 6 needs to be replicated if we are performing a number of shift higher than 1 (ie 2, 3 or 4). Therefore, we will need 3 pairs of multiplexers. Following this logic, Bit 5 will need 2 pairs, bit 4 only one and bits 3, 2, 1 and 0 don't need any since we don't need to replicate them. Using words of 32 bits in BLADE, this block of 8 bits has to be replicated 4 times.

For simplicity, only the block diagram corresponding to Bit 5 of the previous example is shown on Figure 11. A block diagram of a 8 bit word can be found on Figure 12.

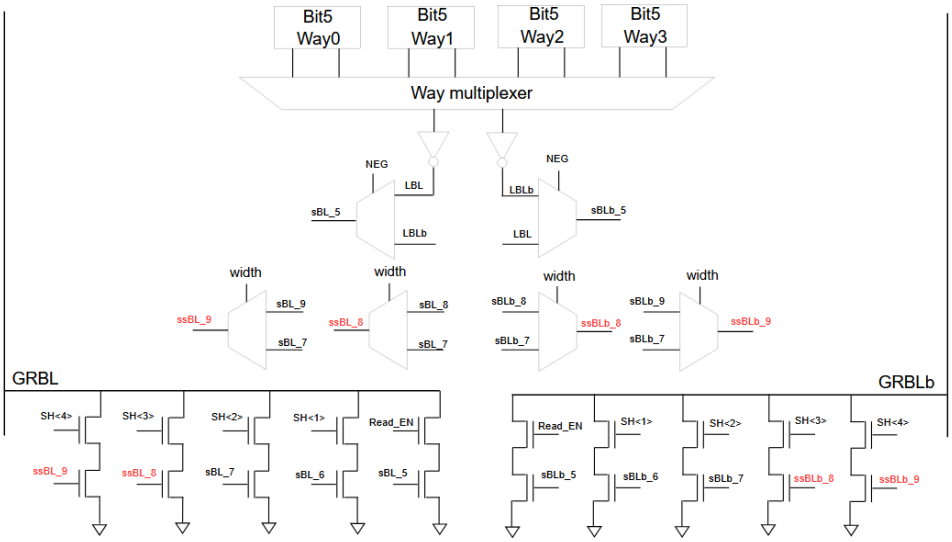


Figure 11: LGP block diagram corresponding to bit 5 with 4 Embedded shifts

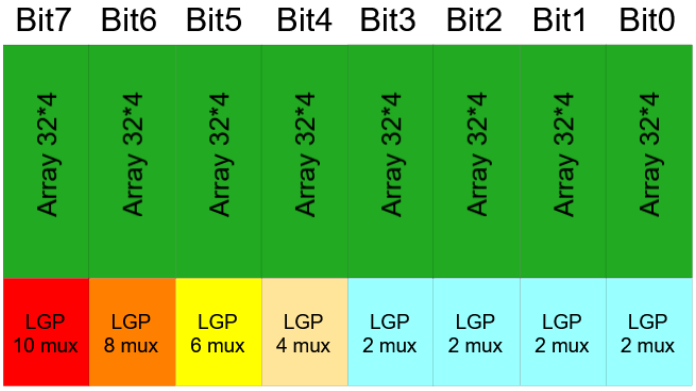


Figure 12: Block view of 8 bit words. Bit 7 needs 10 MUX since 4 pairs are needed to replicate this bit and the last two are needed for the embedded negation.

3.1.3 Verification and area overhead estimation

Since we determined the way to implement the embedded shifts and in-situ negation at schematic level in the previous section, we can now proceed with some logic verification. In order to simulate its behaviour, a spice netlist of the proposed memory has been generated from the schematic view and HSPICE has been used as logic simulator. For the generation of the inputs, a vector file has been generated and incorporated in the main simulation file. The different operations tested as well as their expected result are summarized in the following table.

Tests			
Operation	Operand A	Operand B	Expected result
Add 16 bits	900A 900A	F008 F008	8012 8012
Add 16 bits + A shifted 3 times right	900A 900A	F008 F008	E209 E209
Add 16 bits + B negated and shifted 2 times right	900A 900A	F008 F008	9407 9407

Table 1: Tests realized in order to verify the functionality of the architecture "Embedded Shift + in-situ negation"

The result of the simulation is displayed on Figure 13. After a writing phase of the two operands, we execute the three operations in the next 3 cycles. The first operation tested is a reference in order to see if we can perform correctly the addition on 16 bits (no carry propagation between different words). Concerning the second operation, we are testing if data A is correctly shifted (becoming $A = F201\ F201$) and added to operand B. Finally, the last operation is testing the simultaneous use of the embedded shifts and the negation on operand B (becoming $B = 03FD\ 03FD$). As the results obtained in output (data_out) corresponds to those expected in Table 3, we can conclude on the correct functional behaviour of our implementation.

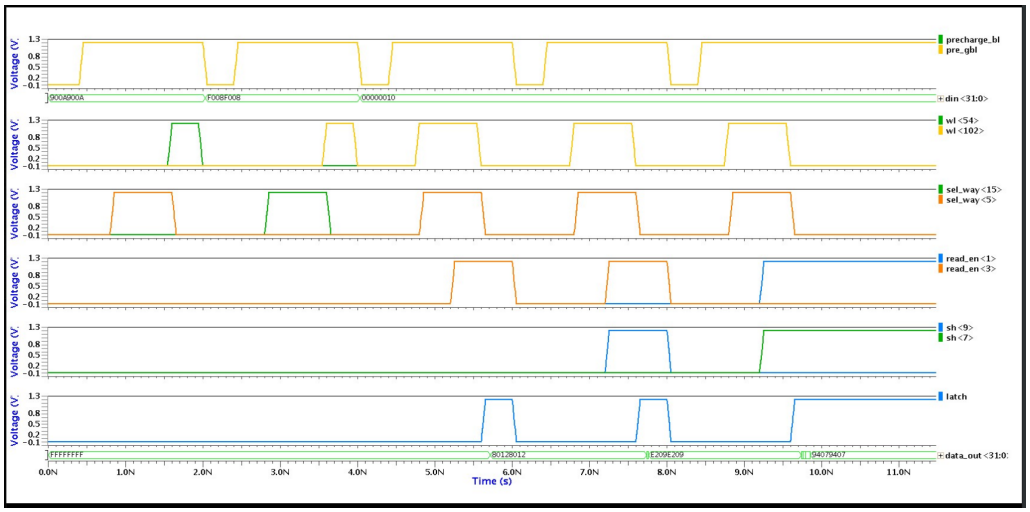


Figure 13: Simulation verifying the logic behaviour of the architecture "Embedded Shift + in-situ negation"

An another thing that can be interesting to know is the area overhead caused by the additional hardware in the LGP. Indeed, this allows us to get an estimation of how difficult it is to integrate those improvements in BLADE. Based on the layouts of a multiplexer and the read port (2 NMOS transistors), a rough estimation can be done. The estimation has been realized for two versions of implementation : the first one contains 4 embedded shifts as presented previously and the second one has only one ES in order to propose a lighter solution since we expect a huge area overhead due to the insertion of 4 ES and several multiplexers.
The results are summed up in Table 2.

Area Overhead		
Cells	Number of instances	Total area (μm^2)
Multiplexer	1	7.64
Embedded Shift	4 NMOS	3.70
LGP (Darkside)	1	20.46
LGP_V1 (max)	4 ES + 10 MUX	111.6
LGP_V1 (min)	4 ES + 2 MUX	50.51
LGP_V2 (max)	1 ES + 4 MUX	54.7
LGP_V2 (min)	1 ES + 2 MUX	39.4

Table 2: Area overhead estimation

As expected, the first version is almost 6 times bigger than the original one whereas the second version is only three times bigger. On one side, the first version (the most efficient one theoretically) will be really hard to implement inside the memory subarray since it will deal with LGPs having a large difference of size. This will create some heterogeneity inside the array and may make it fail. On the other hand, the second version is less efficient since we can only shift the data once but the LGPs involved in this solution have similar size which will ease its implementation at layout level. It has to be noted that we considered only the use of "real multiplexer" for this study. The advantage of this choice is that we can directly use the cells provided by the foundry but it involves a large number of transistors resulting in a huge area. For further work, we may think about implementing the same kind of multiplexer we used for selecting the way so as to only have 2 PMOS transistors instead of a dozen.

4 Power optimization

In the scope of edge computing, devices are relying on the use of batteries and by consequences, power saving is an important feature that has to be taken into account by the designer. We have seen in the previous section that energy can be saved through the amelioration of the hardware which enables a better computation efficiency. However, this amelioration is not sufficient for low power devices and additional improvements are often required. Thus, after the evaluation of the main responsible sources of energy consumption in the subarray, we identified and implemented other hardware optimizations that we will present in the following section. In addition, another way to save energy is to power only the instances and modules that are needed for the circuit to run and to deactivate the remaining instances. This technique is intensively used in integrated circuit design under the name of "Power Gating" and allows designers to reduce the power consumption and the leakage of a chip. Thus, in a second phase, we will detail a new floorplan in order to integrate a "Power gating" strategy in the subarrays. Finally, a third way to reduce the power consumption can be done by decreasing the supply voltage since we should theoretically decrease the source and gate leakage of transistors. However, decreasing the supply voltage of the bitcells during retention mode may provoke their failure and may result in data loss. That's why, the first part of this section will aim at characterizing the stability of our bitcells by evaluating their Static Noise Margin (SNM) so as to see if we can combine this idea with the previous ones.

