POLITECNICO DI TORINO

DIPARTIMENTO DI ELETTRONICA E TELECOMUNICAZIONI (DET)

Master Degree in Electronic Engineering

Master Degree Thesis

Advanced SDRAM controller architecture for Approximate Computing

Supervisor
Prof. Maurizio Martina

Candidate
Angelo Fusillo

April, 2019

Acknowledgements

I would like to thank my supervisor Prof. Maurizio Martina at Politecnico di Torino for its constant help. Our meetings and his advice have been fundamental to me to make progress in this thesis work. Every time I had a problem or a doubt, he was always available to discuss and support me in finding the best solution. That was very important to me.
Thanks also to Prof. Guido Masera and PhD candidates Riccardo Peloso and Maurizio Capra: they provided to me inputs to get on with the work and our discussions have been very useful to solve any problem I encountered during these months. Thanks to Prof. Shafique and PhD candidate Alberto Marchisio from TU Wien, they gave me lots of ideas and new works to focus on in order to find new solutions.
Last but not least, I have to express my immense gratitude to my family for their continuous support in any sense, giving me the strength to face all the obstacles I encountered during my university career: all this would not have been possible without them.

Summary

Refresh operations are the main cause of reduced performances in modern systems using DRAMs as main memory. This task is unavoidable to have a reliable dynamic memory but it is power consuming and increases the accesses latency of a service request. Thus, this thesis proposes an advanced and reconfigurable hardware architecture whose main aim is to reduce the refresh overhead in a typical chip of DRAM. The designed architecture communicates with the memory and provides a way of reducing the amount of refreshes that a standard controller usually addresses to the memory. The architecture exploits the actual retention times of the memory rows, that come after an initial profiling, and then postpones the unnecessary refreshes until the very moment at which a given row needs to be refreshed.
This kind of solution is widely treated in literature, but this work provides an innovative implementation that supports and draws benefits from memory accesses, that are seen as issued refreshes as well. Then allows to characterize the cells' retention times with a pattern suited to the used memory capacity and handles a challenging problem for DRAMs like temperature variations.
Finally, the designed implementation provides a reconfigurable feature that allows the user to have a memory controller calibrated on the system application: in some cases, the presence of non-critical data saved in memory can allow to skip at all refreshes for those locations, achieving further benefits in terms of power savings and response latency.

Contents

List of Tables

List of Figures

List of Abbreviations

PRE Precharge

ACT Active

CBR CAS-Before-RAS

RGR Row Granular Refresh

SDR Single Data Rate

DIP Dual In-Line Package

RRM Row Refresh Machine

FSM Finite State Machine

FIFO First In First Out

ASM Algorithmic State Machine

DPD Data Pattern Dependance

VRT Variable Retention Time

LFSR Linear Feedback Shift Register

RAS Row Access Strobe

SRGR Selective Row Granular Refresh

RTL Register Transfer Level

BL Burst Length

RP Row Precharge

RRD Row-to-Row Delay

FAW Four Activate Window

ODT On-Die Termination

Chapter 1

Introduction to approximate memory computing

1.1 Background and motivation

Nowadays systems are based on dynamic random-access memory (DRAM) as building block for main memory. They are widely used in a large number of applications and in mobile systems too. A typical DRAM cell stores the bit content in the form of a charge in a capacitor. The charge leaks off from the capacitor through its access transistor and the data stored is lost. This is the reason for which, to avoid losing the data, DRAM cells need to be refreshed, that means reading out and restoring the content in the cells in a periodic way. These operations, however, are degrading in terms of system performances and wasted energy, resulting in large power consumptions. The first problem is mainly due to the fact that a refresh operation cannot be interrupted whenever started: if a user request comes during a refresh, this will be delayed in the worst case for a time equal to the refresh cycle time, that depends on the memory architecture too. So this interference caused by the normal working with the surrounding system increases the latency of the operations this memory is used for.
Regarding the wasted energy, a refresh operation needs to precharge the bitlines and close the row (PRE) and then activate it (ACT) to store the contents in the *row buffer* of sense amplifiers: at the end of the operation the data is restored again in the current row preventing the data to be lost. These operations are, as obvious, power consuming if one considers that this sequence has to be executed as many times as the number of rows present inside the memory array. All these problems are supposed and expected to get worse in the future technology scaling, where DRAM cells density is expected to rise.
Modern DRAM devices refresh the cells according to the one that shows to leaks off more than the others: this cell determines the rate of refreshing for the entire

memory. In fact, regarding this, the manufacturers set the refresh period to 64 ms to avoid losing data and this refresh time is widely used in modern DRAM standards. However, most of them can *retain* their data for a period longer than the actual *Auto-Refresh* (CBR) one of 64 ms even if this command is optimized to reduce the overhead described before, for example refreshing one or multiple rows per bank in parallel. So, even if a so small value of refresh period is used to avoid corruptions and guarantee the integrity of the data stored, the time that a typical DRAM cell can sustain without losing the content, known as *retention time*, is quite longer. This means that most of the refreshes issued by the controller are not necessary to guarantee the correct behavior.
The idea comes from this consideration: by knowing cell retention times, it would be possible to group DRAM rows into retention times *bins* and apply different refresh rates at each row. This is feasible and it is corroborated by the fact that there is a very reduced number of weak cells in a DRAM memory. Then, skipping unnecessary refreshes and so lowering the rate at which cells are refreshed today, it would be possible to gain great benefits in terms of power consumptions and system performances. This will lead to the concept of *Approximate Memory Computing*, where the system can rely on an *Approximate Memory* obtained with the refreshes reduction solution. Prior works have focused on this idea of minimizing the overhead of refresh, like "RAIDR" [1] implementation, where in a 32 GB DRAM used as main memory reaches 74.6 % of refresh reduction, a power saving in average of 16.1 % and performances improvements of 8.6 % at the cost of a retention times storage overhead. This overhead depends on the implemented architecture, if it is a codesign of both hardware and software or if it is merely either a hardware or software solution. Nevertheless, the purpose is the same and aims in reducing this unnecessary introduced latency: in fact, as showed in work [1] and reported here in Figure 1.1, the *cumulative cell failure probability* is almost null at the refresh period of 64 ms and a reduced number of cells requires to be refreshed at a rate of 256 ms, that is four times the standard refresh interval.

The realized hardware controller presented in this work is a FPGA-based implementation that provides several features that allow to face typical DRAM issues that will be described in successive chapters. The idea of performing refreshes based on actual retention times relies on the fact that, basically, DRAM cells, especially in modern devices, show a retention time that extends approximately from 1 to 6 seconds at room temperature (till 45 °C) making, as mentioned, most of refreshes unnecessary. In fact, such short refresh interval of 64 ms takes into account an increase of temperature up to 85 °C, where cells retention time decreases exponentially. In Figures 1.2 and 1.3, taken from the experimental studies performed on different chip families and with different capacities [2], the normalized retention time at different room temperatures and the cumulative distribution of the retention times for the tested chips are reported respectively.

In Figure 1.2, the retention time of a certain cell, evaluated at different room

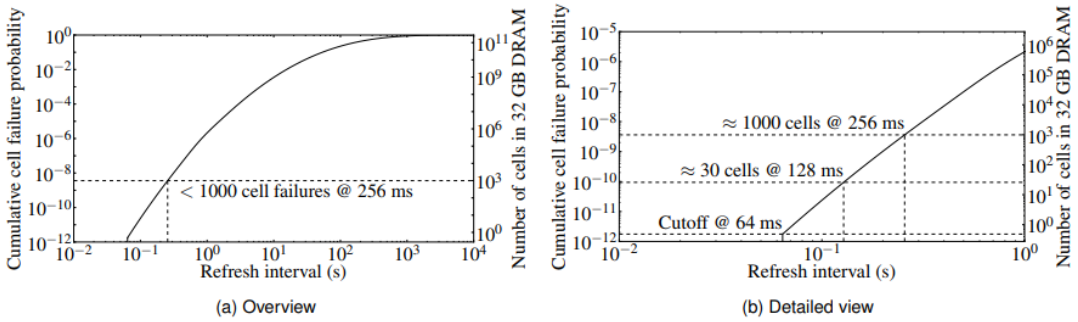


Figure 1.1: Cumulative cell failure probability overview and detailed view [1]

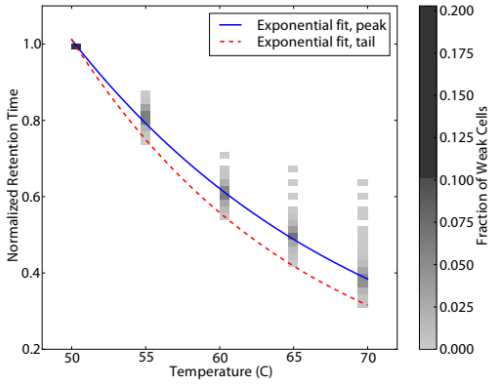


Figure 1.2: Normalized retention time versus temperature [2]

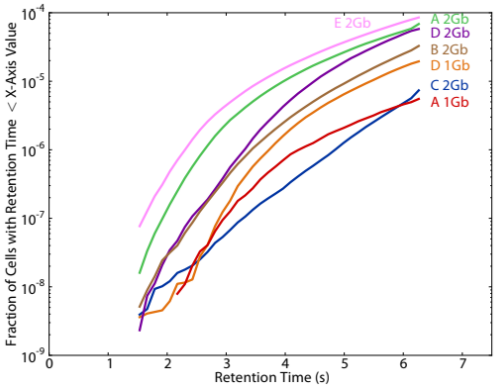


Figure 1.3: Cumulative distribution of retention times [2]

temperatures reported on horizontal axis, is normalized to the retention time of the same cell at 50 °C. The important consideration that comes out from this is that the retention time of a DRAM cell falls exponentially as the temperature rises. It would mean that, in some situations, the retention times chosen for particular rows as refresh rate could be different if the temperature changes and so this poses a challenging issue to select the correct refresh rate in order to avoid data corruption. In Figure 1.3, the retention times distribution has been obtained at 45 °C: as visible, the fraction of cells with retention time below about 1.5 s is very small, meaning that a large percentage of cells could be refreshed at lower refresh rate accordingly. So taking into account these considerations and if the retention times are known, the controller could be able to issue different refresh rates to each row, where the weakest cell fixes the minimum retention time sustained by that row. To do that, it is necessary to issue single row addresses to be refreshed on the bus in contrast with the highly optimized *Auto-Refresh* used in modern DRAM chips: in practice,

the idea is to establish again the *RAS-only refresh* feature present in older DRAMs and refresh every single row according to its actual retention time. This, as known, could be expensive in terms of activating each single row but, if the distribution of the retention times is similar to the one mentioned in previous experimental works, one could achieve considerable power savings and refresh overhead reductions at the cost of an enlargement of the controller area, without doing any modification to the DRAM itself.
So, in general, the high impact of refresh in modern devices has led the researchers to realize new schemes in order to reduce the latency of a refresh operation or to reduce the amount of refreshes issued to the memory itself. Even if the modern *Auto-Refresh* is optimized to operate on multiple rows, the higher is the density of cells, the higher is the impact of this overhead on system performances and power consumptions. In the previous mentioned work [1], an estimation of the refresh latency, throughput loss and power consumption has been performed across the device capacity. Figure 1.4 taken from this work shows how things are going to worsen in future, where throughput loss and power consumption can jeopardize the constraints posed by a given system, making these devices unusable.
Clearly, when the density of cells per row rises, then as well increases the access

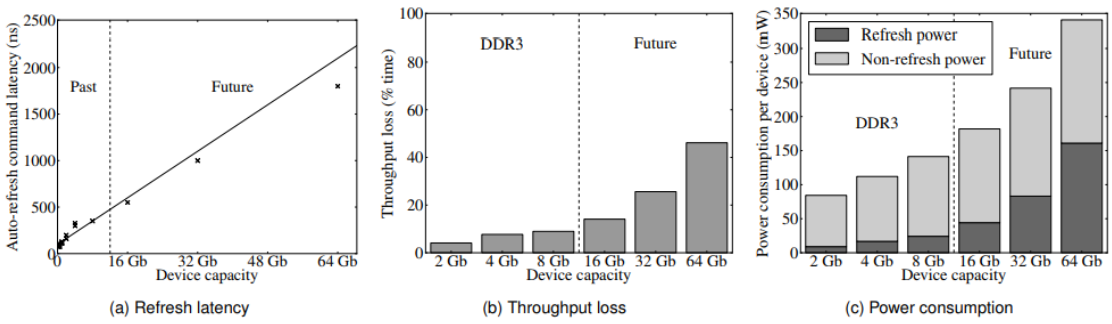


Figure 1.4: Effects of refresh on current and future DRAM devices [1]

time to the desired location due to the longer wordlines, causing so a high access latency. The power consumption is affected too, considering that all the bitlines of the same row are precharged first and driven after an active command to the issued bank: when opening a row to perform a refresh, it requires to drive the entire wordline of the row to be accessed and all the bitlines. Moreover, bitlines have a big parasitic capacitance, then making this operation of activating a row very expensive in terms of power and access latency.

1.2 Auto-Refresh feature in DRAMs

In modern DRAM chips the controller issues an *Auto-Refresh* command to the memory every tREF = 64 ms to ensure data integrity at normal room temperature conditions. The DRAM reacts by refreshing some rows in all banks using an internal counter, where the number of issued rows depends on the capacity and density of the memory device. In order to refresh the entire memory in the tREF window, *Auto-Refresh* commands are issued at a fixed time interval called tREFI. From work [4], this interval can be calculated as:

$$tREFI = \frac{tREF * r}{R} \tag{1.1}$$

having the total number of rows per bank (R) and the number of rows per bank that are refreshed every tREFI (r).
During normal room temperature conditions, which is indicated by the manufacturers to be up to 85 °C, the average time delay between refresh commands is 7.8 µs. When the DRAM works in conditions where the temperature rises between 85 °C and 95 °C, the refresh interval is set to 3.9 µs meaning that the refresh window tREF is halved to 32 ms. In Figure 1.5 from work [4] it is summarized what happens during a refresh window.
Whenever an *Auto-Refresh* command is issued every tREFI, this operation involves

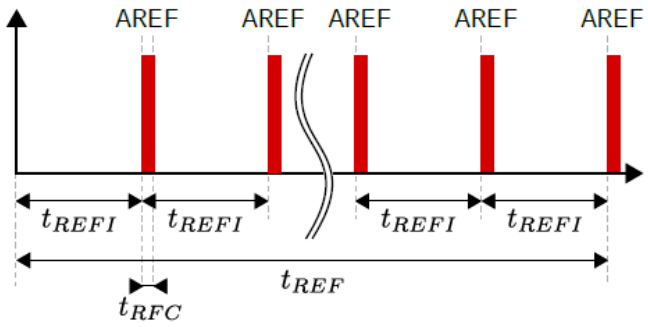


Figure 1.5: Refresh cycle window tREF [4]

all the banks of the given rank at the same time and no accesses are allowed to be served on that rank. The number of accessed rows per bank (r) can be easily calculated by inverting the previous formula, having so:

$$r = \left\lceil \frac{tREFI * R}{tREF} \right\rceil \tag{1.2}$$

All the involved banks of the same rank are occupied for a time delay called *refresh cycle time* (tRFC) from performing any other operation: this negatively affects performances and energy efficiency too in terms of increased access latency and decreased hit rate, causing a reduction of memory throughput in postponing requested accesses. In fact, in modern DRAMs, the refresh latency given by tRFC is, in average, in the range from about 200 ns to 300 ns and such long time delay heavily degrades performances. Throughput loss evaluated as the ratio between tRFC (time spent in doing refreshes) and tREFI (time interval between each refresh command) is expected to reach the half of the totality of the memory throughput in the future (Figure 1.4) and the power consumptions due to refresh are going to become the prevailing contribute to the total DRAM power. For these motivations, a solution to reduce refreshes overhead has been proposed and it will be discussed in the successive chapters.