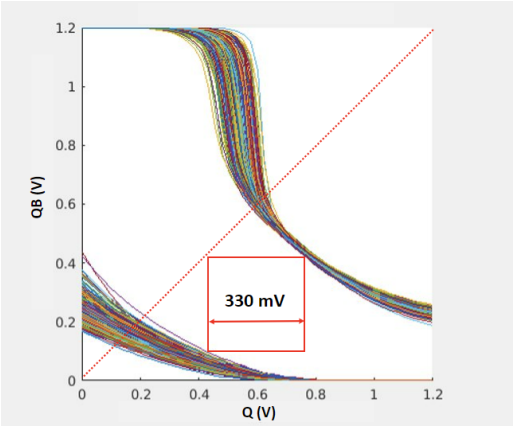
4.1 Bitcells characterization

4.1.1 Bitcells Static Noise Margin

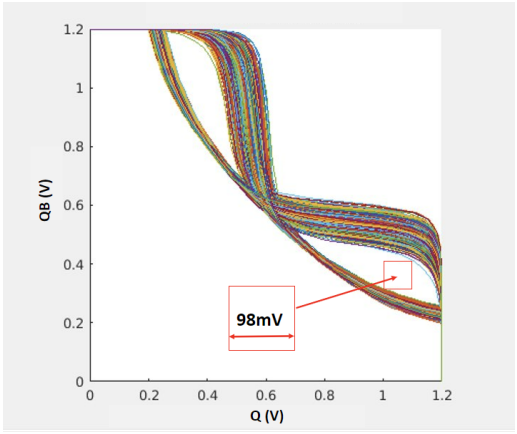
In order to be on the safe side, we only want to lower the supply voltage of the bitcells when they will enter the retention mode (see section 4.3 regarding power gating) and use a maximum supply voltage equal to 1.2 V when we will write and read the bitcells. Thus, we will first evaluate the Hold, Read and Write SNM at 1.2V over a wide range of temperature in order to check the stability of the bitcells during different operating conditions. The Hold SNM will be then characterized for different supply voltages and for several temperatures in order to determine the minimum retention voltage VDD_min_ret of the bitcells.

The Hold margin of a 6T SRAM bitcell represents its ability to keep the data with respect to a voltage perturbation occuring at one of its storage nodes. To evaluate this situation, we conducted a DC analysis by applying a voltage ramp from 0 to VDD on the storage nodes and then extracted the Voltage Transfer Characteristics (VTCs). Plotting them on the same graph, we can extract the Hold SNM which corresponds to the side of the maximum square we can fit inside the VTCs. In order to be robust against various Process, Voltage and Temperature (PVT) conditions, a Monte-Carlo study of 1k runs has been conducting for different temperatures. In a similar way, the Write and Read margin of the 6T SRAM bitcells have been extracted. A DC voltage ramp has been applied to both storage nodes successively and the VTCs obtained during the Monte-Carlo study have been plotted on the same graph. The results are depicted on Figure 14 and 15. We can observe on Figure 14c the typical "butterfly" curve usually obtained for bitcells as well as a degraded version for the Read operation on Figure 14b. This is due to the fact that the wordline and both BL and BLb are set to logic '1' which put the bitcell in its weakest state [17]. According to Figure 15, the SNM for the three operating conditions is decreasing when the temperature is increasing and this is probably due to an increase in

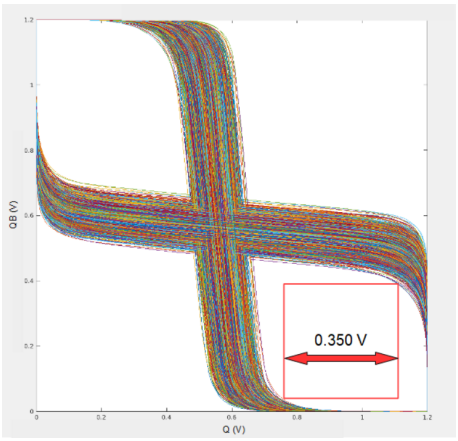
the leakage of the bitcells which make them weaker. However, since we obtained a decent SNM for each condition (>50 mV), we can affirm that our 6T SRAM bitcells are robust against PVT variations at 1.2V which should guarantee the good operation of the bitcells.



(a) Write



(b) Read



(c) Hold

Figure 14: Example of Hold, Read and Write SNM extraction realized on a 6T SRAM bitcell at 1.2V and 25C

To study the capacity of retention of the bitcells for different voltages, we conducted a 10k runs Monte-Carlo study covering a wide range of temperatures (from -40C to 125C). The results have been gathered in Figure 16. We can observe a degradation of the Hold margin when the temperature is increased and the voltage supply is decreased. In addition, we can see that the impact of the temperature is insignificant compared to the one produced by the reduction of the supply voltage. This is the reason why we decided at the beginning to keep the voltage as high as possible for Read and Write operations of the memory in order to prevent possible issues due to SNM degradation. However, since we obtained a Hold SNM greater than 50mV for the range of voltages studied, we can conclude that the bitcells are stable for low voltages until 0.4V. In order to be on the safe side, we decided to pick 0.6V as value for VDD_min_ret. This voltage provides us a good SNM (150 mV) which allows us to recover easily the values stored in the memory after waking up as well as providing us a good reduction of energy when the memory is not used.

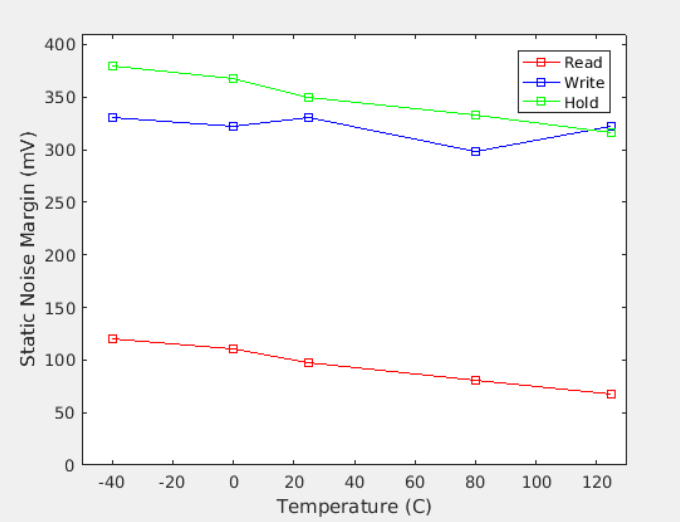


Figure 15: Hold, Read and Write SNM characterization for a 6T SRAM bitcell for different temperatures

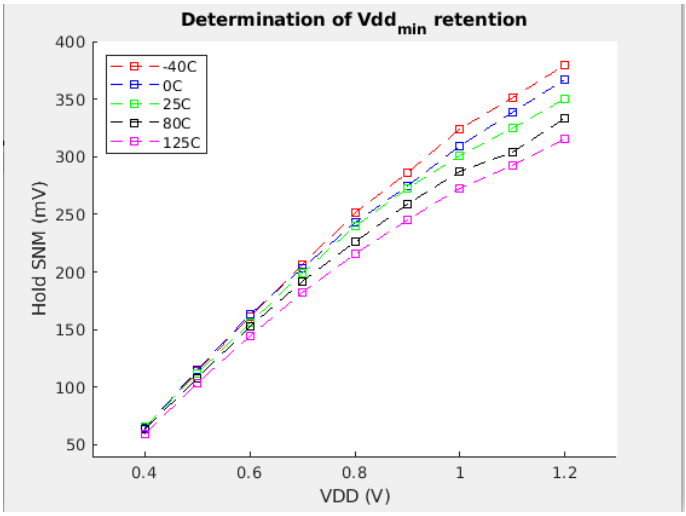
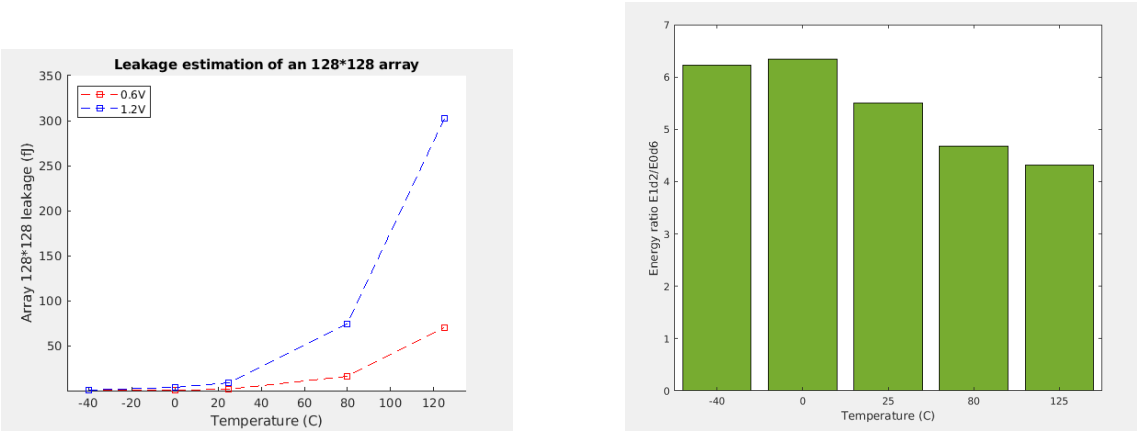


Figure 16: Determination of VDD_min retention

4.1.2 Bitcells leakage

In this section, we are aiming to get an estimation of the static consumption of a 2kB array of BLADE using the previously characterized 6T SRAM bitcells. To determine the total leakage of the array, the leakage of one bitcell has been determined using a generated netlist of a bitcell and simulated through HSPICE. The leakage of a transistor depending mostly on the temperature and the voltage supply used to bias it, the leakage has been estimated for various conditions of temperature and for two supply voltages (0.6V and 1.2V).

Figure 17a shows an exponential behaviour of the leakage with the increase of the temperature for both supply voltages which is coherent with the theory since the current due to thermal agitation is increasing exponentially with T. Over the range of temperature studied, dividing the voltage supply by a factor of 2 allows us to reduce the leakage by a factor of 5 on average according to Figure 17b. To conclude, reducing the supply voltage



(a) (b)

Figure 17: (a)Leakage estimation of a 2kB array in function of temperature and supply voltage, (b) Energy savings made when reducing the supply voltage from 1.2V to 0.6V

of the bitcells is a good approach to reduce their static consumption.

4.2 Intrinsic power optimization

4.2.1 Precharge and way optimization

Relying on bitline computing and the use of local groups, one of the main source of power consumption of BLADE corresponds to the precharge of its global and local bitlines. Because of their high number and high capacity, a huge amount of energy is indeed consumed for the precharge of those wires and is done at the beginning of each cycle. In its current implementation, all bitlines are precharged at the beginning of each cycle regardless the local groups involved in the operation. Knowing that the worst-case scenario is only involving 2 different LGs (during an IMC operation), we can easily deduce that more than 50% of the energy used for the precharge can be saved for each cycle if the precharge was carefully designed. The logical path implemented in BLADE is currently composed of a single signal *pre_charge* going through a simple inverter so as to activate the precharge block of the LBLs and GRBLs. Despite its simplicity, this implementation wastes a lot of energy since we actually don't need to precharge the GRBLs and LBLs during a write cycle as well as a part of the LBLs during a read cycle.

Figure 18: Schematic view of the implemented gates

In order to improve this architecture, inverters have been replaced with a NAND3 and

NAND2 in association with some control signals as shown in Figure 18. The signal OP<0> allows us to know if we are doing a write operation (logic 0) or an operation involving a read in the memory like during an IMC or a simple read (logic 1). On its side, en_lg<0> tells us if the data required is belonging to the first local group or not.

The logic behaviour has been verified using HSPICE as spice simulator and a netlist of the implemented gates. The result of the simulation is shown on Figure 19. We can see on this latter that we obtained the desired effect that is no precharge on write cycles and we activate only the precharge of useful LGs during Read cycles.

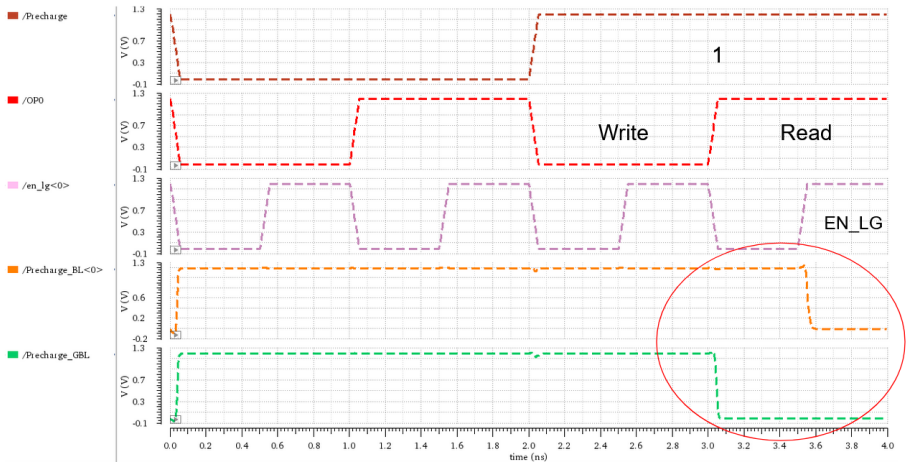


Figure 19: Logic verification of the implemented gates for precharge optimization

Since the logic has been checked, we can now try to estimate the energy saved by our implementation. To do so, the three following test cases have been implemented (where W=Write and R=Read) :

- Pattern 1 : **Start** - W1 - R1 - R2 - **Wait_1ms** - R1 - R2
- Pattern 2 : **Start** - W1 - W2 - R1 - R2 - **Wait_1ms** - R1 - R2
- Pattern 3 : **Start** - W1 - R1 - **Wait_1ms** - W1bar - R1

The first two patterns allow us to highlight the energy savings that we made during a read operation after a waiting time symbolizing an unused period of the subarray (SA). Pattern 1 is reading the data in the same LG whereas Pattern 2 is reading the data in different LGs. Finally, Pattern 3 aims at highlighting the benefits obtained for a write operation. The results are summarized in Figure 20 for each pattern.

Figure 36a and 36d show an energy reduction by 4 regarding the energy spent for the precharge of the LBLs compared to the previous implementation. This reduction was the one expected theoretically since we are now precharging only the local group that contains the data to be read and no more the four LGs constituting the subarray. Nevertheless, a 35% increase of the energy is noticeable during the second read cycle of Pattern 2 in order to precharge the LBLs. This effect is due to the fact that we didn't precharge all the LBLs of the subarray at the first Read cycle contrary to the previous implementation. However, the energy spent to precharge the LBLs of a LG is approximately equal to the energy spent for maintaining to VDD the LBLs of the entire SA and this little artefact will be largely

Cycles (after wait)	Darkside	Precharge optimization	Way optimization
R1_lbl	7.93 pJ	1.99 pJ	0.55 pJ
R1_gbl	0.72 pJ	0.72 pJ	0.70 pJ
R2_lbl	1.27 pJ	1.27 pJ	0.41 pJ
R2_gbl	0.90 pJ	0.90 pJ	0.88 pJ

14.4x reduction
4x reduction
3x reduction

(a) Pattern 1

Cycles (after wait)	Darkside	Precharge optimization	Way optimization
R1_lbl	7.94 pJ	1.98 pJ	0.54 pJ
R1_gbl	0.72 pJ	0.70 pJ	0.70 pJ
R2_lbl	1.32 pJ	1.78 pJ	0.54 pJ
R2_gbl	0.9 pJ	0.9 pJ	0.88 pJ

14.4x reduction
4x reduction
2.5x reduction
35% increase

(b) Pattern 2

Cycles (after wait)	Darkside	Precharge optimization	Way optimization
W1bar_lbl	7.94 pJ	~0	~0
W1bar_gbl	0.71 pJ	~0	~0
R1_lbl	1.32 pJ	1.78 pJ	0.52 pJ
R1_gbl	0.016 pJ	0.72 pJ	0.70 pJ

2.5x reduction
35% increase

(c) Pattern 3

Figure 20: Energy comparison between the three implementations

compensated by the savings of the first cycle.
On the other side, Figure 36c shows the energy spent for the precharge of the LBLs and GRBLs for a Write cycle followed by a Read cycle for the two implementations. By disabling the precharge on Write cycle, a huge quantity of energy can be saved while being able to write correctly the data but we still have to pay the price of precharging the unprecharged LBLs at the next Read cycle.

In the same manner, we can push further this idea at the way level in order to save more energy. Indeed, each time we are reading a data, only one way is selected out of 4. Thus, if we are able to only precharge the ways needed, we could reduce theoretically the energy spent by a factor of 16 compared to the implementation of Darkside (4 ways and 4 LGs per subarray). To this purpose, we can simply add an inverter followed by a NAND2 gate on the path of the signal pre_charge_lbl as shown by Figure 21. This implementation has to be repeated 16 times since we need to control the precharge of each way independently by using their own activation signal LG_way_en. This implementation has been logically verified in the same way as the previous study and tested on the same 3 patterns. The results are reported in Figure 20.
This time, the energy used during the first read cycle of Pattern 1 and 2 has been reduced by a factor of 14.4. The difference with the 16x factor expected from the theory may be explained by the large number of inverters and NAND2 gates that we added for this implementation as well as the additional leakage current coming from those gates. Moreover, we can also notice an improvement in the energy consumption during the second read of Pattern 1, 2 and 3. This is due to the fact that we are now precharging only one way of local bitlines out of the four we discharged during the previous Read cycle.

To better compared the two optimizations discussed previously, Figure 22 summarizes

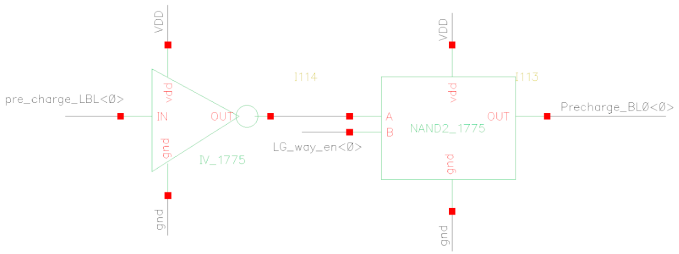


Figure 21: Logic used for optimizing ways precharge

the total energy spent for the blocks "Array+IO" with respect to the implementation of reference (Darkside).

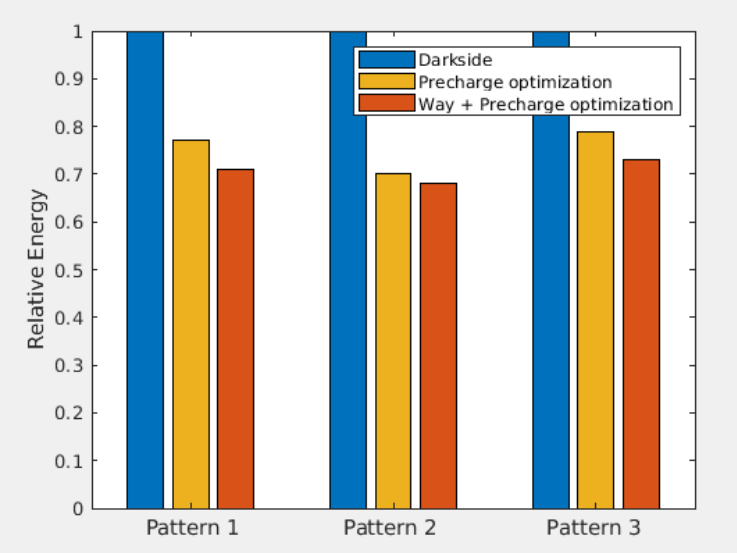


Figure 22: Energy consumption of the block "Array+IO" for the 3 implementations with respect to Darkside

As expected, the third implementation is the one that allows us to save the maximum amount of energy (around 30%). However, considering the number of additional gates that was implemented, the second one appears to be a good trade-off between efficiency and easiness of implementation. Indeed, the downside of the second solution relies on its implementation inside BLADE. This solution requires a lot of gates that need to be added in the controller of the SA and will imply lots of changes in the wiring of signals between the controller and each local group periphery (LGP) as well as inside this latter. As we are looking for regularity and homogeneity inside the array, we decided to implement the first solution in the layout of BLADE. (available in Annexe)

4.2.2 Control signals optimization

In order to decrease the power consumption of the array, another point that we found interesting to study is the generation of the control signals performed by the control part of the SA (Timing generator in Figure 1). This part is in charge of the generation of all control signals present in the subarray for its good operation such as the precharge signals, read and write enables, wordline activation, latch signals.... Currently, the timing of all

those signals is set through the use of a long chain of inverters which is responsible of the delay between the different signals as shown by Figure 23. With a fine characterization of the delay caused by a predefined number of inverters, this implementation is quite easy to implement but its main issue lies in this huge number of inverters (more than 200 in series) which have a high switching frequency since the delays are generated at each clock cycle.