1.3 Overview of the proposed design implementation

The controller, aware of the retention times distribution of the DRAM cells, could be able to perform row-by-row refreshes at different rates, known also in literature as *Row Granular Refresh* [4]. This, as mentioned, would reduce the overhead given by the *Auto-Refresh* carried out by modern DRAM devices.
But how the controller can be aware of the retention times distribution of the memory rows? The controller architecture, hence, has to perform first an initial profiling of the actual retention times of each row and save these values in a storage, also called *retention times bins*.
Then, whenever the retention time of a given row has elapsed, that row has to be refreshed to avoid to lose data integrity. How to perform single row refreshes at different rates? The *Auto-Refresh* feature doesn't allow such a kind of granular refresh, so a different strategy has to be followed: as mentioned before, the idea is to restore the *RAS-only refresh* used in older DRAMs, especially in the asynchronous ones. Although vendors don't say clearly how the refresh operates, it is quite known that a row is activated (ACT) for an access, its content is written in the *row buffer* of sense amplifiers and it is kept there till a successive row is required to be served: at this point, the content is restored again in the row and the bank is precharged (PRE) preparing it for a successive operation, closing so the previous opened row. So, a sequence of ACTIVE (ACT) and PRECHARGE (PRE) commands has the same effect of refreshing, hence it will be used by the designed controller to issue a *RAS-only refresh*.
In order to avoid data corruption, every row refresh is issued every 64 ms and the controller has to check if its retention time (saved as multiple of the same refresh window tREF) has elapsed: if so, that row is refreshed.

In conclusion, *RAS-only refreshes* could help to reduce the refresh overhead, provided that the distribution of the retention times is similar to the one mentioned in previous experimental works. The proposed hardware solution offers a way of getting the entire profiling of the memory cells retention times, challenging with some problems that will be described in detail; then offers a way of exploiting memory accesses requests to further reduce the refresh overhead and last but not least a way of handling the temperature variation, which has been demonstrated to be crucial in causing retention times variations among the memory cells.

Chapter 2

Basic SDRAM memory controller

2.1 General organization

The basic idea is to realize a memory controller that is aware of the actual retention times of the cells and refresh each row accordingly, implementation also known in literature as *Row Granular Refresh* (RGR [4]). The FPGA-based hardware platform has been designed and then tested by using the DE1-SoC development board provided by Terasic® and its integrated ISSI® IS42S16320D-7TL SDR SDRAM, whose capacity is 512 Mb. The used memory configuration, in terms of bus parallelism, is 8M rows x 4 banks x 16 data bits bus (64 MegaBytes). The chosen clock frequency is 143 MHz, with a *CAS Latency* of 3 cycles. From the datasheet [7], the memory issues 8K (N) refresh cycles every 64 ms and from the equation 1.2 it is possible to extract the number of rows per bank (r) that are refreshed every tREFI:

$$r = \frac{R}{N} = 1 \tag{2.1}$$

Hence, one row per bank is issued every refresh command to complete, in N commands, the refresh of all the rows in a tREF window.
The starting point is to realize a basic memory controller able to issue standard *Auto-Refresh* and single word reading and writing operations. This controller is completely configurable, meaning that the user can manually set the most important parameters before using it. To correctly perform operations, the memory controller has to respect timing constraints fixed by the DRAM and sent corresponding commands accordingly. The configuration allows the user to set the following timing parameters in terms of clock cycles:

- tRAS_CYCLES → Row Access Strobe cycles

- tRCD_CYCLES → Row to Column Delay cycles
- tRP_CYCLES → Row Precharge cycles
- tDPL_CYCLES → Input Data To Precharge Command Delay cycles
- tRFC_CYCLES → Refresh Cycle Time cycles
- tMRD_CYCLES → Mode Register Set To Command Delay cycles
- CAS_LATENCY → CAS Latency cycles
- CLK_FREQUENCY → Clock Frequency in MHz
- REFRESH_TIME → Refresh Period tREF in ms
- REFRESH_COUNT → Number of refreshes in a refresh window (N)
- BANK_WIDTH → Bank Address Width
- ROW_WIDTH → Row Address Width
- COL_WIDTH → Column Address Width
- HADDR_WIDTH → Total Address Width
- SDRADDR_WIDTH → Address Width on the SDRAM side

Provided that the wanted configuration has been chosen, the user can be able to communicate with the SDRAM issuing reading and writing operations. The state diagram of the basic SDRAM controller is depicted in Figure 2.1.

At reset assertion, the controller enters through the initialization states where the *Mode Register* is loaded. As the datasheet of the memory suggests, at power-up a minimum initial delay of 100 µs is required before issuing any command apart from NOPs. Then, the initialization steps provide a precharge of all banks and a double cycle of *Auto-Refreshes* before issuing the *Load Mode Register* command correctly, where the internal memory configurations are set such as data bus width, *CAS Latency* and mode of operation (single word accesses or burst accesses): in this very first realization, only single word accesses are provided. At this point, the memory is configured with the desired operation mode and is left idle to receive write or read commands.

The refreshes sent through the CBR *Auto-Refresh* are handled by a monitor counter: every time the counter exceeds the threshold corresponding to 7.8 µs (tREFI), the controller enters in the refresh states where all the banks to be refreshed are closed first (*Precharge* command) and refreshed then (*Auto-Refresh* command). Each command is a sequence of bits recognized by the memory: for a better visual understanding, the command truth table referred to the used SDRAM taken from the datasheet [7] is shown in Figure 2.2.

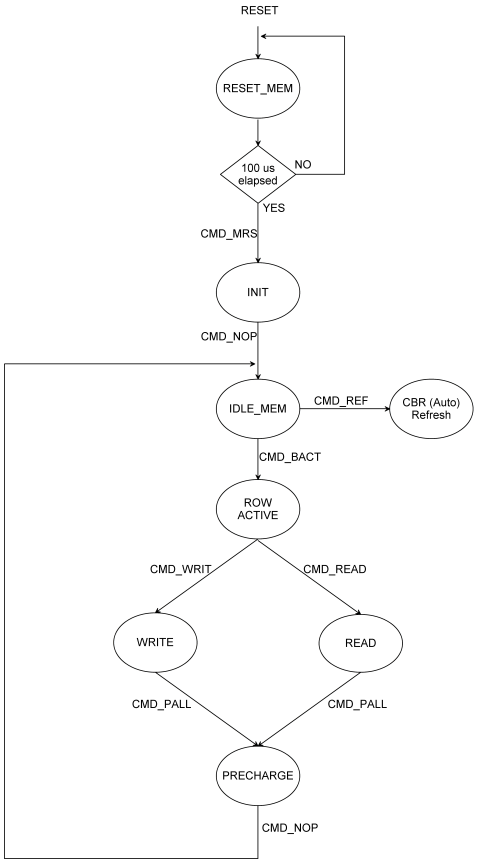


Figure 2.1: Basic SDRAM controller state diagram

COMMAND TRUTH TABLE

Function	CKE n – 1	CKE n	$\overline{CS}$	$\overline{RAS}$	$\overline{CAS}$	$\overline{WE}$	BA1	BA0	A10	A12, A11 A9 - A0
Device deselect (DESL)	H	×	H	×	×	×	×	×	×	×
No operation (NOP)	H	×	L	H	H	H	×	×	×	×
Burst stop (BST)	H	×	L	H	H	L	×	×	×	×
Read	H	×	L	H	L	H	V	V	L	V
Read with auto precharge	H	×	L	H	L	H	V	V	H	V
Write	H	×	L	H	L	L	V	V	L	V
Write with auto precharge	H	×	L	H	L	L	V	V	H	V
Bank activate (ACT)	H	×	L	L	H	H	V	V	V	V
Precharge select bank (PRE)	H	×	L	L	H	L	V	V	L	×
Precharge all banks (PALL)	H	×	L	L	H	L	×	×	H	×
CBR Auto-Refresh (REF)	H	H	L	L	L	H	×	×	×	×
Self-Refresh (SELF)	H	L	L	L	L	H	×	×	×	×
Mode register set (MRS)	H	×	L	L	L	L	L	L	L	V

Note: H=V_{IH}, L=V_{IL} x= V_{IH} or V_{IL}, V = Valid Data.

Figure 2.2: Command truth table of the used SDRAM [7]

For the used memory configuration, if the memory clock frequency is set to 143 MHz and there are 4 banks refreshed in parallel composed by 8192 rows each, the clock cycles between refresh commands (tREFI) at that frequency are:

- CLK_FREQUENCY = 143 MHz
- REFRESH_TIME = 64 ms
- REFRESH_COUNT = 8192

$$CYCLES\_REF = \left\lfloor \frac{CLK\_FREQUENCY * REFRESH\_TIME}{REFRESH\_COUNT} \right\rfloor \quad (2.2)$$

That corresponds to 1117 cycles. This means that every time the refresh counter overcomes this threshold, the controller moves towards the *Auto-Refresh* and so refreshes the rows according to its internal counter. Then, in the proposed idea to issue row-by-row refreshes, this simply results in never making the counter reach the *CYCLES_REF* threshold: an enable signal is inserted and so the *Auto-Refresh* can be disabled. However, you must accept the condition that the controller never goes in *Self-Refresh* states, otherwise the memory refreshes the rows by itself according to its internal oscillator and counter. A phase-locked loop has been used in the FPGA-based implementation to generate a SDRAM clock frequency of 140 MHz from a clock of 50 MHz provided by the board (CLOCK_50), not exactly 143 MHz to keep relaxed on timing constraints. This means that the new threshold for *CYCLES_REF* is equal to 1093 cycles.
In Figure 2.3 the top level entity of the realized basic SDRAM controller is depicted, showing all the provided signals.
The host interface side has the ports of addresses for writing and reading, write and read enable signals to issue corresponding commands, input write data port when writing and output read data port when reading. Moreover two additional signals are provided: a busy signal (*BUSY*) saying when you are not allowed to issue commands, meaning that the memory either is already executing a writing or reading operation, or it is refreshing or it is in the initialization steps where the user is configuring the operation mode; the second signal is a ready reading signal (*RD_READY*), saying when the data to be read is available on the bus. These two signals will be fundamental when communicating with the memory. The *REF_CNT_EN* signal is used to disable the *Auto-Refresh* feature.
The SDRAM side of the component has all the signals corresponding to the pins needed to drive the memory itself. In the bottom part of the figure, the generic parameters are assigned a value corresponding to the used configuration. Just a note on the *SDRADDR_WIDTH* parameter needed on the SDRAM side pins: it refers to the maximum between *ROW_WIDTH* and *COL_WIDTH*, because the

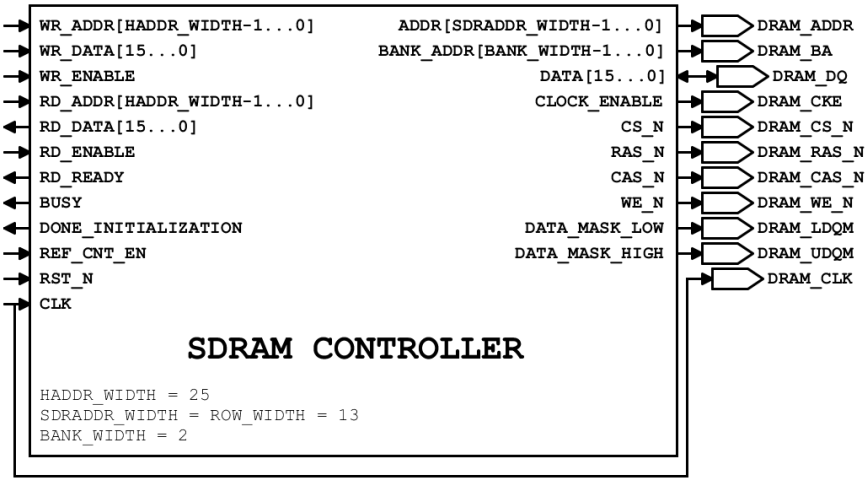


Figure 2.3: Top level basic SDRAM controller

address bus is unique and row addresses and column addresses are alternated on this during the common operations. So the memory address is calibrated on the row address width, that is the maximum among the two.
Looking at Figure 2.1, when the memory is idle and a reading or a writing operation is issued, the controller first enters in the *ROW ACTIVE* state where the row in the requested bank is activated, that means the row is opened by activating the wordlines: so in this phase, bank and row addresses are provided to correctly perform this row opening. Then, after a time equal to the tRCD delay, the column address can be placed on the bus to access the 16 bits location that has been requested: if a reading operation is requested, after the *CAS_LATENCY* delay the data will be present on the bus and the *RD_READY* signal asserted to acknowledge the user; otherwise, if a writing operation is issued, the controller will drive the bitlines and write the data put on the bus by the user inside the pointed memory location. After this, the bitlines of all the banks are precharged (PRE) preparing the memory for a successive operation. The timing constraints are verified by inserting NOP states between two issued commands, as all the datasheets show in their timing diagrams.
This basic controller has been tested to work by designing a simple testing platform on the development board, exploiting push-buttons and DIP switches to issue operations and send both data and addresses to the memory: it provides FIFOs to store issued operations and sent them to the memory. FIFOs are used to handle both reading and writing operations and they are one operation deep: this means that they themselves put in busy state the user interface whenever an operation has been already taken in charge from the controller. The user interface testbench is pretty much simple and since it is out of the scope, it will be omitted in the following sections.

2.2 Controller enhancement

The basic controller has been enhanced to provide burst support and power-down mode. The state diagram and the top level entity are reported in Figures 2.4 and 2.5, and a description of the added signals will follow.

The output signal *DONE_INITIALIZATION* is asserted by the controller when

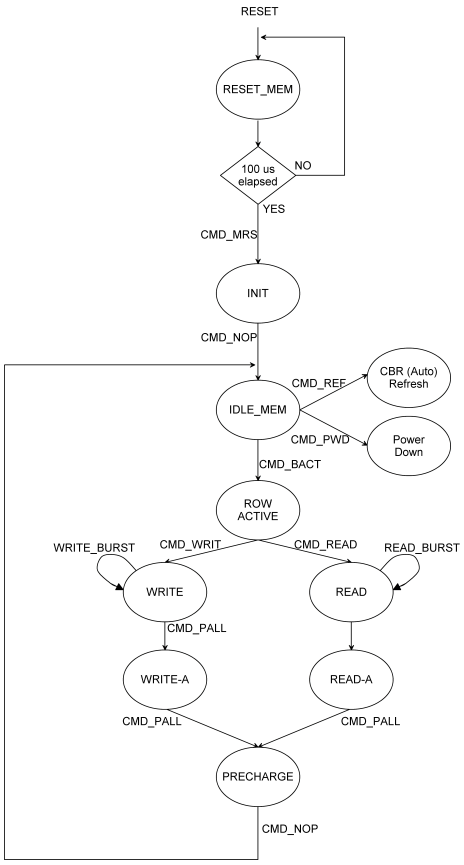


Figure 2.4: Enhanced SDRAM controller state diagram

the 100 µs time delay has elapsed at power-on. Since then, the memory is ready to receive commands and the controller is left idle.

The power down mode has been added through a homonym input signal: when asserted, the memory controller puts the SDRAM in this mode where the clock is suspended. Take care that during the period in which the SDRAM is in this state, no refreshes are provided: this means that the user has to reactivate the clock signal by exiting from this state before a time corresponding to a refresh cycle of 64 ms

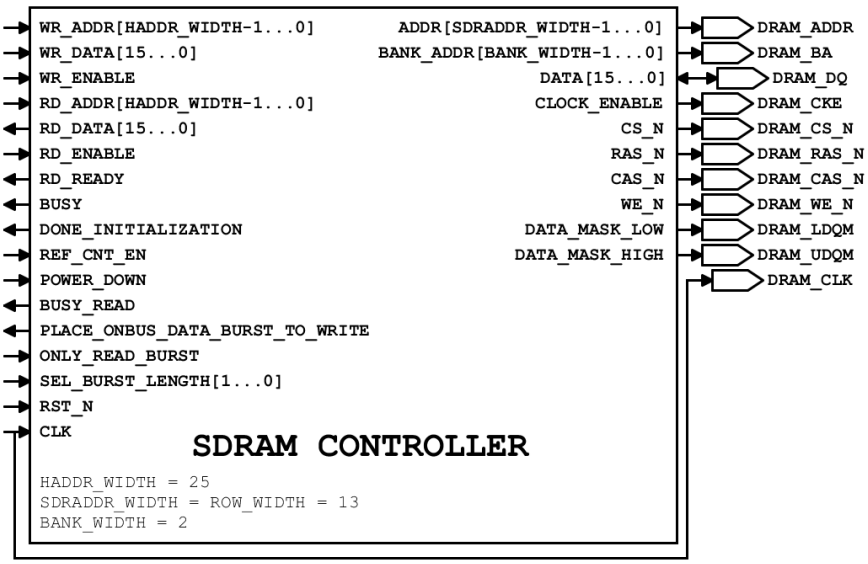


Figure 2.5: Top level enhanced SDRAM controller

has elapsed, otherwise data integrity will not be guaranteed.
Then the burst feature signals have been provided: the input signal *ONLY_READ_BURST* programs the *Mode Register* to work either with fixed-length bursts readings and single access writings or with burst support for both readings and writings. The input signal *SEL_BURST_LENGTH* selects the length of the burst among 1 word, 2, 4 or 8 words.
Finally, the output signal *PLACE_ONBUS_DATA_BURST_TO_WRITE* is an acknowledgement provided by the controller when the user wants to perform a burst writing: when this signal is asserted by the controller, in the successive clock cycles the memory is ready to receive the input data to write in and so the user has to place the data on the bus as many clock cycles as the number of the words to write given by the chosen burst length. This is necessary to respect the memory timing parameters from issuing the *ACTIVE* command (providing bank and row addresses) to the *WRITE* one (providing the burst starting column address). In the following three Figures 2.6, 2.7 and 2.8 are showed some timings extracts of how single reading and fixed length of four burst readings and writing respectively are handled, as a visual explanation of what just described.

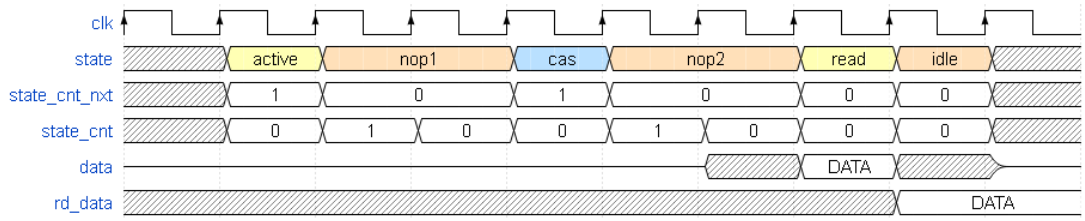


Figure 2.6: Single reading timing diagram

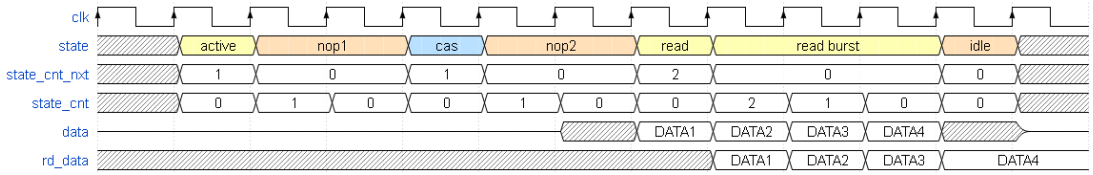


Figure 2.7: Burst reading of length of four timing diagram

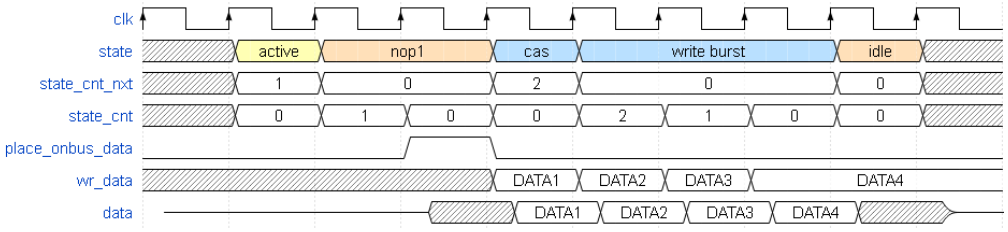


Figure 2.8: Burst writing of length of four timing diagram

As mentioned before, NOP states are added to respect the timing constraints fixed by the memory. The number of NOPs are controlled through two auxiliary counter signals that act like *present state* and *next state* signals: whenever a number of NOPs has to be added according to the delay, in clock cycles, to wait for from the current state the memory controller is and the next state, *state_cnt_nxt* is loaded in *state_cnt* and the last one is decremented till arrives to zero; at this precise instant, the memory controller changes state from the NOP one to the next predicted one. The time delay that fixes the number of NOPs depends on the delay in cycles between two successive "active" commands and they are given in the list of timing parameters reported in section 2.1.

Chapter 3

Controller architecture modification

Once the SDRAM controller has been verified to work, the surrounding layer that is able to command the core controller is designed according to the intended purpose. First of all, the *Row Refresh Machine* (RRM) has been realized by scratch: this is the modification applied to a standard memory controller where the *Auto-Refresh* is thrown away and a "retention times-aware" refresh is applied to each single row. In order to determine at which rate each row should be refreshed, the initial profiling is needed: so, before performing the characterization, the RRM will be showed to refresh the rows at a random rate first, simply to understand how this machine works and how the standard *RAS-only refresh* can be emulated; then, a "true" characterization will be done and the RRM will refresh the rows according to the retention times distribution.
But how to store the retention times in order to perform row-by-row refreshes? To do that, two SRAMs are used to store the retention times in the form of thresholds as multiples of 64 ms: the first SRAM stores the thresholds corresponding to the retention times of each row, the second the related counters. In every refresh window tREF of 64 ms, whenever a row is issued then its counter is decremented: when each counter elapses, the controller issues a *RAS-only refresh* as a sequence of an ACTIVE (ACT) command followed by a PRECHARGE (PRE) command (observing timing constraints correctly), provided that the bitlines are precharged first. In this way the row is refreshed to prevent any corruption. Practically, it is similar to what happens when reading from a memory location: the row is activated, placed in the *row buffer* and then written back again to restore the contents. So from now on, the operation issued by the controller when a row at a particular address needs to be refreshed will be called a *dummy reading*: hence, the name of the signal is *RD_DUMMY* and the corresponding address sent is *RD_DUMMY_ADDR*. The state diagram of the SDRAM controller when handling *RAS-only refreshes* is shown

in Figure 3.1.

Whenever a dummy reading is asserted to a given row, the corresponding bank

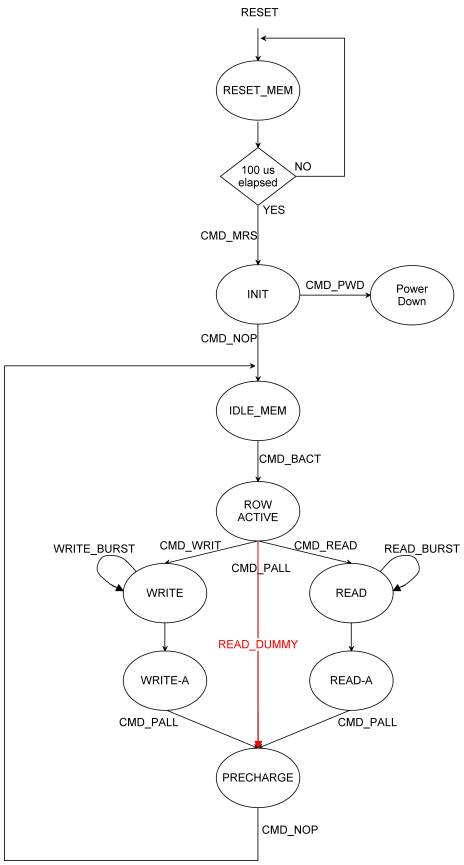


Figure 3.1: Modified SDRAM controller state diagram

will not be available for readings or writings for the entire duration of activation-precharge sequence tRAS + tRP. Then, the profiling will follow with an exhaustive analysis of the best data pattern for the used SDRAM.
Finally further optimizations will be presented, together with the handling of a challenging problem that is the strong temperature dependence of the retention times distribution.

3.1 Row Refresh Machine

The *Row Refresh Machine* (RRM) represents the finite state machine that works in parallel with the normal operations of the SDRAM and that issues refreshes

whenever the counters elapse. The fact that this machine works in parallel to the memory means that the latency of a memory operation is not affected, but the average response of the controller to a user service request could be heavily reduced for the same reasons that have led to realize such a kind of controller architecture. Regarding the memory used to test the realized controller, since there are 32768 rows and each one has to be issued every 64 ms to avoid to lose data, at 140 MHz each row has to be issued every tREFI that can be computed using equation 1.1:

$$tREFI = \frac{tREF}{R} = \frac{64ms}{32768} = 1.95\,\mu s \tag{3.1}$$

So, at this frequency, there are about 273 clock cycles between each row issuing, where a possible refresh command can be sent in case of elapsed counter. That would be more expensive than actual *Auto-Refresh* but we will see that the row-by-row refresh won't follow after each tREFI of 1.95 µs, meaning that the number of *row misses* in an application, that uses such a memory in its main memory system, will be highly reduced. So in order to emulate the refresh cycles (tRFC), where in this case must check the value of the current counter and eventually assert a dummy reading or simply decrement a counter, a window of clock cycles is provided for each row for performing these operations. Moreover, another optimization has been applied when performing this type of refreshes: the activation of a row for a request issued by the user also restores the content at the end of the operation cycles; so whenever a reading or a writing is performed to any location, the retention time counter of that corresponding row is reset again to its maximum initial threshold value minus one, without letting decrease it towards zero. This is the reason for which the window of clock cycles emulating tRFC is needed, because the requested operation uses as well the current counters memory to reset the threshold and this would cause a conflict with the normal check for a dummy reading refresh. But if any requests are sent while the RRM is flowing through this cycles window, some of them could be lost and the controller can lose the opportunity to reset the counters and provide further benefits in refreshes reduction. In order to allow the controller to take advantage from all the possible accesses requested to the memory, a FIFO is added to the architecture. In this FIFO all the access addresses are saved when RRM is not available to reset the counters and, when it is not busy, used by the controller to reset them. The bits width of the addresses to save inside the queue can be obtained from Figure 2.5 and it is equal to:

$$\text{HADDR\_WIDTH} = \text{BANK\_WIDTH} + \text{ROW\_WIDTH} = 15 \text{ bits}$$

and that corresponds to the address number of bits used to point to the two SRAMs storing thresholds and counters.
For what concerns the depth of the FIFO these considerations have been taken into account: first of all, a worst case single word reading or writing operation takes 7

clock cycles and during this period no further accesses are allowed since the memory is busy; secondly, considering the window of clock cycles used by the controller to check for a dummy reading and a possible superposition with a requested access, a FIFO of depth 2 would be sufficient. To keep relaxed constraints, a FIFO of depth 4 has been used. A simplified representation of the actual component used in the VHDL architecture is reported in Figure 3.2.

The enable to write signal *ACTIVATED_ROW_FOR_ACCESS* and the input

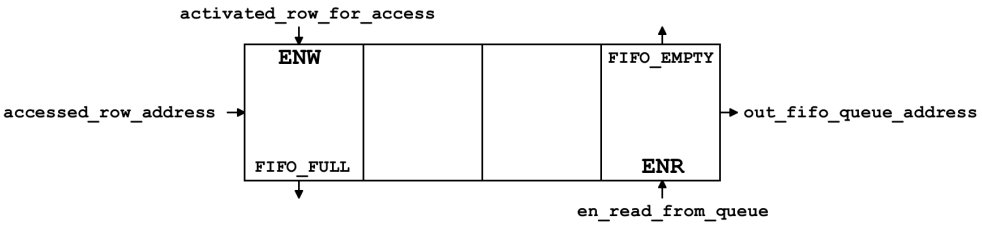


Figure 3.2: Accesses handling FIFO

data to write signal *ACCESSED_ROW_ADDRESS* are provided by the SDRAM itself through the controller whenever an access has been detected: the *FIFO_FULL* signal automatically detects if the queue is full, but it will never happen due to the fixed relaxed capacity. The enable to read signal *EN_READ_FROM_QUEUE* is sent by the controller whenever it is not busy in checking for a dummy reading: the controller checks for accesses to handle by using the *FIFO_EMPTY* signal and, if any present, serves the request by asserting the enable to read to *ENR*.

Whenever an access is handled, the reason for the reset to its maximum initial threshold minus one stands in having to cover the "dead" period left during the remaining time of the 64 ms refresh cycle issuing the successive rows, before returning to the same row again after the refresh window. And, hence, that is the reason for which it is necessary to save the initial thresholds somewhere: the first SRAM storage is necessary, then, to restore back the initial value to the counter whenever either an access request operation or a dummy reading refresh is performed.

Supposing that the profiling has been carried out and the thresholds have been obtained, the ASM chart of the *Row Refresh Machine* is reported in Figure 3.3.

The final RRM ASM chart will be showed after the characterization where the complete machine will be presented.

After the profiling step the controller enters in the *START_USR_REFRESH* state where, if in the *CYCLE_REF_CYCLES* window, checks the value of the current counter and either executes a dummy reading or decrements the value of the counter and writes back in the same location of the SRAM; otherwise if not in the window of checking for a refresh cycle, in the case of an access request issued by the user (FIFO accesses not empty), the counter is reset to its maximum initial value minus one as mentioned before. The need for 4 states of *RESET_CNT_FOR_ACCi* here

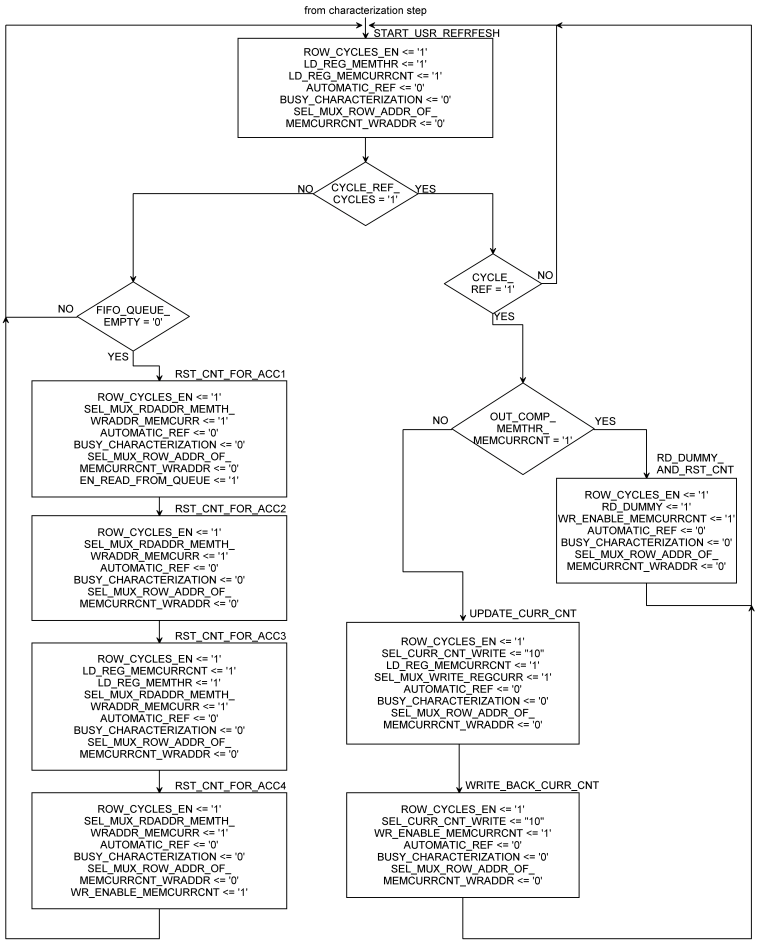


Figure 3.3: *Row Refresh Machine* (RRM) ASM chart

is due to the fact that the IP RAM2 megafunction used in Quartus® synthesizer for the two SRAMs has registered both write and read address inputs and also both write enable signal and input data to write; moreover it is necessary to consider the latency of one clock cycle for the FIFO accesses, so in order to perform correctly the reset of the counters without entering in conflict with a counter decrement, 4 states in sequence are needed.
For what concerns the datapath of this initial version of the controller architecture, it is composed by the two memories of thresholds and counters, an integer counter of about 273 cycles for each row described before (configurable according to the

number of rows and to the clock frequency) and a counter of row addresses. Finally, some multiplexers to correctly select the addresses to the memories and output registers to save the contents. The complete structure of the datapath will be reported in successive sections.
As proof of working, some simulations of the main features of the RRM have been performed and they will be showed in the following figures. Just note that the characterization has not been performed yet, hence the testing thresholds have been set all to 4, meaning that all the rows have a retention time equal to 4 * 64 ms = 256 ms before being all refreshed in the fifth refresh cycle window. Moreover, to simplify the computations and get an integer clock period value, the simulations have been performed with a clock frequency of 100 MHz instead of 140 MHz then used in the final implementation, meaning that in these simulations each row has 195 clock cycles between each tREFI: nothing changes, it is only a matter of simplification.