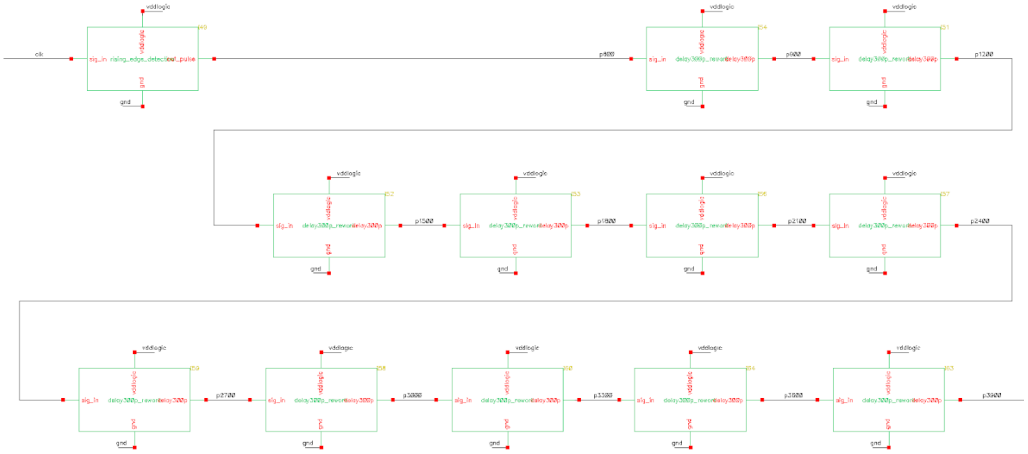


Figure 23: Schematic view composed of the rising edge detector followed by a long chain of inverters. Each block being composed of 20 inverters.

At it is, this implementation causes a lost of energy when we are not using the memory. To solve this issue, we decided to modify the design of the rising edge detector which triggers the whole chain of inverters at each rising edge of the clock. A simple AND gate has been added at the beginning in order to generate the control signals only when the subarray is enable (SA_EN=1).
In order to evaluate the efficiency of this optimization, we first evaluated the energy cost of the "Timing generator" block with and without the optimization for 100 clock cycles ($T_{clk} = 5ns$) mimicking an unused period of the subarray. The consumption of these blocks has been then compared to the consumption of the SA in the same conditions. The results are shown on Table 3.

Control signals optimization			
	Darkside Timer	Timer optimized	Darkside subarray
Energy	64.3 pJ	57.6 fJ	116 pJ

Table 3: Energy consumption estimation realized for 100 clock cycles with $T_{clk} = 5ns$

As shown by Table 3, we managed to suppress the consumption of the Timer generator since we reduced it by a factor of 1000 approximately. This consumption represented approximately 55% of the whole energy consumption of the subarray. Thus, this little modification will enable us to save a huge amount of energy when the subarray is inactive.

4.3 Power gating and Sleep Mode

Being an in-SRAM computing memory, BLADE belongs to the group of volatile memories. This means that we cannot stop to supply our SA without losing the data stored inside. Even if this characteristic may prevent the use of Power gating for such memories, many studies [18] [19] demonstrated that a good compromise was to split the memory

into different power domains so as to be able to control them independently. A memory being composed of the array that stores the data and some pure combinational logics, the combinational part can be disconnected from the power supply while the array is not. This state is usually called "Sleep mode" by memory designers and it allows them to save a lot of energy when the memory is not accessed.

4.3.1 Power domains and modes

As we have seen previously, "Power gating" can be applied to our subarray but only under some conditions. The peripherals (decoder, controller and IO) needs to be separated from the array in order to be biased independently. In order to save power at best, we thought about implementing 3 main power modes (ON/Sleep/OFF) for the SA which allow us to save energy accordingly to its usage. The Sleep mode will disconnect the peripherals from the power supply while keeping it for the array. This will save a lot of energy since it will reduce the power consumption of the memory to the leakage of the bitcells during this mode. However, since we demonstrated in section 4.1.2 that bitcells leakage reduces with the voltage supply, we can think about implementing a fourth mode, called Deep-Sleep, in which the peripherals are disconnected from the supply voltage and the voltage supplying the bitcells is reduced to VDD_min_ret, the minimum voltage that keeps bitcells in retention state. Those four modes are represented on Figure 24.

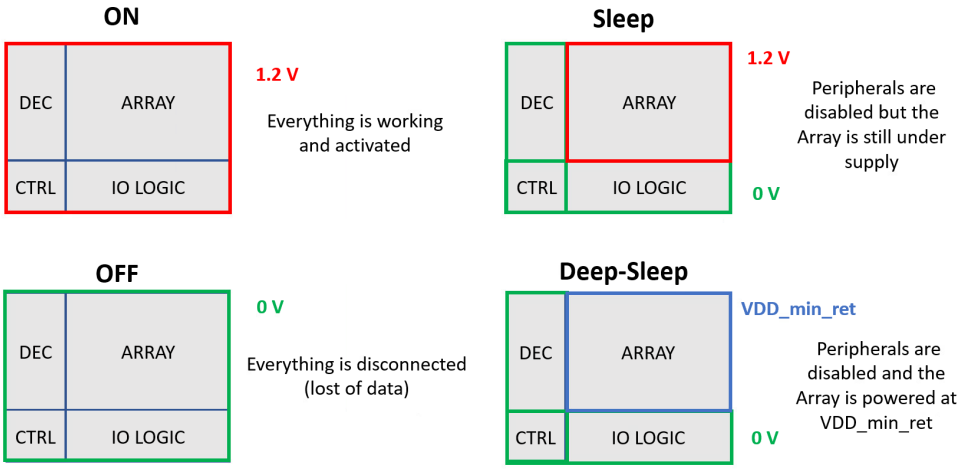
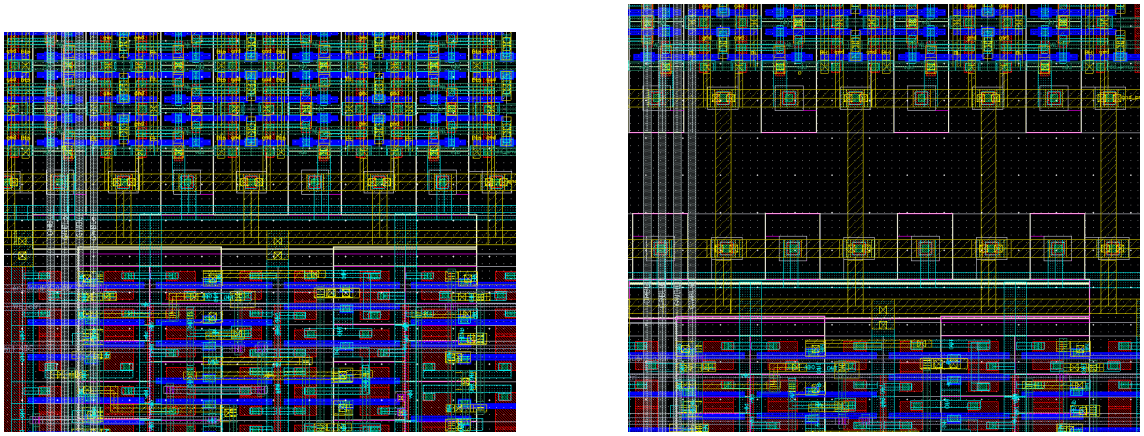


Figure 24: Schematic representation of the 4 power modes

In order to determine VDD_min_ret, we need to analyse the Hold Static Noise Margin (SNM) of the bitcells constituting BLADE. This step has been done during the bitcell characterization in section 4.1 and we found out that a voltage of 0.6V would ensure us a good SNM in order to be on the safe side. Thus, we will use that voltage for our Deep-Sleep mode.

4.3.2 Implementation

In order to implement the four modes described in the previous section, we need first to create different power domains in our memory as we need to supply them with different VDD. By consequence, we need to separate the NWELL of the peripherals and the array to make them independent. This implies to enlarge the whole design since we need to put additional TAP cells in order to properly biased the circuit as well as to respect the



(a) Previous layout (b) New layout

Figure 25: Layout views of the interface Array/IO before and after NWELL separation

DRC rules regarding adjacent NWELLs. The modifications done at the layout level are highlighted on Figure 25.

Once the NWELLs have been properly separated, we can control independently the supply voltage of each block. However, doing such implementations, we need to be sure that disconnecting the peripherals from the memory and the outside world will not disturb the rest of the circuit and especially our array which contains data. Thus, we need to add some "isolation cells" in order to control the value of the floating signals that may activate some parts of the circuit against our will. By consequences, those cells has to be placed at the interface Peripherals/Array and Peripherals/Outside.