3.2 Simulations and comments

In all the figures that follow, extracted from the simulations performed by using ModelSim® simulator, only the most important of many signals will be showed to provide a better clarity of how things are going on.
In Figure 3.4, a piece of the starting of row refresh is showed. This will coincide with the end of the characterization where the thresholds and current counters SRAMs are filled; in this simple simulation, this coincides with the end of the SRAMs filling with a fixed threshold for each row (4, as said previously). The *CYCLE_REF_CYCLES* signal defines the clock cycles refresh window (like tRFC) and the *CYCLE_REF* signal defines the exact clock cycle where the current counter is checked for a possible dummy reading, like issuing a refresh command every tREFI in *Auto-Refresh*: since the counter is equal to 4 at the beginning, it is decremented and written back again in the same location, as visible on the current counter memory output signal Q at the bottom.

In Figure 3.5 there is another extract of the row-by-row refresh similar to the previous one, this time on bank address 1 and row address 0, during the first 64 ms and exactly after a time equal to tREFI = 1.95 µs as obtained in equation 3.1. One can observe again the refresh window covers the needed clock cycles taking into account also a possible request operation issued by the user just before this window and in agreement with the ASM chart shown in Figure 3.3, preventing so any conflict with the state machine in using counters memory.

In treating the bank and row addresses as a unique row address array, this row at bank address 1 and row address 0 corresponds to row address 8192.
Next in Figure 3.6 is indeed showed what happens when an access request is issued to bank address 0 and row address 1 after about 192 ms and the state machine is not flowing through the refresh window: as mentioned before, in this case the counter in the location corresponding to row 1 is reset to the initial threshold value

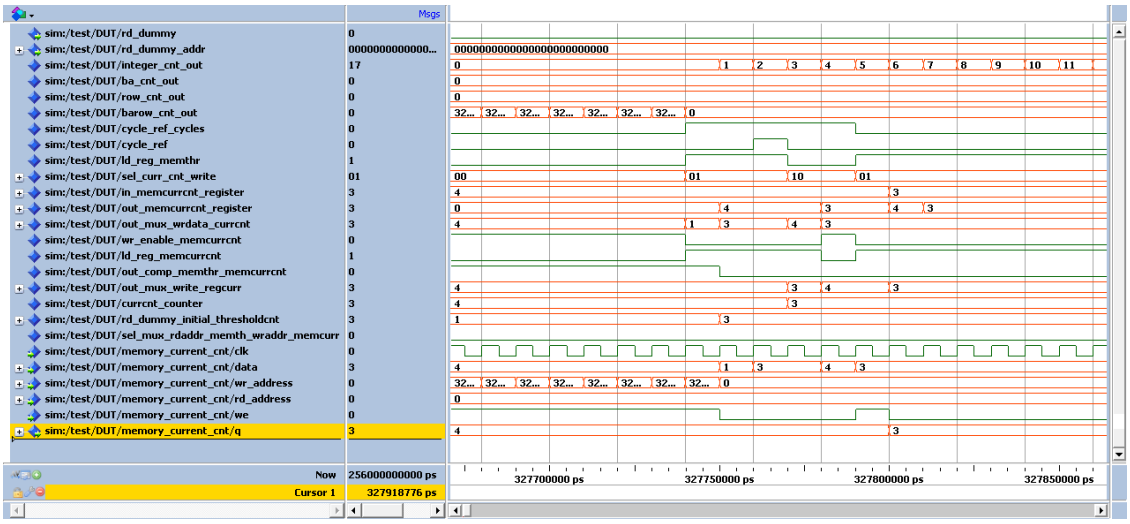


Figure 3.4: Starting of row refresh after characterization

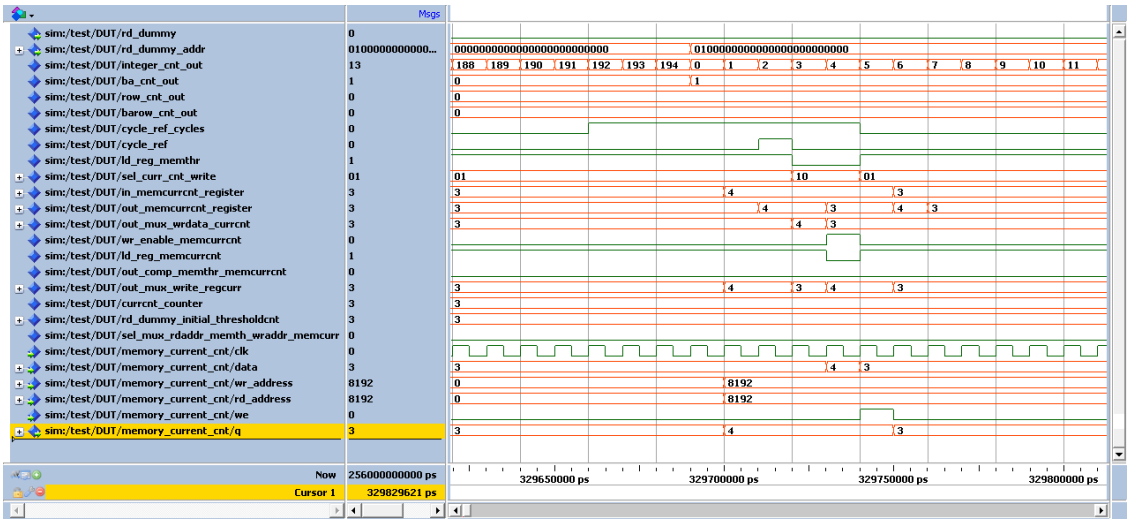


Figure 3.5: Updating and writing back of the current counter on bank address 1 and row address 0

minus one to take into account the current 64 ms tREF cycle ending. The value is taken from the thresholds memory and correctly written in the counters memory: as visible, in fact, the highlighted write enable signal (*WE*) of that memory is asserted and the write address signal (*WR_ADDRESS*) changes temporarily from current one (8189) to the address issued for a reading (1). All is handled through the used FIFO to serve access requests to prevent losing them whenever the state machine is refreshing. Subsequently it will be possible to see how this will affect the dummy

reading on that row with respect to the other rows having initially fixed, indeed, all the thresholds equal.

A similar situation is showed in Figure 3.7, where now the access is issued to row

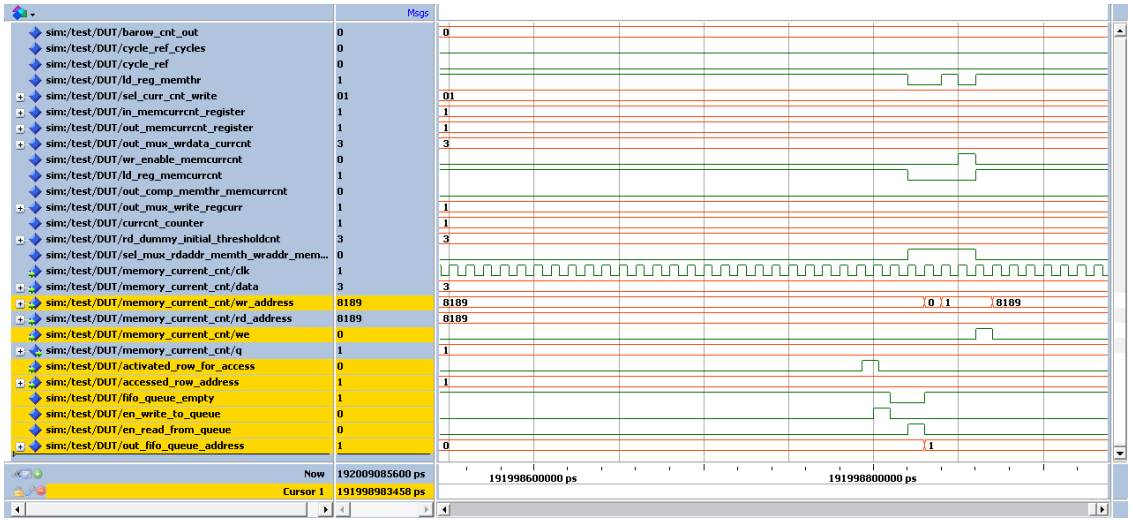


Figure 3.6: Access request issued by the user to bank address 0 and row address 1

address 5.

Having set all the thresholds to 4, after 256 ms (four refresh cycles tREF) one can

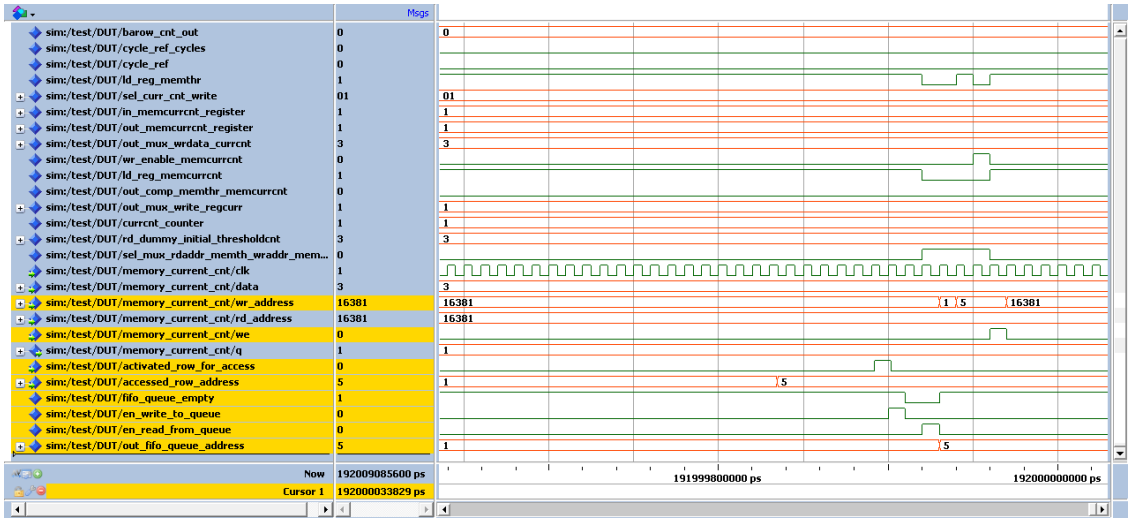


Figure 3.7: Access request issued by the user to bank address 0 and row address 5

expect that there will be dummy reading assertions for all the row addresses in the same tREF cycle, except for row addresses 1 and 5 due to the previously showed

access requests sent by the user which caused their counters to reset. In Figure 3.8, a dummy reading operation is performed on row address 0 after 256 ms.

A dummy reading is performed on this address and this is specified in the first most

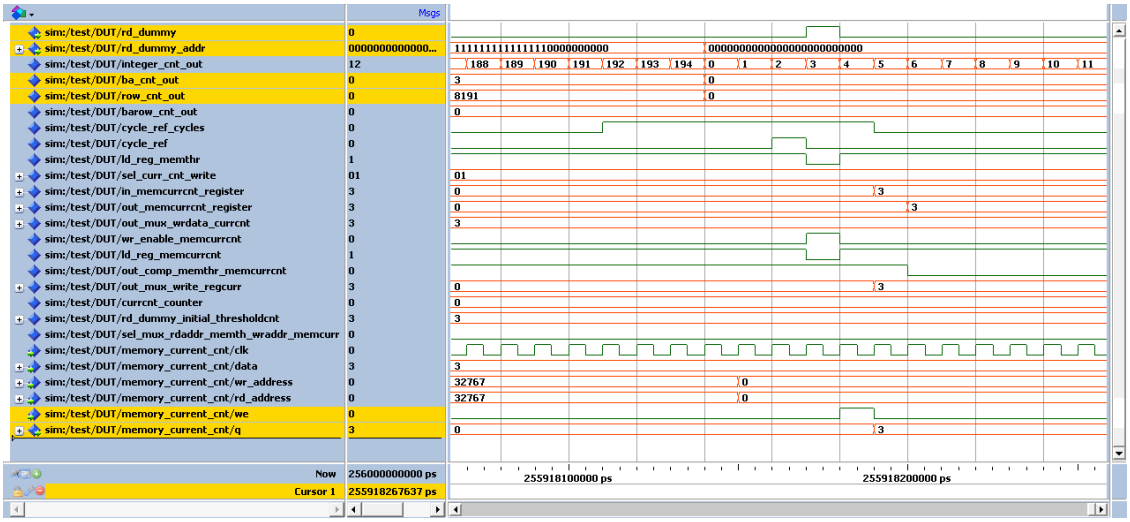


Figure 3.8: Dummy reading assertion on bank address 0 and row address 0

significant 15 bits of *RD_DUMMY_ADDR* signal (the address must be always on 25 bits, as required by the SDRAM, so the least significant 10 bits of this signal, standing for the column address, are set all to 0 to have a fast access to the first column when sending a refresh command). As a result, a write enable is asserted on the counters memory and the initial value of the threshold minus one is restored. At this point an example on what happens on row addresses 1 and 5 is showed in Figures 3.9 and 3.10.

It will explained what happens only on row address 1 as it is the same for row address 5. At the starting of the third 64 ms refresh cycle the counter is decremented from 2 to 1 but at the end of the same cycle a reading operation is issued to the same address, resetting so the value of the counter to its initial threshold minus one, that is 3. At the last of the four 64 ms refresh cycles, the counter is decremented from 3 to 2. Then, in Figure 3.9 we are in the fifth 64 ms refresh cycle and the counter is correctly decremented from 2 to 1. Just note that the fourth refresh cycle doesn't terminate exactly at 256 ms, due to conversions with respect to the used clock frequency, but it terminates at about 255.918 ms (it is always better to stay below the 64 ms or its multiples for a refresh cycle when you get approximations from the computation, to avoid bit failures only due to bad conversions). In Figure 3.10 it is possible to see an overview of what happens: the *RD_DUMMY* signal is not asserted in the fifth refresh cycle on row address 5 due to the previous access, so its rate of refreshing has changed and it has been postponed with respect to the shown adjacent rows.

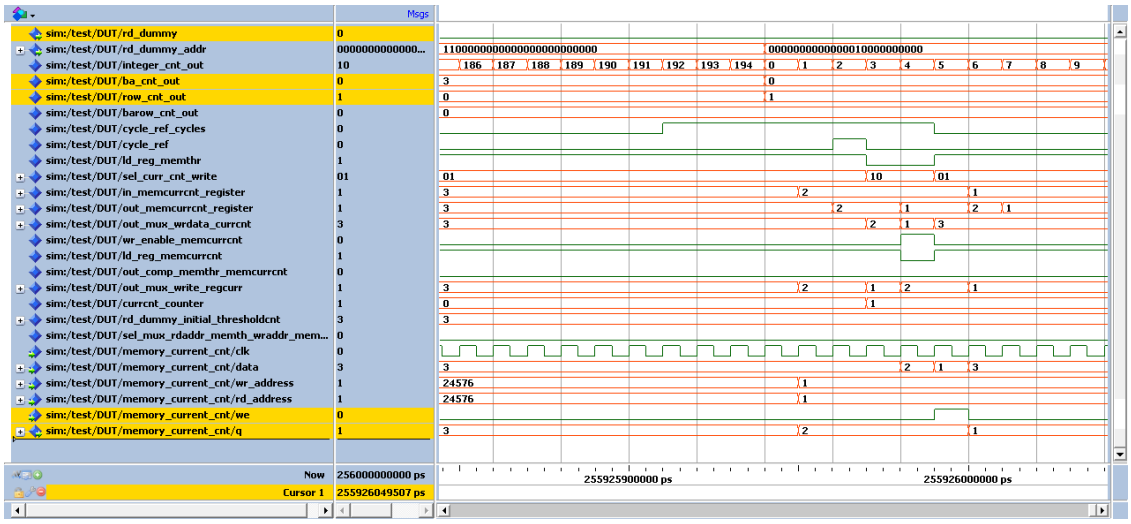


Figure 3.9: No dummy reading assertion on bank address 0 and row address 1

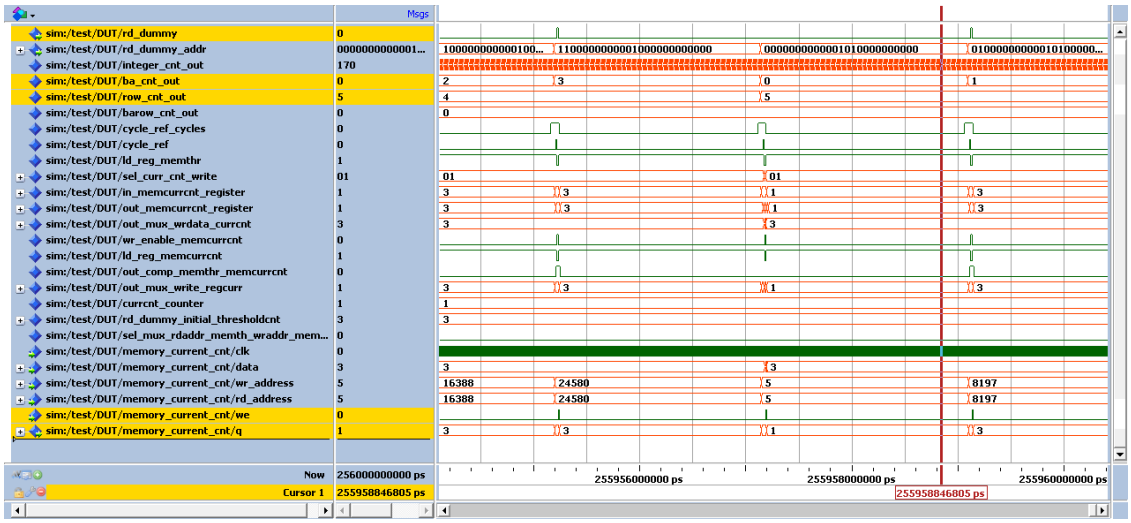


Figure 3.10: No dummy reading assertion on bank address 0 and row address 5

In Figure 3.11 there is an example of the benefits given by the usage of a FIFO to handle the access requests. If a request came exactly in the refresh window given by *CYCLE_REF_CYCLES*, the controller would not be able to reset the related counter when serving the access to that row. The used FIFO allows to always take advantage of accesses and, as soon as possible, to serve the request and reset the associated counter: in the figure, a pending request is issued at row address 8199 and, after the refresh cycles window, the value is updated by temporarily changing the *WR_ADDRESS* signal of the counters memory and asserting its write enable

WE signal.

As a further consideration in these showed figures the accesses to the rows, as

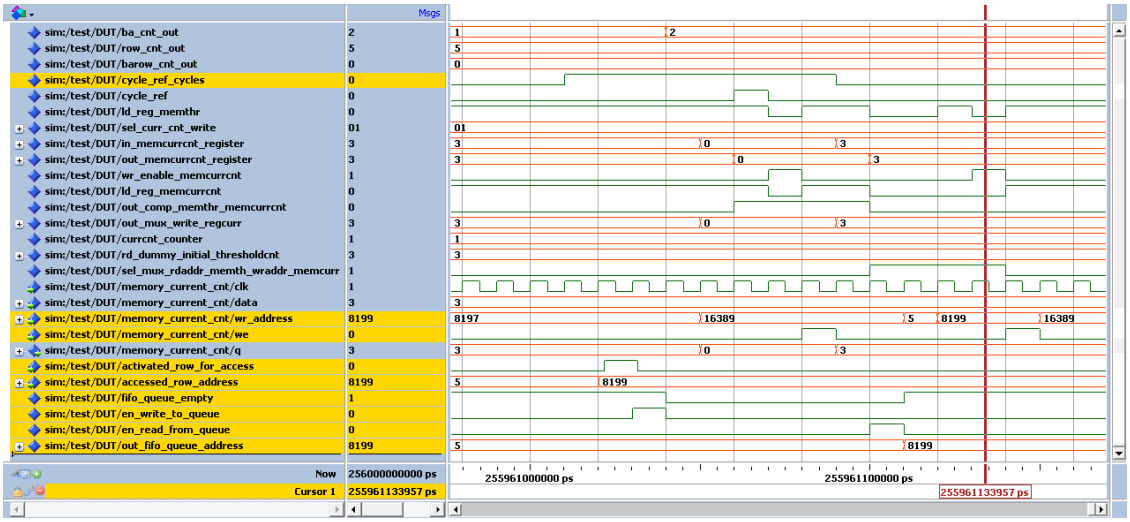


Figure 3.11: Postponed access on bank address 1 and row address 7

separated bank addresses and row addresses, are performed in an interleaved way, that means that a row per bank is accessed before returning back to the first bank, as in the *Auto-Refresh* feature. This is important to avoid starvation on the same bank when refreshing and so to reduce accesses overhead to any bank. Even during the characterization the accesses will be performed in this manner, to stress all the banks in the same way. For completeness, the equation 1.2 that says how many rows per bank have to be accessed during a refresh command, given tREFI equal to 7.8 µs fixed by JEDEC [19] standard, the number of rows per bank R (8192 in this case) and the refresh cycle period tREF equal to 64 ms, is:

$$r = \left\lceil \frac{tREFI * R}{tREF} \right\rceil \simeq 1$$

So it means that, in parallel, four rows of four different banks are refreshed every tREFI. In this implementation, due to a row-by-row refresh, a single row will be accessed during a refresh command (dummy reading) but however the rows will be interleaved subsequently. This is, of course, time consuming due to the non-parallelized bank accesses provided by the ACT command with respect to *Auto-Refresh*, but the benefits will come just after the retention times characterization results.

3.3 Dummy reading requests handling

Every time a row needs to be refreshed, the modified controller now issues a *RAS-only refresh* to that row. As for *Auto-Refresh* or for a normal request to read or write sent by the user, all these operations cannot be interrupted till they end. This means that if the memory starts serving a reading request sent by the user from a memory location and the *Auto-Refresh* monitor counter has elapsed, the reading operation terminates and the *Auto-Refresh* is so optimized that it starts refreshing just after the termination of the request sent by the user. Normally, the refresh command has the priority with respect to normal operations however respecting typical FCFS scheduling, since data integrity has to be guaranteed.
But in this case the refresh commands are sent with a sequence of activation and precharge, and so it is not so different from a normal accessing request. Clearly, the priority is given to this sequence of commands to ensure data integrity too, but what happens if a dummy reading comes exactly during an already started operation? As it is treated as a request too, it would be lost. For this reason, a dummy readings FIFO has been provided as well to avoid losing the refresh commands in such a kind of situation. This queue has been placed inside the SDRAM core controller to allow to be synthesized near this one, since the outcome is the row address to be refreshed.
What about the depth? The FIFO implementation used also for accesses handling

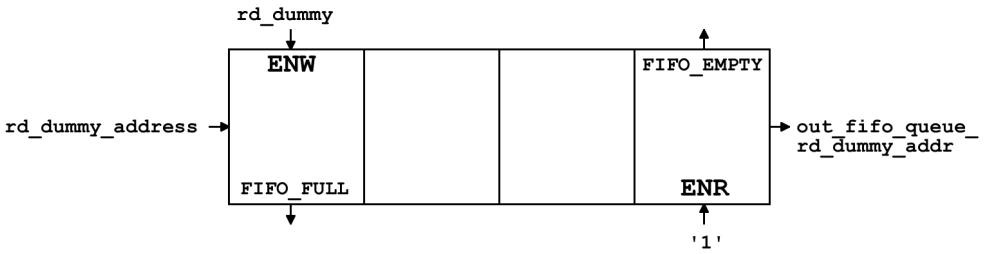


Figure 3.12: Dummy readings handling FIFO

is configurable in terms of both bits width and depth and is reported in Figure 3.12. The bit width is on 25 bits, since this address has to be provided to the the input address bus of the SDRAM, even if only the most significant 15 bits are needed during an activation command and that correspond to bank address and row address. The depth is fixed to 1 location since a dummy reading (in the worst case of refreshes every 64 ms for each row) comes every tREFI = 1.95 µs. This means that if a dummy reading comes exactly when a request is being taken in charge by the SDRAM, it is saved inside the FIFO and handled just after the end of the request, because this refresh sequence of commands has been given the priority over the other operations: so when the memory terminates the request and returns idle,

if the dummy reading FIFO is *full*, then a refresh is immediately issued to guarantee the integrity of the row. In the worst case situation that a dummy reading comes exactly at the beginning of the request, the refresh will be postponed at maximum for a duration of a longest burst reading or writing operation allowed, that would take about 100 ns: compared to the retention times sustained by the rows, this delay is expected to not produce data corruption of the row contents. This feature is not showed in the simulations when asserting a dummy reading, because added in a second moment, but it has been tested to work correctly.

Chapter 4

Retention times profiling

4.1 Test structure

Once the *Row Refresh Machine* (RRM) has been verified to work, the next step is to fill the thresholds and current counters memories with actual retention times values: so the characterization step is needed. In performing this task, similar steps of work [2] have been followed. First of all, the memory locations are filled with some data, then some time is waited for, eventually the memory locations are read and compared with which had been written to see if that location is able to retain the written data for that time without being refreshed. The overall general procedure of the test application is shown in Figure 4.1.

How to choose the best data pattern that allows to find as many bit failures

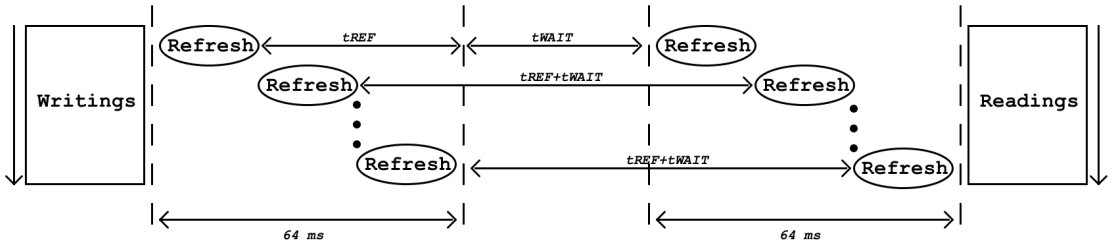


Figure 4.1: Test application structure [2]

as possible? As mentioned in work [2], there are several factors that make the profiling unable to provide exact and confident results, mainly due to *Data Pattern Dependence* and *Variable Retention Time* effects. But if some precautions are taken in performing this operation, the results could be quite acceptable. The retention time of a given cell depends on the value that is stored in the same cell and in the nearest ones: bitlines coupling and effects of crosstalk are the main responsible for such variations. In this experimental study was found that static patterns as all 1s

or all 0s are able to find out not more than 15% of all the actual weak cells. This however depends on the complexity of the memory architecture where the *Data Pattern Dependence* has a different effect regarding crosstalk between cells. Then, from device to device, the behavior could be different in profiling the retention times with static or dynamic and unpredictable patterns.
Hence, an experimental study has been conducted first, finding which is the best pattern that allows to obtain the best coverage of bit failures in the used SDRAM: the result, as said, is not expected to be the same for the all existing memories so the controller architecture has been configured to choose among selected patterns. Moreover, as suggested in the work, repeating the experiments in rounds separated each other by the same time interval helps to free the results from DPD in the beginning and then from the effects of temperature, that is kept pretty much constant, are limited too through a small simulation time. From previous work simulations, dynamic random data seem to be the ones that typically allow the highest coverage of bit failures with the predominant presence of VRT that causes big troubles in the retention times profiling. So, in order to provide which is the best pattern for the SDRAM used in the designed architecture, exhaustive simulations have been conducted to provide quite confident results: both static and quasi dynamic patterns have been tested, such as all 1s, all 0s, alternated 0 and 1 (checkerboard) and pseudo-random. Each pattern, moreover, has been followed by its complement to take into account the behavior of each single bit cell as *true-cell* and *anti-cell* due to the presence of differential sense amplifiers. Then, the simulations have been performed in 5 consecutive and repeated rounds to free from the effects of DPD. The reason for a pseudo-random pattern, although it is static, is that it can be reproduced (and that is the reason for which it has been used) and it is the one that is likely to be similar to a dynamic and unpredictable pattern when applied to the SDRAM. The test has been performed for 8 retention times indicated on the horizontal axis, with the same procedure in Figure 4.1, and the obtained results of the worst case and average number of rows coverage for all the described patterns are reported in a bar graph in Figure 4.2. The number of rows on vertical axis indicates the number of rows that are able to sustain that retention time and that have failed the test on the next retention time. This doesn't identify how many cells show a bit flip during the test, because the characterization fixes the retention time for a given row whenever the first cell of that row fails the test. So the weakest cell fixes the retention time for the entire row and then all the other cells are skipped when a bit flip is found out. The average and maximum room temperature along the simulation have been annotated: the maximum temperature differs from the average one only for 0.13 °C and since a temperature sensor with an absolute uncertainty of ± 0.5 °C has been used, it is possible to conclude that during the simulation the estimated room temperature stayed quite the same.