On one hand, the control of the output signals at the interface Peripherals/Outside can be simply done by adding a NOR2 gate at each output where one of the input is tight to a control signal "Sleep" that takes the binary value 1 when we want to go in Sleep mode (cf. Figure 26). Therefore, this gate is transparent during "ON" mode and ensures a "0" output value during Sleep mode, preventing any fluctuation at the output of the memory.

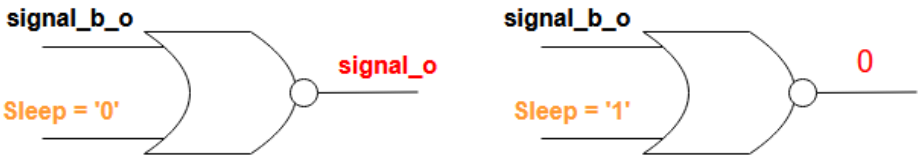


Figure 26: Isolation cell used for the memory outputs

On the other hand, we need to control also the control signals that come from the peripherals towards the array such as Write and Read enable signals, precharge signals, WL enable, etc... Before reaching the array, those signals are going through huge inverters in order to be able to drive the huge capacitances coming from the long metal lines crossing

the array. Those inverters impede us to add a gate as we did before for the output of the memory since we would kill the driving force of those latter. The solution found was to add a minimal size NMOS transistor with the source tight to the ground after the inverters so as to drive the signals to logic 0 (ie. WLs, write_en, read_en) or a minimal PMOS transistor when the signal needs to be maintained at VDD (ie. precharge).

As everything is set up to safely biased the different power domains of the SA, we can think about a way to bring the different voltages needed. To do so, we had two possibilities : either the different voltages (VDD=1.2V, VDD_min=0.6V and some intermediate voltages) are produced outside of the SA and brought inside with the help of some additional pins or we can try to implement a voltage manger directly inside the SA in order to reduce the number of signals that has to be managed by the controller of BLADE. For this work, we decided to investigate into the second option. According to the literature, several possibilities are existing in order to build a voltage manager : it can be achieved through the use of stacked PMOS [20], mock cells [21] or with a Low Drop-Out regulator (LDO) [22]. For this work, we decided to explore the last option because of its simplicity, low area and ability to reach voltages close to the supply voltage compared to other structures. Indeed, a LDO is a DC linear voltage regulator simply composed of a differential Amplifier and a PMOS sleep transistor as shown on Figure 27. The differential amplifier compares a voltage reference to a reference voltage Vref and adjust it through the Drain-Source voltage V_{DS} of the PMOS thanks to a negative feedback-loop.

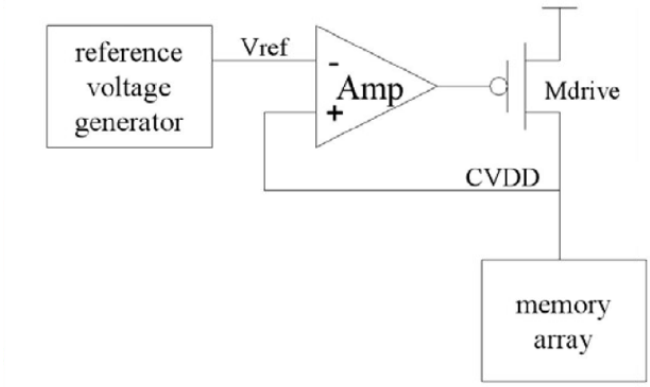


Figure 27: Block diagram of a LDO [22]

The main advantage of this structure is its tiny area so it can be easily inserted inside the layout of the decoder. However, its simplicity makes it harder to design so as to cover a wide range of voltages and to be robust against PVT variations. On figure 28, we can see that our initial implementation is able to follow accurately intermediate values of voltages but struggles to reach 1.2V and 0.6V. Because of the some issues related to the testing phase of Darkside which came back from fabrication at that time of the project, we decided to focus our attention on the verification of the chip and keep the improvements explained previously for a next iteration. By consequence, this is still an ongoing work that I would like to continue in the future. For instance, a more complex structure may be investigated in order to replace the differential amplifier. We may lose in terms of area but this will help us to gain in precision as well as robustness against PVT variations. As it can be noted on Figure 28, the current implementation does not allow yet a precise voltage tracking inside the array, specifically at low voltages. Thus, this specific point still needs some follow-up

works.

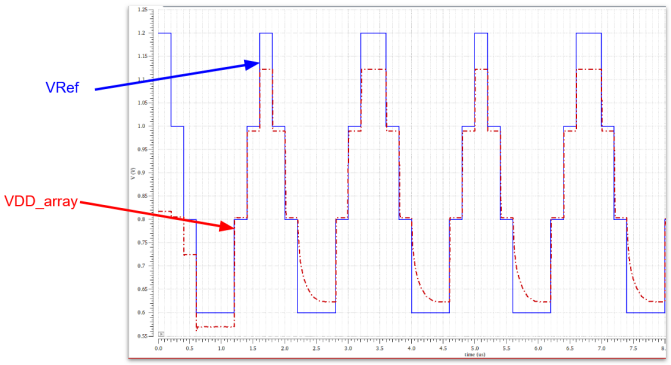


Figure 28: Simulation's result of our LDO obtained with spectre simulator

5 AMS verification flow

In preparation of the next BLADE's tapeout named HEEPpocrates expected for the end of 2022, we decided to concentrate our efforts on the verification of the connection between the analog and digital domains involved in BLADE. Indeed, this interface is more likely to be defective since it doesn't have a well established flow of verification compared to others instances (BLADE's controller and subarrays). Since BLADE's controller has been designed using a digital flow and the memory macro with Full-custom one, this interface has to be verified using an Analog Mixed-Signal (AMS) approach. Thus, at that point of my project, my work consisted in implementing an AMS flow of verification for the interface controller/memory in order to highlight and correct possible mistakes that may occur due to the use of two independent design flows.

5.1 AMS flow of verification

In order to perform a complete and systematic verification of the interface, we will start the verification by checking the logic behaviour at the interface and we will progressively replace the instances with more accurate models in order to end up with a system close to the reality.

Thus, the first step of the flow will consist in performing an intensive verification of the behavioural model of the subarray that has been used to design the controller. To that purpose, we decided to realize several tests. Each test tries to verify one functionality of the memory and is composed of two parts. The first one aims at verifying that the model and the memory are able to output the right result when the inputs are correctly set whereas the second part tries to verify that the memory is not working when the inputs are wrong. Indeed, the fact that the memory is working when it should not means that we may have done something wrong during its design and it may cause some trouble in the future. In case of a perfect match of behaviour with the real memory, this step will ensure us that any fault we may find out in the next steps are not coming from the design of the memory. The simulation will be carry with the AMS simulator of Virtuoso proposed by Cadence. In order to use it, a "config" view (similar to a schematic view) gathering the behavioural model and the designed memory has to be created. In addition, a vector file will be used in order to generate the inputs for both instances.
A schematic view of the testbench used to compare the model and the subarray as well as an example of VEC file are available on Figure 29 and Figure 30. As we can see on Figure 29, a XOR gate has been added at the outputs of the model and the memory so as to provide a logic '1' if the outputs differs. This feature will ease the checking step and save a lot of time since a lot of tests have to be performed.

Once the behavioural model is verified, we can insert the controller of BLADE in the testbench so as to create the interface understudied. The controller will be first tested in its digital form (RTL) in association with the memory macro. Once again, we need to run AMS simulations in the same way as previously since we want to transmit a digital signal coming from the controller to an analog instance (the memory). A vector file will be used to generate the inputs of the controller.
The testbench used for this simulation is available on Figure 31.

In order to get closer to the reality, the third and last step of this verification flow will

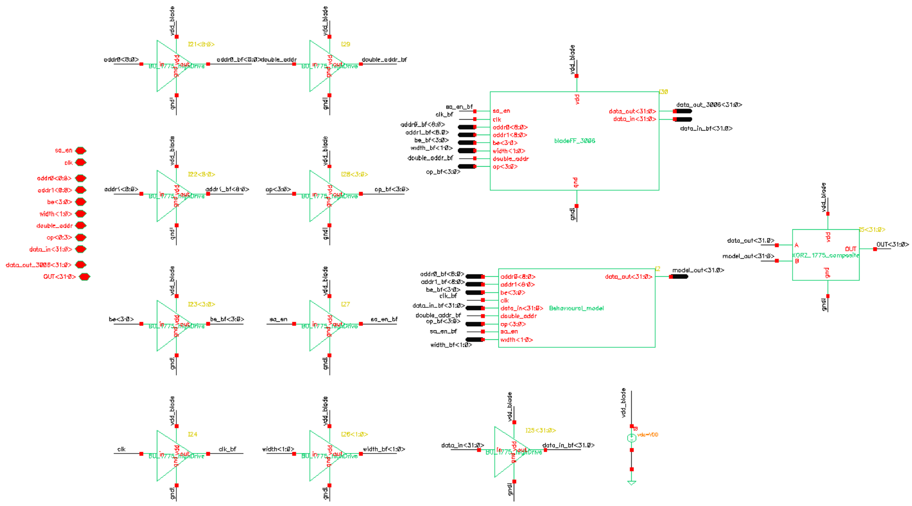


Figure 29: Schematic view of the testbench used for the verification of the behavioural model of the memory

```
RADIX 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
IO    i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i i
VNAME clk sa_en op[3] op[2] op[1] op[0] addr0[8] addr0[7] addr0[6] addr0[5] addr0[4] addr0[3] addr0[2] addr0[1] addr0[0] addr1[8] addr1[7]
addr1[6] addr1[5] addr1[4] addr1[3] addr1[2] addr1[1] addr1[0] double_addr data_in[31]  data_in[30] data_in[29] data_in[28] data_in[27]
data_in[26] data_in[25] data_in[24]  data_in[23]  data_in[22] data_in[21] data_in[20] data_in[19] data_in[18] data_in[17] data_in[16]
data_in[15]  data_in[14] data_in[13] data_in[12] data_in[11] data_in[10] data_in[9] data_in[8] data_in[7]  data_in[6] data_in[5] data_in[4]
data_in[3] data_in[2] data_in[1] data_in[0] be[3]  be[2] be[1] be[0] width[1] width[0]
TUNIT ps
SLOPE 100
PERIOD 2500
VIH 1.2
VIL 0
; CL SA  OP   A0    A0      A1    A1     DB  DI    DI    DI    DI    DI    DI    DI     DI    BE    WI
  0  1  1100  11  0000000  00  0000000  0  0000  1111  0000  1111  0000  1111  0000  1111  1111  00    ; address 0 way 3    ->  0f0f 0f0f
  1  1  1100  11  0000000  00  0000000  0  0000  1111  0000  1111  0000  1111  0000  1111  1111  00
  0  0  1100  00  0000000  00  0000000  0  0000  0000  0000  0000  0000  0000  0000  0000  0000  00
  1  0  1100  00  0000000  00  0000000  0  0000  0000  0000  0000  0000  0000  0000  0000  0000  00
  0  1  1100  10  1000000  00  0000000  0  1010  1010  1010  1010  1010  1010  1010  1010  1111  00    ; address 64 way 2   ->  aaaa aaaa
  1  1  1100  10  1000000  00  0000000  0  1010  1010  1010  1010  1010  1010  1010  1010  1111  00
  0  0  1100  00  0000000  00  0000000  0  0000  0000  0000  0000  0000  0000  0000  0000  0000  00
  1  0  1100  00  0000000  00  0000000  0  0000  0000  0000  0000  0000  0000  0000  0000  0000  00
  0  1  1100  01  1100100  00  0000000  0  0000  0001  0000  0001  0000  0001  0000  0001  1111  00   ; address 100 way 1   ->  0101 0101
  1  1  1100  01  1100100  00  0000000  0  0000  0001  0000  0001  0000  0001  0000  0001  1111  00
  0  0  1100  00  0000000  00  0000000  0  0000  0000  0000  0000  0000  0000  0000  0000  0000  00
  1  0  1100  00  0000000  00  0000000  0  0000  0000  0000  0000  0000  0000  0000  0000  0000  00
  0  1  1100  00  1111111  00  0000000  0  0000  0000  0000  0000  0000  0000  0000  0000  1111  00   ; address 127 way 0 ->  0000 0000
  1  1  1100  00  1111111  00  0000000  0  0000  0000  0000  0000  0000  0000  0000  0000  1111  00
  0  0  1100  00  0000000  00  0000000  0  0000  0000  0000  0000  0000  0000  0000  0000  0000  00
  1  0  1100  00  0000000  00  0000000  0  0000  0000  0000  0000  0000  0000  0000  0000  0000  00; 4 WRITE
```

Figure 30: Example of VEC file used for an AMS simulation

consist in simulating the controller synthesized with a spice netlist of the memory. The synthesis step will convert the RTL design of the controller into a netlist of logic gates. This simulation will better mimic the reality since it will associate to each gate described in HDL a real gate associated to its timing and input capacitances. To perform this task, we used Design Compiler from Synopsys. In order to run the synthesis, we need to provide the tool with the following files :

- The physical libraries in LEF format for the 65nm TSMC standard cells and BLADE. These files contain information about the technology process used for the routing layers (name, pitch, size, spacing...).
- Timing libraries in LIB format for the standard cells and BLADE. They contain timing information (set up/hold, delay ...) for different PVT corners. We will use a worst and best case to verify timing constraints.

Once the synthesis is done, we obtain in output a verilog netlist of the synthesized controller. This netlist is then converted into a spice netlist in order to be simulated with HSPICE later. We use the tool v2cdl from Cadence for the translation and the simulation is run with HSPICE. We also validated the functionality of this flow with fast spice simulators such as Finesim in order to reduce the simulation time by several order of magnitude

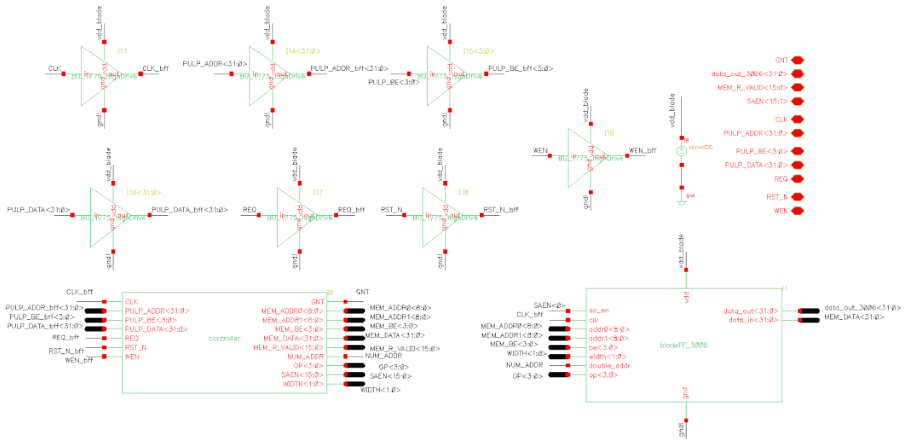


Figure 31: Schematic view of the testbench used for the verification of the interface controller(RTL)/memory

(estimated between 10x and 100x). Even if those simulations are less precise than HSPICE simulations, it provided a great help during the debugging phase.

5.2 Results

As mentioned in the previous section, the first step of this verification flow consisted in verifying the correctness of the behavioural model of the memory. To that purpose, a lot of cases have been tested but, for the sack of this thesis, we are going to describe only one. This latter was aiming to check the functionality of the addition step. Thus, after a writing and reading phase of two operands ($0F0F$ $0F0F$ and 0101 0101) in different LGs as well as the initialisation to 0 of the result address, the operation between the operands is realized two times in a row. In the first one, the control signals are set correctly whereas the signal double_address is not activated for the second addition in order to mimic a defective behaviour. A XOR gate has been added between the outputs of the memory and the model so as to ease the checking step (Figure 29). The result of the simulation is available on Figure 32.
First of all, we can see that the behavioural model and the memory are giving the same outputs since the signal OUT (output of XOR gates) remains at logic "0" during the whole simulation. Looking now at the output model_out, we can see that the result is only correct when the signal double_addr is asserted since the memory outputs only the first operand otherwise. Thus, we can conclude that the memory and its model are working in this situation. After several simulations as the one described previously, we concluded that the memory and its model were working as expected.

Since the behavioural model and the memory are behaving the same way, we decided to start the verification of the interface controller/memory. Thus, we first simulated the HDL description of the controller with a HSPICE netlist of the memory on Cadence Virtuoso. For this simulation, we tested a set of operation composed of bitwise operations such as XOR and NOR as well as more complex operations (ADD, ADD + SHIFT). The testbench used for this step is shown on Figure 31. As previously, this simulation turned out to be correct and confirmed the good functionality of the interface at that point. The result of the simulation is available in Annexe.

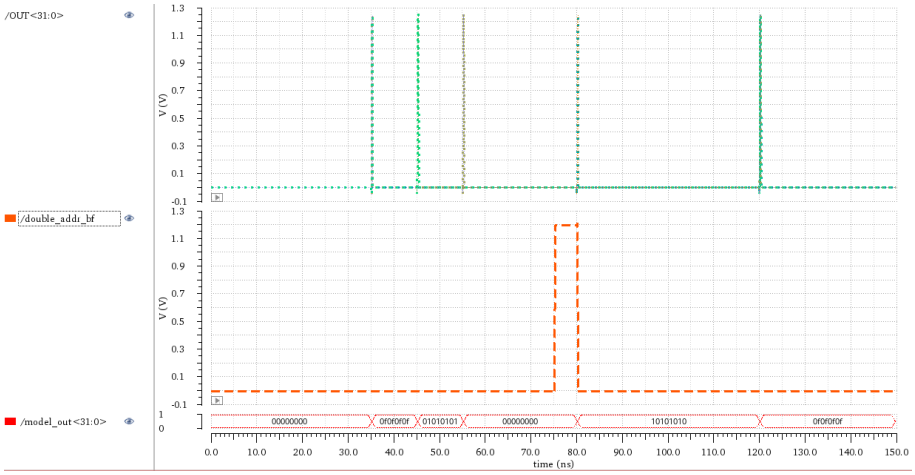


Figure 32: Simulation verifying the functionality of the memory and its behavioural model for the addition step

Finally, to simulate an interface closer to the one implemented on-chip, the controller has been synthesized with Design Compiler. This step provided us a verilog netlist that we translated into spice. We added this latter to the spice netlist of the memory so as to create the interface and simulated this file with HSPICE. Results showed that the interface was not working : some signals arriving in the subarray were either incoherent or inverted.