As visible from figure, the pseudo-random pattern is one of the best patterns together with all 1s and all 0s that guarantee a good coverage of bit failures at lower

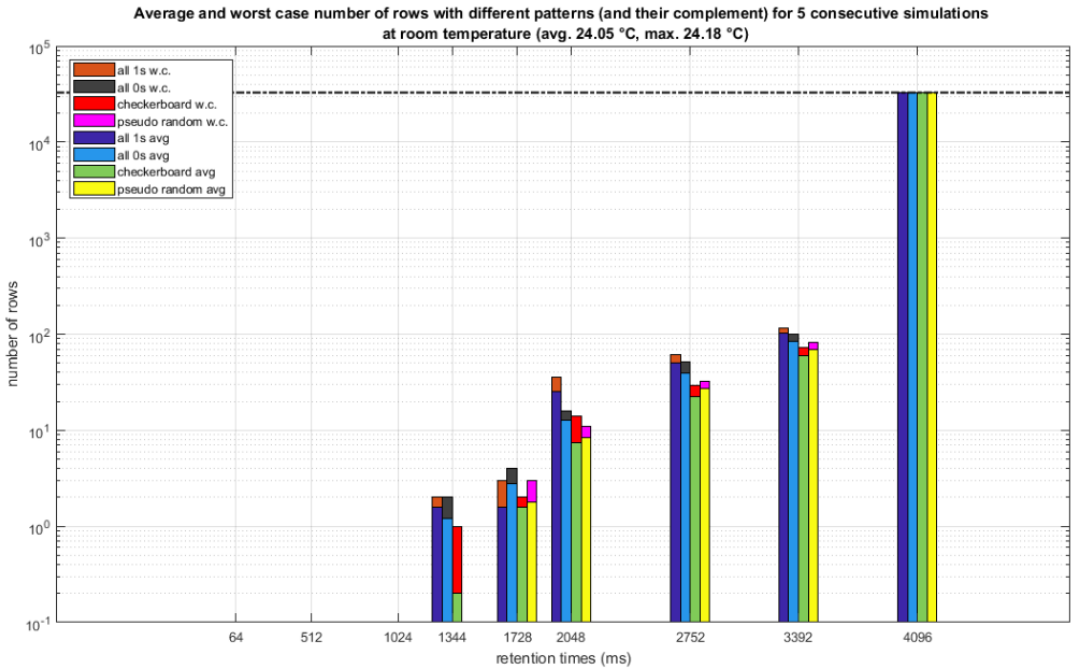


Figure 4.2: Average and worst case number of rows with different patterns analysis

retention times than the maximum analyzed one (4.096 s): so it will be used for the exhaustive temperature dependent simulations that will follow. The fact that also static patterns like all 1s and all 0s provide a good coverage stands in the reduced density of cells of the used SDRAM: with only 64 MB of capacity, the memory is less affected by the problems described before and the patterns show more or less the same behavior. In previous work [2], instead, tests on different DRAMs of 1 GB upwards and from different vendors have been performed and the results showed that, typically, the best pattern in terms of bit failures coverage is the random one. However, the architecture is suited to perform the profiling with a data pattern, among the four introduced, that the user can choose according to its DRAM.
So, due to its capability to regenerate the same data, a linear feedback shift register (LFSR) has been used to produce pseudo-random data: in particular, we need to generate 1024 data of 16 bits each for each of the 32768 rows. Then, in order to provide different data on every row, two 16 bits linear feedback shift registers have been used: one only on the first column of every row that provides the *input seed* on the other LFSR that provides 16 bits data to write inside the 1024 columns of each row. The column LFSR, indeed, will be used during the comparison step to regenerate the written data and compare them with the ones read from the SDRAM: a maximal-length polynome has been used for the LFSRs to cover all possible cases on the used number of bits. During the writing procedure the *Auto-Refresh* is

enabled to avoid the corruption of data. At this point, refresh is kept enabled for 64 ms while SDRAM is in idle state and then it is disabled for a chosen period, in Figure 4.1 tWAIT, keeping the SDRAM in idle. This helps to understand if a given row is able to retain its data without being refreshed. After this delay, the *Auto-Refresh* is enabled again for 64 ms: so, in total, a tREF + tWAIT delay has elapsed since last refresh for every single row. Then the readings step can start: the characterization machine issues reading operations to each location and performs comparisons with the output of the column LFSR; if the comparison is successful, it means that the location in that given row can retain the data for that analyzed time without being refreshed, otherwise it cannot. The comparison is done on 16 bits data, according to the memory data bus parallelism for the used configuration, and the worst case cell fixes the retention time for the entire row: whenever a cell content flips, it means that the entire row has to be refreshed at a higher rate than the current period given by tREF + tWAIT to retain the data and the test on the remained columns of the same row are skipped. As for writings step, also the readings step is performed by keeping enabled the *Auto-Refresh* to avoid corruptions during the procedure.
Next is to choose the values of tWAITs to use in these successive tests as done in Figure 4.2. Now, considering that the profiling takes time to be performed, it has been decided to choose only four tWAITs for the final retention times characterization and execute a fixed number of 10 rounds on these: so, in sequence, the scheme in Figure 4.1 will be repeated four times and consequently fixing, for each row, the actual refresh time. In the experimental studies conducted at a temperature of 45 °C [2], none of the cells showed a retention time less than 1.5 s and that is considerable at such a high temperature. So taking into account that the tests that will be performed will never overcome such value at room temperature, the four total delays (tREF + tWAIT) chosen for repeated simulations are: 512 ms, 1.024 s, 2.048 s and 4.096 s. So the values of tWAITs are 448 ms, 960 ms, 1.984 s and 4.032 s that, in terms of multiples of 64 ms tREF, correspond to 7*tREF, 15*tREF, 31*tREF and 63*tREF respectively. Then, for thresholds and current counters memories, the values that will be stored are:

1. tREF = 64 ms $\longrightarrow$ threshold = 1
2. tREF + tWAIT = 512 ms $\longrightarrow$ threshold = 8
3. tREF + tWAIT = 1.024 s $\longrightarrow$ threshold = 16
4. tREF + tWAIT = 2.048 s $\longrightarrow$ threshold = 32
5. tREF + tWAIT = 4.096 s $\longrightarrow$ threshold = 64

The profiling step is executed at power-on, so the user could choose its own thresholds according to the room temperature considering the retention times

distribution in Figure 1.2: the thresholds are fixed on 8 bits, so the user can select the four retention times of the characterization from 1 (64 ms) to 256 (16.384 s) as desired. The pattern for profiling, instead, is set by default to the quasi dynamic pseudo-random one, for the reasons described before regarding the complexity of new chip devices.
In computing the retention times for each row, the first threshold equal to 1 has to be considered too. This is necessary in the first cycle where the retention time of 512 ms the memory locations are tested for. In the very first cycle, if the comparison doesn't succeed, then that row is not able to retain data for 512 ms and so it will be refreshed at 64 ms, assigning to that a threshold equal to 1. For the successive cycles of the four tested retention times, the reference will always be the previous retention time analyzed: if the comparison is successful update with the new retention time threshold, otherwise keep unchanged the previous one.

4.2 Characterization state machine

The ASM chart of the characterization machine used to perform the retention times profiling is reported in Figures 4.3, 4.4 and 4.5.

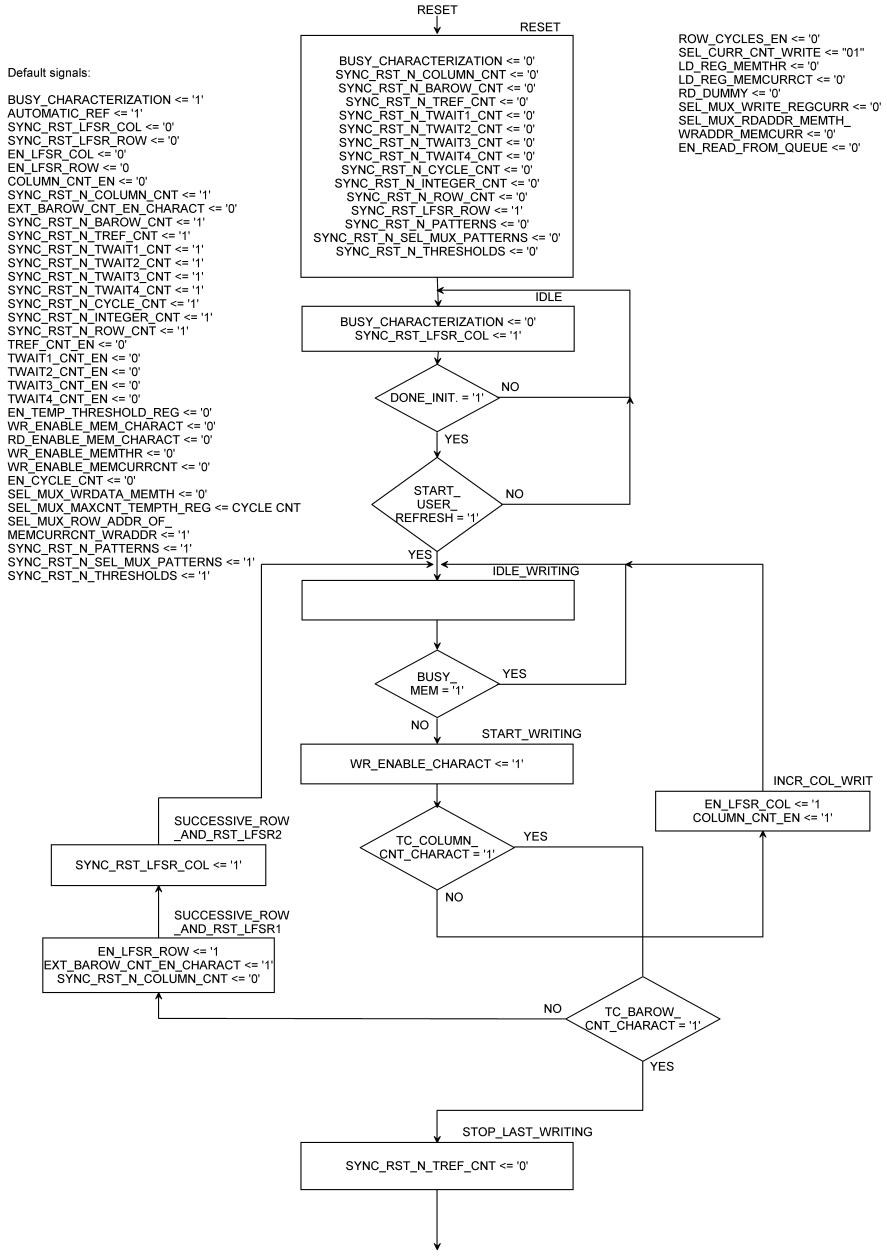


Figure 4.3: Characterization machine ASM chart - Writings step

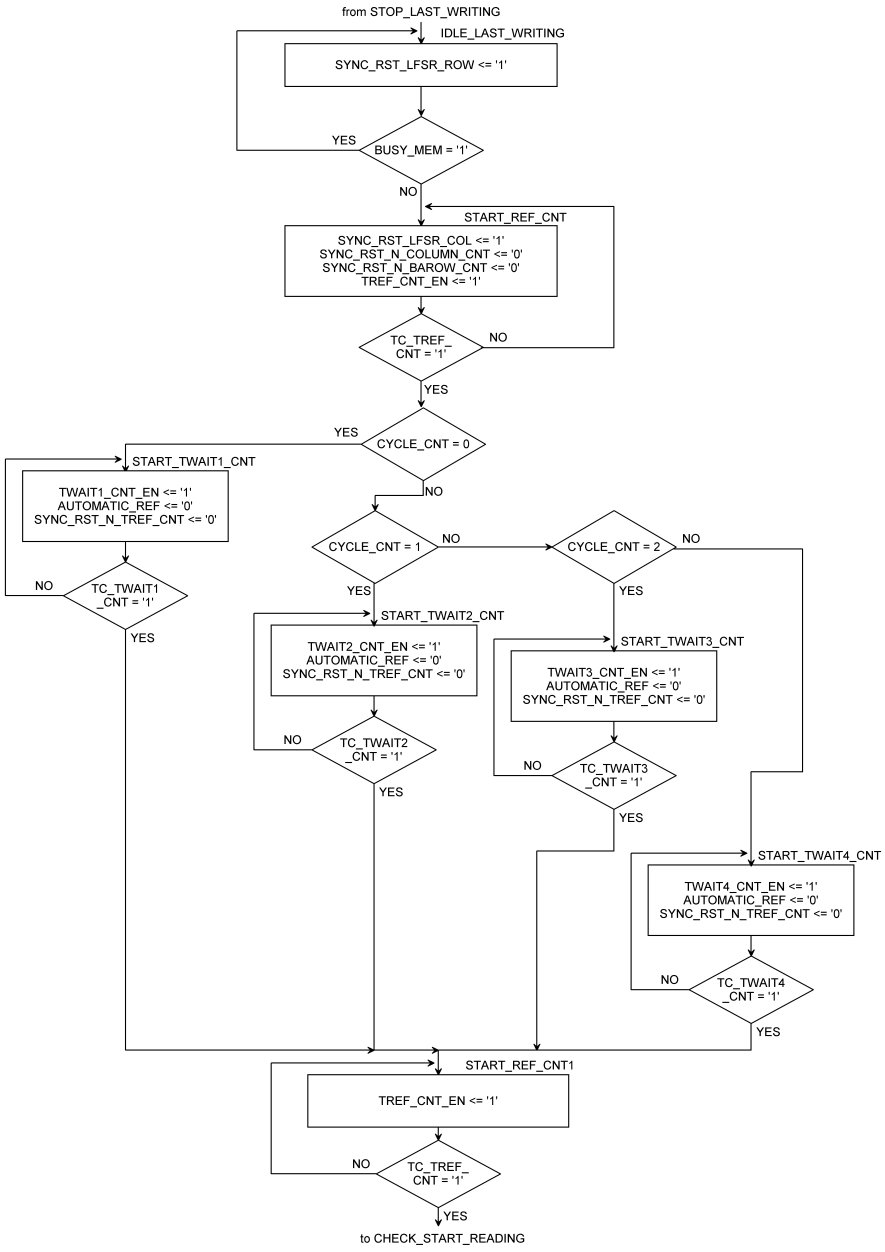


Figure 4.4: Characterization machine ASM chart - tREF + tWAIT step

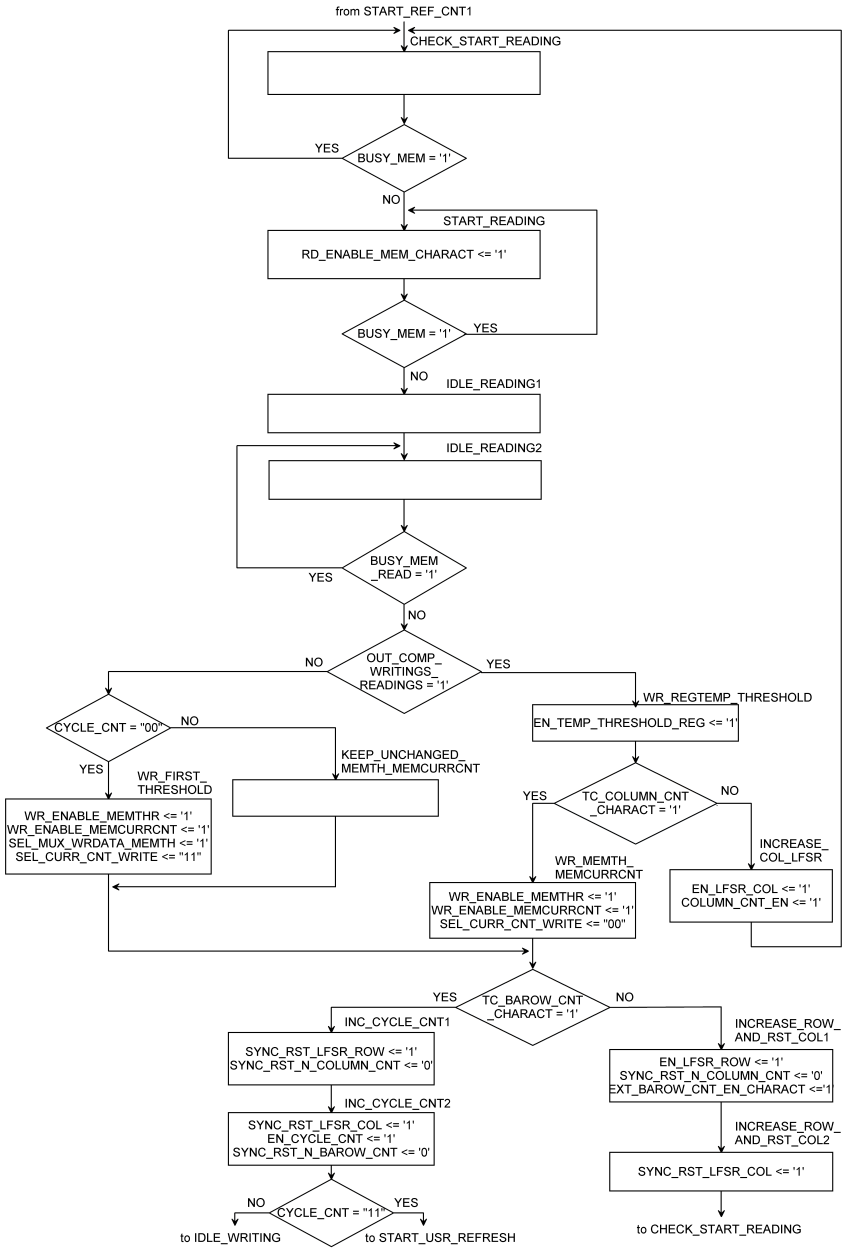


Figure 4.5: Characterization machine ASM chart - Readings-comparisons step

For completeness, some realized timings of the most important steps of the procedure (writings and readings-comparisons steps above all) are showed in Figures 4.6, 4.7 and 4.8.