At that point, we suspected that a different endianness has been used between the controller outputs and the subarray inputs. Looking into the files used for the synthesis of the controller, we found out that 5 signals have been defined from the LSB to the MSB in one of them whereas those same signals have been defined from MSB to LSB in the subarray(Figure 33).

```
wire [1 : 0][ADDR_BITS-1 : 0]          memAddr;
wire                                   numAddr;
wire [NUM_SAS-1 : 0]                   saEN;
wire [1 : 0]                           width;
wire [3 : 0]                           op;
wire [DATA_WIDTH/8-1 : 0]              be;

wire [DATA_WIDTH-1 : 0]                data_in;

wire [0 : ADDR_BITS-1]                         memAddr0;
wire [0 : ADDR_BITS-1]                         memAddr1;
wire                                           memNumAddr;
wire [NUM_SAS-1 : 0]                           memEn;
wire [0 : 3]                                   memOp;
wire [1 : 0]                                   memWidth;
wire [0 : DATA_WIDTH/8-1]                      memBe;
wire [0 : DATA_WIDTH-1]                        memDataIn;
wire [NUM_SAS-1 : 0]                           RValid;
wire [DATA_WIDTH*NUM_SAS-1 : 0]                memRData;
```

Figure 33: Different endianness used for signals definition

Thus, to verify if this can be the source of the problem, we did again the synthesis step with the corrected endianness and launch another simulation. Unfortunately, the simulation failed as well but we suspected this time an issue related to the translation of

the netlist from verilog to spice since we obtained different errors when running twice the netlist translation step with the same input file. This hypothesis was very hard to verify because we assumed that v2cdl identical nets verification was working properly. However, we later noticed that the identification of identical nets was defective and responsible of the errors previously detected. For instance, we can see on Figure 34 that node n729 has been correctly distinguished from node N729_no_collide but not n731 and N731 which results in a short circuit.

```
XU227 n729 n706 n803 n723 N729_no_collide VDD VSS OAI22D1
XU228 SAEN[11] n803 VDD VSS INVD1
XU229 n729 n707 n800 n723 N730 VDD VSS OAI22D1
XU230 SAEN[12] n800 VDD VSS INVD1
XU231 n729 n708 n804 n723 N731 VDD VSS OAI22D1
XU232 SAEN[13] n804 VDD VSS INVD1
XU233 n729 n709 n801 n723 N732 VDD VSS OAI22D1
XU234 SAEN[14] n801 VDD VSS INVD1
XU235 n729 n710 n805 n723 N733 VDD VSS OAI22D1
XU236 SAEN[15] n805 VDD VSS INVD1
XU237 n337 n807 n786 n808 n586 VDD VSS OAI22D1
XU238 n367 n789 n351 VDD VSS ND2D1
XU240 n338 n711 n339 n585 VDD VSS OAI21D1
XU241 n338 n696 n339 n584 VDD VSS OAI21D1
XU242 n338 n697 n339 n583 VDD VSS OAI21D1
XU243 n338 n698 n339 n582 VDD VSS OAI21D1
XU245 n338 n699 n339 n581 VDD VSS OAI21D1
XU246 n338 n700 n339 n580 VDD VSS OAI21D1
XU247 n338 n701 n339 n579 VDD VSS OAI21D1
XU248 n338 n702 n339 n578 VDD VSS OAI21D1
XU249 n338 n703 n339 n577 VDD VSS OAI21D1
XU250 n338 n704 n339 n576 VDD VSS OAI21D1
XU251 n338 n705 n339 n575 VDD VSS OAI21D1
XU252 n338 n706 n339 n574 VDD VSS OAI21D1
XU253 n338 n707 n339 n573 VDD VSS OAI21D1
XU264 n338 n708 n339 n572 VDD VSS OAI21D1
XU265 n338 n709 n339 n571 VDD VSS OAI21D1
XU275 n338 n710 n339 n570 VDD VSS OAI21D1
XU276 n787 n879 n309 n587 VDD VSS OAI21D1
XU277 n337 n786 VDD VSS INVD1
XU286 n462 n785 VDD VSS INVD1
XU287 n618 n783 VDD VSS INVD1
XU288 n616 n782 VDD VSS INVD1
XU289 n468 n784 VDD VSS INVD1
XU290 n737 n347 n731 GNT VDD VSS OAI21D1
XU291 n667 n723 VDD VSS CKBD1
```

Figure 34: Short circuited netlist

For further work, we can try to solve this issue through one of the following options :

- Investigate into the options of Design Compiler to remove the use of uppercase in the generation of the verilog netlist;
- Investigate into the options of v2cdl to detect all the conflicts;
- Change the translator for the one of Synopsys (netTran) so as to keep the same provider for the synthesis step and the netlist translation;
- If none of the above options work, we can write a python script which detects contentious nodes and solves conflicts;

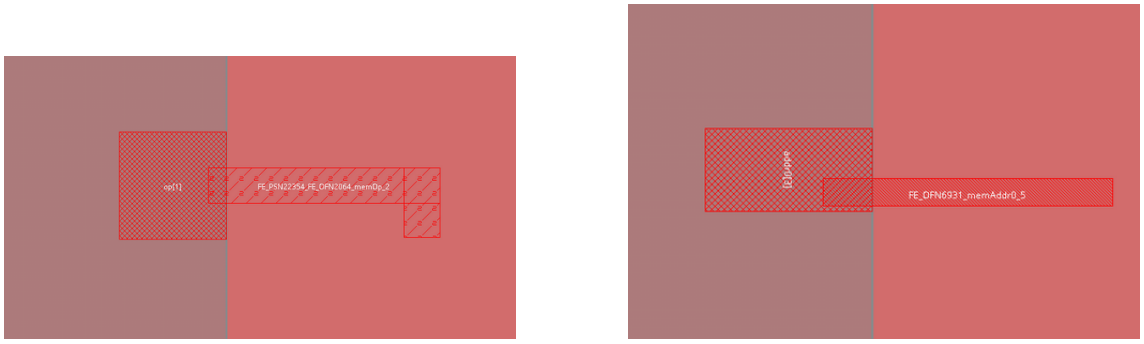


Figure 35: GDS view of Darkside showing a pin inversion on different signals

Meanwhile, we managed to receive the gds view of Darkside from ETH and we had the confirmation that some pins have been indeed inverted during the Place&Route step as depicted by Figure 35. This error was not detected during the second step of the flow because the tool from Cadence was able to correctly do the connection even if different endianness were used. Thus, depending on the tool used, different interpretations are possible.

To conclude this section, we have seen that many differences (endianness, case sensitivity) are existing between the tools coming from different providers and may cause the failure of a circuit. So, we need to be really careful when we use tools from different providers in a design flow and a good practice during the conception of a design would be to keep the same endianness for all design levels, specifically at the interface between full-custom and digital design since, according to the tool used, a different interpretation is possible.

6 Conclusion

As a conclusion, BLADE is an In-Memory Computing architecture with a great potential for AI applications. Throughout this study, we tried to propose a solution to the main design constraints such as latency and power consumption which still constitute nowadays a barrier for the integration of Convulational Neural Network (CNN) in IoT edge devices.

On the one hand, the latency issue was assessed through a design exploration at schematic level of new logical instances aiming at accelerating the computation efficiency of multiplications. We demonstrated indeed that we could increase the computation speed up to 4 times thanks to the use of Embedded Shift and Embedded Negation inserted inside each Local Group Periphery. However, as a general rule in circuit design, the improvement of one figure of merits always implies some trade-off. In our case, we saw that we should accept some area overhead to the benefit of the computation rate.
On the other hand, we proposed as well different solutions in order to improve the power consumption of BLADE. We first proposed two new designs for optimizing the precharge of the LBLs and GRBLS resulting in an energy saving between 25% and 30%. Then, we proposed a new subarray floorplan in order to obtain a power-gated array with four different states. The bitcells have been characterized to propose an optimal implementation of the retention state with a good trade-off energy-robustness. However, some work need still to be done on the implementation of the voltage manager in order to bias correctly and precisely the different power domains.

Finally, we designed an Analog Mixed-Signal (AMS) flow for validating the interface between the full custom 2kB subarray and its digital controller. This flow will help us to carefully implement the new version of BLADE inside HEEPpocrates that will be fabricated in the end of 2022.

7 Glossary

AI : Aritficial Intelligence

AMS : Analog Mixed-Signal

BLADE : BitLine Accelerator for Device on the Edge

CNN : Convulational Neural Network

CPU : Central Process Unit

ES : Embedded Shift

GRBL : Global Read Bitline

HDL : Hardware description Language

IMC : In-Memory Computing

LBL : Local Bitline

LG : Local Group

LGP : Local Group Periphery

LRP : Local Read Port

LSB : Least Significant Bit

MSB : Most Significant Bit

MUX : Multiplexer

PVT : Process Voltage Temperature

RTL : register Transfer Level

SA : Subarray

SNM : Static Noise Margin

SRAM :Static Random Access Memory

References

[1] F.Tsimpourlas and al. "A Design Space Exploration Framework for Convolutional Neural Networks Implemented on Edge Devices". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* PP (July 2018), pp. 1–1. DOI: 10.1109/TCAD.2018.2857280.

[2] C.Chae and al. "A Multi-Bit In-Memory-Computing SRAM Macro Using Column-Wise Charge Redistribution for DNN Inference in Edge Computing Devices". In: *2021 18th International SoC Design Conference (ISOCC)* (2021), pp. 421–422. DOI: 10.1109/ISOCC53507.2021.9613934.

[3] R.Gauchi and al. "Memory Sizing of a Scalable SRAM In-Memory Computing Tile Based Architecture". In: *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)* (2019), pp. 166–171. DOI: 10.1109/VLSI-SoC.2019.89203734.

[4] W.Simon and al. "BLADE: A BitLine Accelerator for Devices on the Edge". In: *GLSVLSI '19, May 9–11, 2019, Tysons Corner, VA, USA* (2019). DOI: 10.1145/3299874.3317979.

[5] W.Simon and al. "BLADE: An in-Cache Computing Architecture for Edge Devices". In: *IEEE Transactions on Computers* 69.9 (2020), pp. 1349–1363. DOI: 10.1109/TC.2020.2972528.

[6] ETH Zurich. *The IIS Chip Gallery : Rosetta*. URL: http://asic.ethz.ch/2019/Rosetta.html. (accessed: 24.07.2022).

[7] ETH Zurich. *The IIS Chip Gallery : Darkside*. URL: http://asic.ethz.ch/2021/Darkside.html. (accessed: 24.07.2022).

[8] Memorism Processor. *Glossary*. URL: https://aot-slid.com/en/explanation/terms/. (accessed: 26.07.2022).

[9] Embedded. *In-memory compute enables more efficient edge processing*. URL: https://www.embedded.com/in-memory-compute-enables-more-efficient-edge-processing/. (accessed: 26.07.2022).