In Figure 4.6 the writings characterization step is reported. Since the profiling

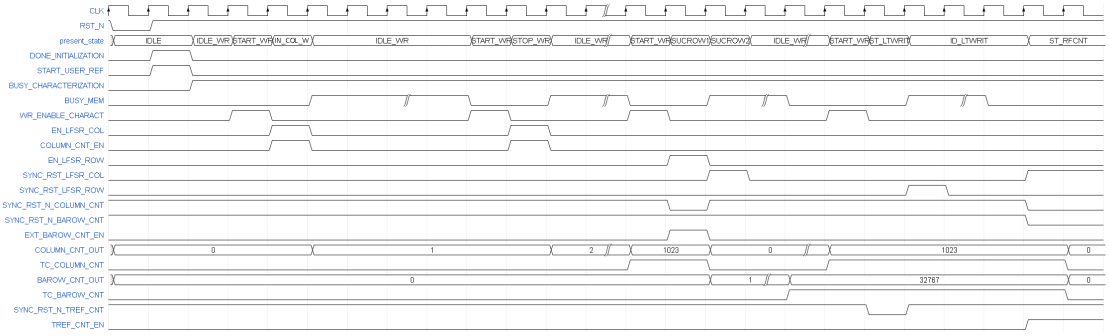


Figure 4.6: Writings step characterization timing

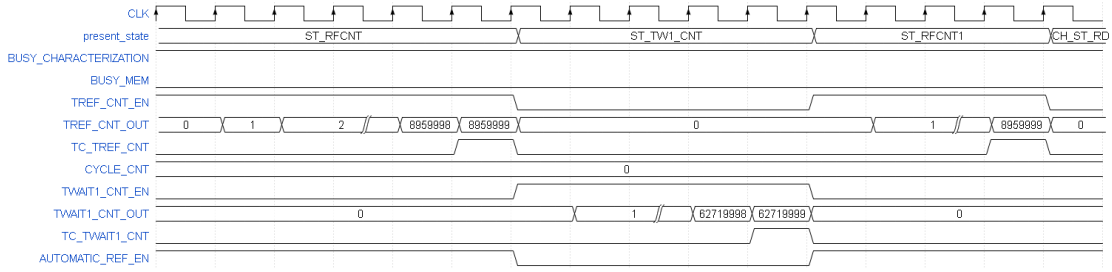


Figure 4.7: tREF + tWAIT step characterization timing

works in single word access for writings, each word is written in the current location and the column LFSR is enabled for each column through the *EN_LFSR_COL* signal. When the end of the row is reached (*COLUMN_CNT_OUT* = 1023), the row LFSR is enabled and the seed for the successive row is provided. In Figure 4.7, the timing shows the tREF + tWAIT step: during the tWAIT counting period, the *Auto-Refresh* is disabled as shown by the *AUTOMATIC_REF_EN* signal. In Figure 4.8 the readings-comparisons step is reported: again, for simplicity, single word access is shown for readings but the profiling works also for burst mode. The decisions for the thresholds update is made on the *OUT_COMP_WRIT_READ* signal value that comes after *RD_READY* assertion: if the comparison signal is equal to '1' till the last column location, the threshold update takes place for that row, otherwise the other columns are skipped and the threshold is not updated.

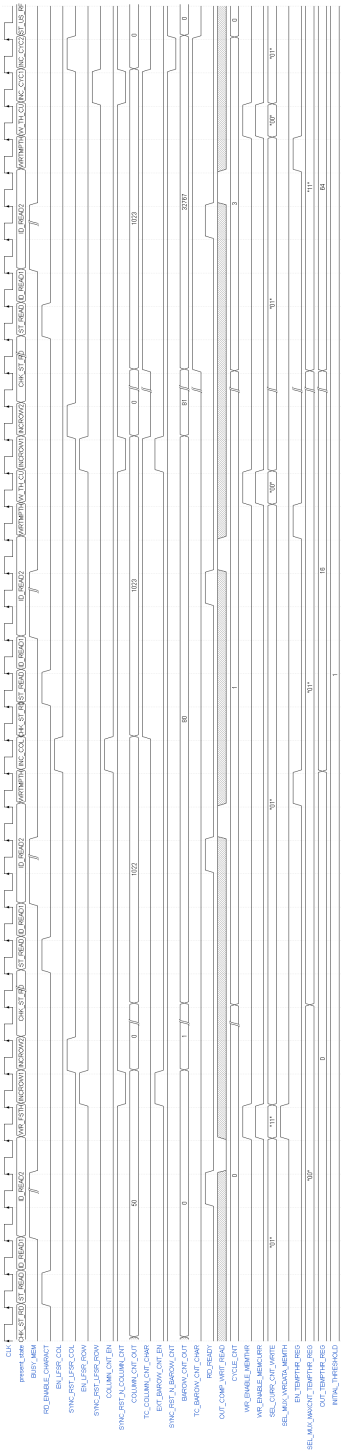


Figure 4.8: Readings-comparisons step characterization timing

In Figure 4.9, a first general overview of the realized architecture that surrounds the SDRAM core controller is showed.

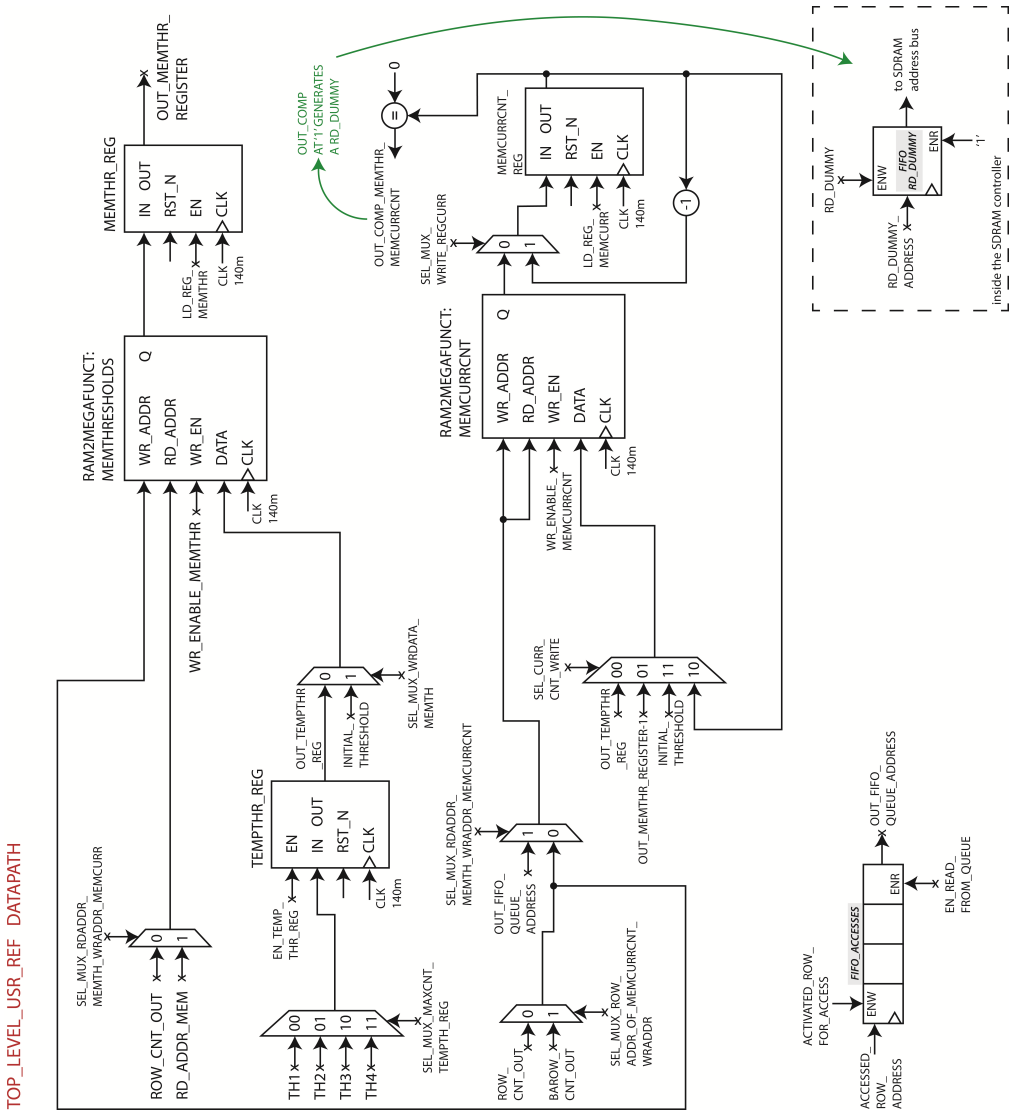


Figure 4.9: Datapath of the controller architecture for characterization and RRM

In the complete architecture there are also the counters needed for generating addresses, for the different tWAITs and for tREF too, for the row cycles across the RRM and also the row and column LFSRs used to write in memory and compare the readings for a pseudo-random coverage. These components are omitted for a better clarity of the figure.
As a final indication, some calculations have been made to understand how much time the characterization requires: the four retention times described before have been analyzed and waited for, so the sum of them is approximately equal to 7.68 s. Then you have to consider writings and readings cycles over all the SDRAM locations: the characterization machine executes four retention times cycles, each of them has one writings task and one readings task. The writings task is faster than the readings one because no comparisons take place: so considering 32768 rows by 1024 columns to be written and the clock cycles required to perform each of them at 140 MHz, the total four writings tasks across the four cycles, leaving out the negligible time when the memory is busy in performing *Auto-Refreshes*, require about 5.63 s. The reading task requires more time than the writing one because of the delay spent in comparing the outcomes, updating the thresholds or keeping them unchanged. So, the readings alone would have a time delay more or less equal to the writings one but, taking into account the delay through the states of characterization, the time spent is a little bit longer.
The design has been loaded and tested in terms of total delay required before entering in the *RAS-only refresh* states of the RRM: the entire characterization requires a total time of about 23 s for the used retention times, so the overhead added from the operations performed by the characterization machine together with the time spent for *Auto-Refreshes* is about l s to 4 s across all the entire procedure. This is a case of study and, of course, time consuming but after repeated rounds in worst case temperature conditions and for long periods, one could define the final thresholds and save them in a non-volatile memory on board to be used to load thresholds and current counters memories of the design at power-on, since retention times distribution showed to not change considerably along the time. After simulations in the following sections, it will be clear how it could be possible to choose the thresholds, and so the refresh rates, for each row across all the rounds at given room temperatures and some comments will follow regarding the main challenging problems linked to the profiling procedure.

4.3 Simulations and comments

In this section, simulations are performed to obtain the distribution of the retention times of the SDRAM cells grouped in rows, taking into account the temperature dependence.
In previous work [3], it has been demonstrated and showed how the retention time of DRAM cells exponentially decreases as room temperature rises. To provide a demonstration of this dependence, a sensor has been used to annotate the room temperature during the simulations. About 30 samples have been used to get final averaged temperature (according to the procedure duration), whose measure is completed with an absolute uncertainty of $\pm$ 0.5 °C provided by the used analog sensor. Across all the simulations, the lowest detected temperature was (16.95 ± 0.50) °C while the highest detected one was (26.37 ± 0.50) °C, so it means that a maximum span of 8 °C has been tested to see the effects on the cells.
The simulations have been divided in 10 rounds a day for the four discussed retention times, performing one simulation every half a hour for the first four days, and one simulation every hour for the last three days, globally covering a week of simulations. The fact that the simulations are repeated in successive rounds at fixed delay frees, as mentioned, the coverage of bit failures from DPD and then its increase or decrease is due mainly to the temperature variation or to VRT, hence limiting the errors to a small number. Moreover, the use of pseudo-random pattern has already been justified in precedence, since it is very similar to a random pattern that, in most of cases, induces the worst-case retention times behavior; but the used SDRAM has a small capacity, so probably even a static pattern like all 1s or all 0s would have provided an acceptable coverage [2], as seen in Figure 4.2.
In the following, figures of the first four days tests will be showed where MATLAB® has been used to analyze the obtained data. In Figures 4.10 and 4.11 there are the results of the simulations performed on day 1.

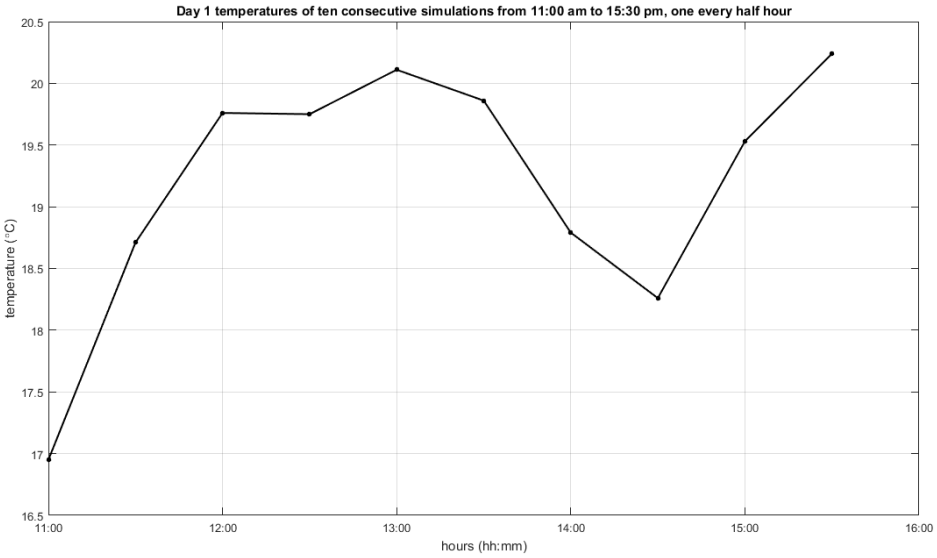


Figure 4.10: Day 1 - Temperature behavior across the simulations

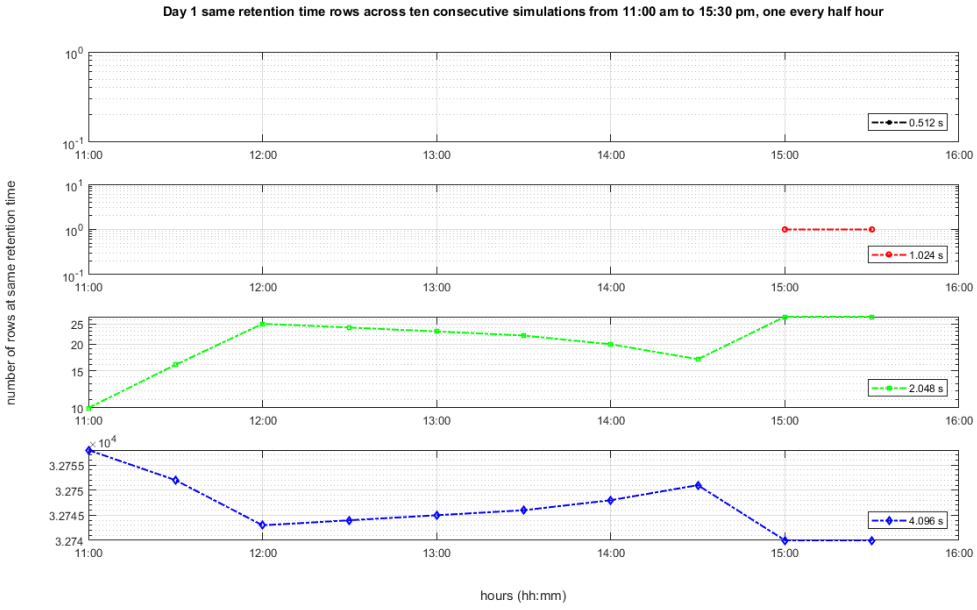


Figure 4.11: Day 1 - Number of rows variation at each retention time

In Figure 4.10 the room temperature variation is reported across the 10 rounds of simulation while in Figure 4.11 one can note the variation of the number of rows at the same retention time across the rounds that have been reported in four different subplots for each of the analyzed retention times. Since the reached temperature is not so high to have rows at the retention time of 512 ms at all and of 1.024 s apart from last two rounds, the curve of rows at 2.048 s has the exact behavior of the temperature trend while the one at 4.096 s has quite the opposite behavior. This makes sense: the higher is the temperature, the higher are the bits failures, the smaller is the number of rows at maximum retention time of 4.096 s that, as a consequence, switch to the previous retention time of 2.048 s. For a better understanding, these numbers are reported in Table 4.1.