[10] R. Gauchi and al. "Memory Sizing of a Scalable SRAM In-Memory Computing Tile Based Architecture". In: *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)* (2019), pp. 166–171. DOI: 10.1109/VLSI-SoC.2019.8920373.

[11] K.C. Akyel and al. "DRC2: Dynamically Reconfigurable Computing Circuit based on memory architecture". In: *2016 IEEE International Conference on Rebooting Computing (ICRC)* (2016), pp. 1–8. DOI: 10.1109/ICRC.2016.7738698.

[12] ETH Zurich. *Github : pulp-platform/pulp*. URL: https://github.com/pulp-platform/pulp. (accessed: 24.07.2022).

[13] A. Agrawal and al. "X-SRAM: Enabling In-Memory Boolean Computations in CMOS Static Random Access Memories". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 65.12 (2018), pp. 4219–4232. DOI: 10.1109/TCSI.2018.2848999.

[14] Francesco Conti. *Hardware rocessing Engines release 2.0*. URL: https://hwpe-doc.readthedocs.io/_/downloads/en/latest/pdf/. (accessed: 24.07.2022).

[15] M. Rios and al. "An Associativity-Agnostic in-Cache Computing Architecture Optimized for Multiplication". In: *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)* (2019), pp. 34–39. DOI: 10.1109/VLSI-SoC.2019.8920317.

[16] F. Ponzina and al. "A Flexible In-Memory Computing Architecture for Heterogeneously Quantized CNNs". In: *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)* (2021), pp. 164–169. DOI: 10.1109/ISVLSI51109.2021.00039.

[17] B. Alorda and al. "Static-noise margin analysis during read operation of 6t sram cells". In: *DCIS Proceedings* (2009).

[18] Y. Wang and al. "A 4.0 GHz 291 Mb Voltage-Scalable SRAM Design in a 32 nm High-k + Metal-Gate CMOS Technology With Integrated Power Management". In: *IEEE Journal of Solid-State Circuits* 45.1 (2010), pp. 103–110. DOI: 10.1109/JSSC.2009.2034082.

[19] H. Pilo and al. "A 450ps Access-Time SRAM Macro in 45nm SOI Featuring a Two-Stage Sensing-Scheme and Dynamic Power Management". In: *2008 IEEE International Solid-State Circuits Conference - Digest of Technical Papers* (2008), pp. 378–621. DOI: 10.1109/ISSCC.2008.4523215.

[20] M. Huang and al. "An Energy Efficient 32-nm 20-MB Shared On-Die L3 Cache for Intel® Xeon® Processor E5 Family". In: *IEEE Journal of Solid-State Circuits* 48.8 (2013), pp. 1954–1962. DOI: 10.1109/JSSC.2013.2258815.

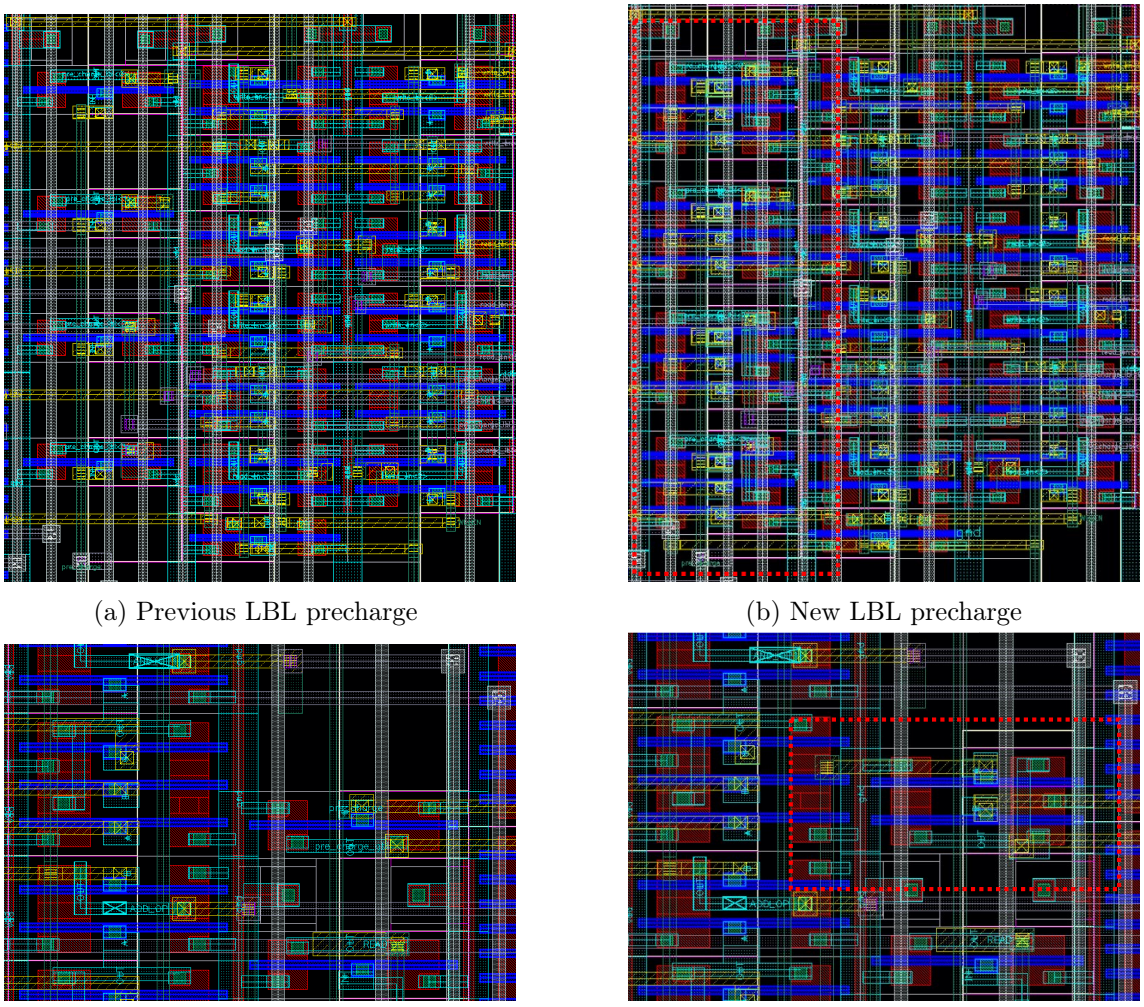
[21] C. Dray and al. "A 40nm low power SRAM retention circuit with PVT-aware self-refreshing virtual VDD regulation". In: *2010 IEEE International Memory Workshop* (2010), pp. 1–4. DOI: 10.1109/IMW.2010.5488323.

[22] C.F. Lee and al. "On-Chip VDC Circuit for SRAM Power Management". In: *2007 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)* (2007), pp. 1–4. DOI: 10.1109/VDAT.2007.372757.

[23] ESL. *ESL | Embedded Systems Laboratory*. URL: https://www.epfl.ch/labs/esl/. (accessed: 19.08.2022).

8 Annexe

8.1 Technical part



(a) Previous LBL precharge

(b) New LBL precharge

(c) Previous GBL precharge

(d) New GBL precharge

Figure 36: Layout views related to the precharge optimization

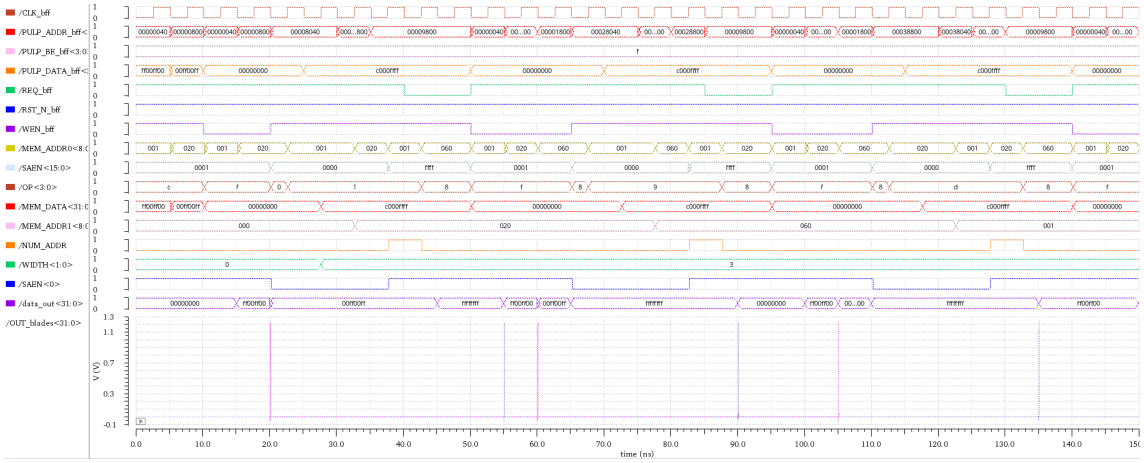


Figure 37: Simulation verifying the functionality of the memory and the controller in its HDL description (Part1)

Figure 38: Simulation verifying the functionality of the memory and the controller in its HDL description (Part2)

8.2 Embedded System Laboratory (ESL)

Being part of the Institute of Electrical and Micro Engineering at EPFL, the Embedded System Laboratory focuses on the definition of system-level multi-objective design methods, optimization methodologies and tools for high-performance embedded systems and nano-scale Multi-Processor System-on-Chip (MPSoC) architectures targeting the Internet-of-Things (IoT) Era[23]. Currently, the head of the lab is composed of David ATIENZA, director of the ESL laboratory, and two other professors. The lab is currently counting 11 post-docs researchers and more than 20 PhD students coming from all around the world in order to assess today's challenges related to embedded systems. My internship was conducted inside the "BLADE" team overseen by Alexandre LEVISSE and Davide SCHIAVONE. A Gantt diagram can be found in the following in order to have an outlook of how this internship ran through.