At first impact, rows seem spending more time in high retention times states, where

Day 1 sim.	Retention time (s)				Room temperature (°C)
	0.512	1.024	2.048	4.096	
sim1	0	0	10	32758	16.95
sim2	0	0	16	32752	18.71
sim3	0	0	25	32743	19.76
sim4	0	0	24	32744	19.75
sim5	0	0	23	32745	20.11
sim6	0	0	22	32746	19.86
sim7	0	0	20	32748	18.79
sim8	0	0	17	32751	18.26
sim9	0	1	27	32740	19.53
sim10	0	1	27	32740	20.24

Table 4.1: Day 1 - Number of rows at each retention time across 10 rounds

an average of more than 99 % of the total rows is able to retain data at a refresh rate corresponding to 4.096 s and none of the rows has a retention time lower than 1.024 s. As visible, in fact, only one row appears to have a retention time of 1.024 s in the last two rounds as a result of the temperature rise. Looking at the table, whenever the temperature rises, the number of rows with retention time 4.096 s decreases and this consequently increases those with retention time lower and equal to 2.048 s, due to a high number of bits failures and then in agreement with the figures. As the retention time is expected to decrease whenever the temperature rises, the numbers in Table 4.1 seem to follow exactly the temperature behavior.
In Figures 4.12 and 4.13 and Table 4.2 there are the results obtained on day 2.

On day 2 simulations, the temperature always rises across the complete test and in fact, looking at Table 4.2, the number of rows at the retention time of 4.096 s

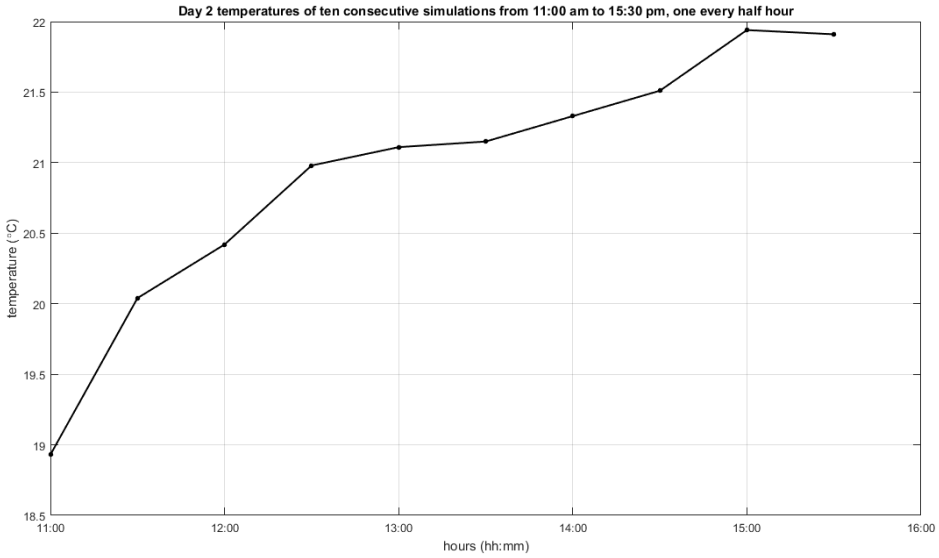


Figure 4.12: Day 2 - Temperature behavior across the simulations

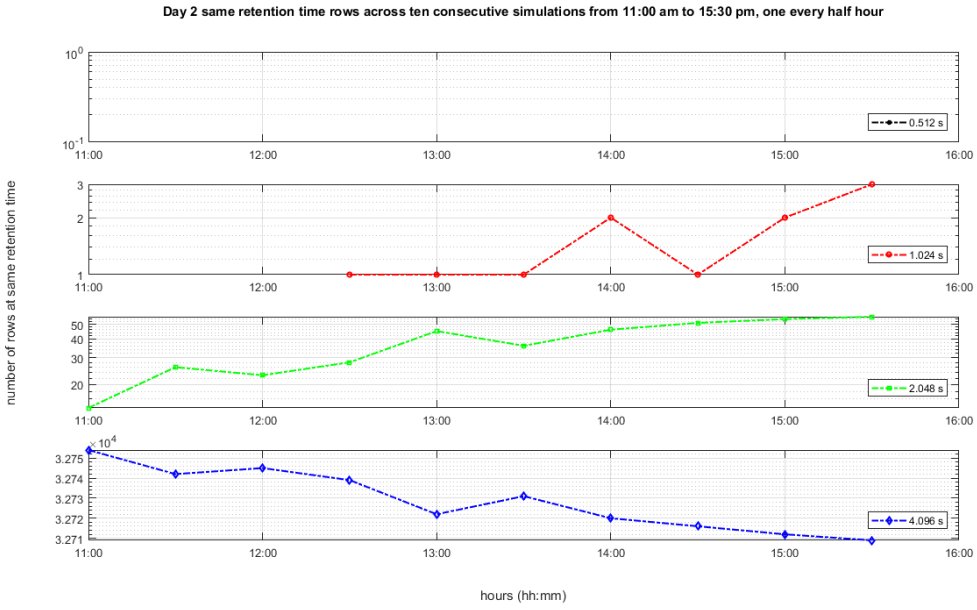


Figure 4.13: Day 2 - Number of rows variation at each retention time

Day 2 sim.	Retention time (s)				Room temperature (°C)
	0.512	1.024	2.048	4.096	
sim1	0	0	14	32754	18.93
sim2	0	0	26	32742	20.04
sim3	0	0	23	32745	20.42
sim4	0	1	28	32739	20.98
sim5	0	1	45	32722	21.11
sim6	0	1	36	32731	21.15
sim7	0	2	46	32720	21.33
sim8	0	1	51	32716	21.51
sim9	0	2	54	32712	21.94
sim10	0	3	56	32709	21.91

Table 4.2: Day 2 - Number of rows at each retention time across 10 rounds

has a decreasing trend along the time of simulation. In fact, the temperatures are higher with respect to the previous day, so more rows are claimed at lower retention times. This trend is more visible in Figure 4.13 comparing it with the temperature behavior in Figure 4.12 of the same day.
Figures 4.14 and 4.15 and Table 4.3 report the simulations performed on day 3.

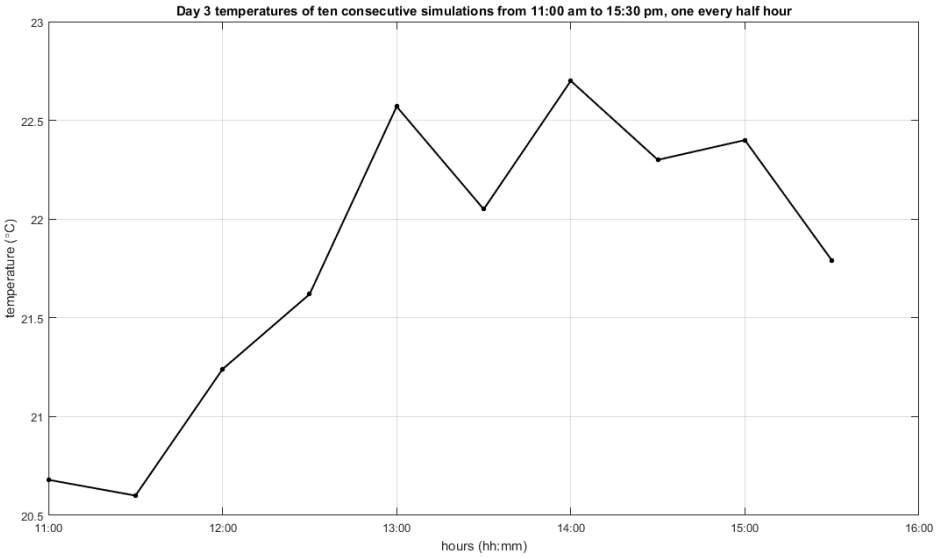


Figure 4.14: Day 3 - Temperature behavior across the simulations

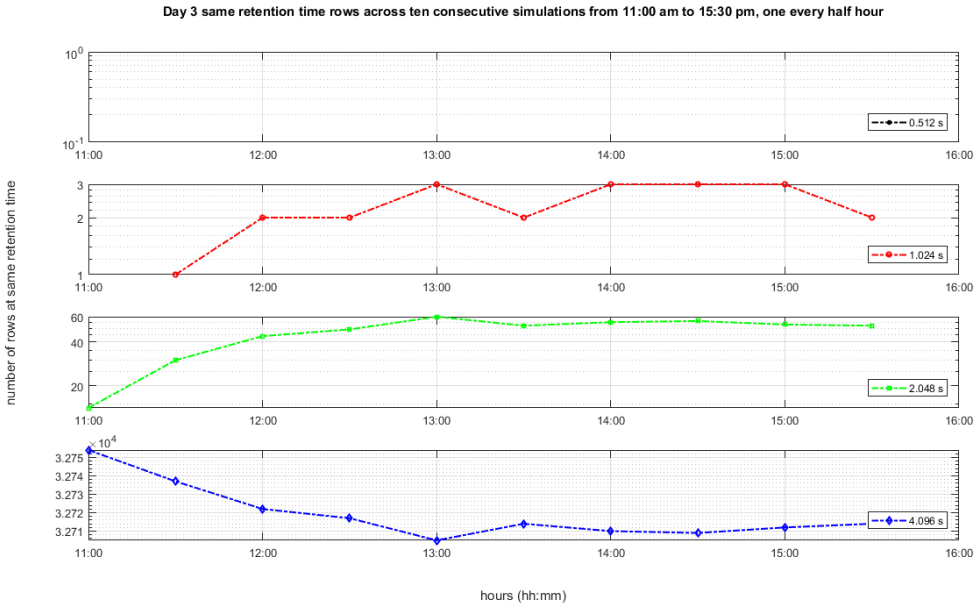


Figure 4.15: Day 3 - Number of rows variation at each retention time

Day 3 sim.	Retention time (s)				Room temperature (°C)
	0.512	1.024	2.048	4.096	
sim1	0	0	14	32754	20.68
sim2	0	1	30	32737	20.60
sim3	0	2	44	32722	21.24
sim4	0	2	49	32717	21.62
sim5	0	3	60	32705	22.57
sim6	0	2	52	32714	22.05
sim7	0	3	55	32710	22.70
sim8	0	3	56	32709	22.30
sim9	0	3	53	32712	22.40
sim10	0	2	52	32714	21.79

Table 4.3: Day 3 - Number of rows at each retention time across 10 rounds

Also in these rounds, the rows behavior is consistent with the temperature variation along the time of simulation. The highest reached room temperature, obtained in sim5, claims back 60 rows at the retention time of 2.048 s: in this case the number of bit failures becomes to be consistent. As final proof, the fourth simulation of rounds separated by half a hour has been conducted and the results reported in Figures 4.16 and 4.17 and Table 4.4.

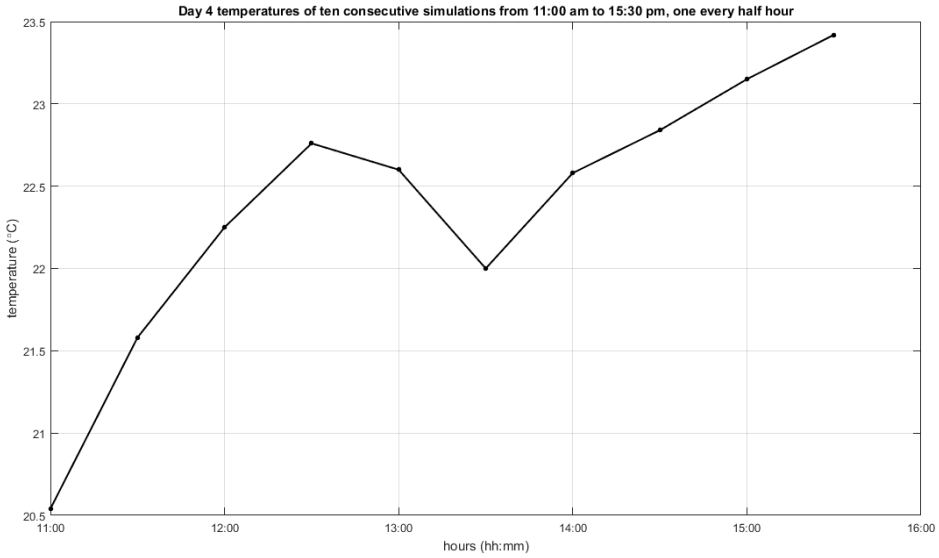


Figure 4.16: Day 4 - Temperature behavior across the simulations

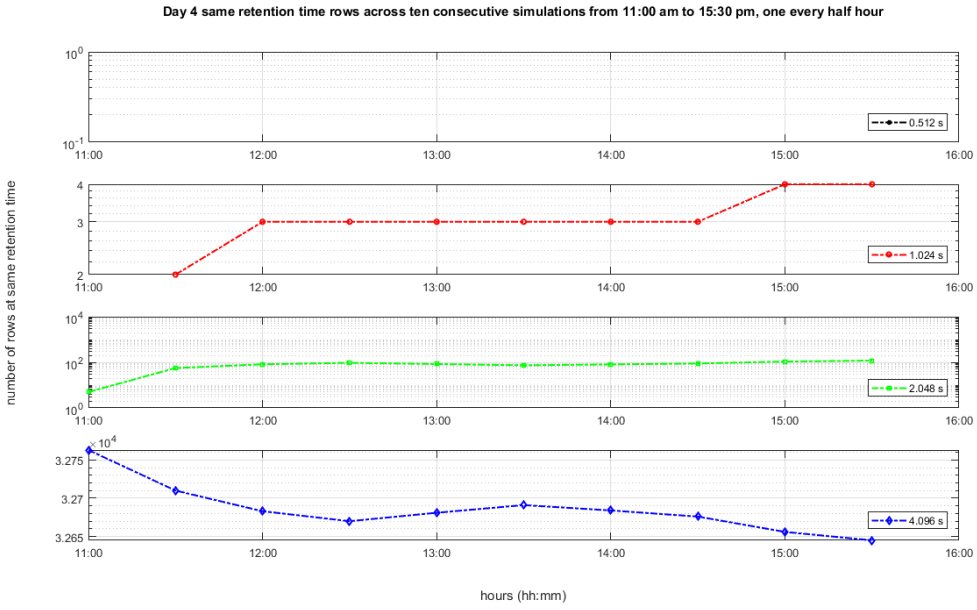


Figure 4.17: Day 4 - Number of rows variation at each retention time

Day 4 sim.	Retention time (s)				Room temperature (°C)
	0.512	1.024	2.048	4.096	
sim1	0	0	5	32763	20.54
sim2	0	2	56	32710	21.58
sim3	0	3	82	32683	22.25
sim4	0	3	95	32670	22.76
sim5	0	3	84	32681	22.60
sim6	0	3	74	32691	22.00
sim7	0	3	81	32684	22.58
sim8	0	3	89	32676	22.84
sim9	0	4	108	32656	23.15
sim10	0	4	119	32645	23.42

Table 4.4: Day 4 - Number of rows at each retention time across 10 rounds

The results obtained in the four days of simulations are pretty much in agreement each other: the rounds are separated by half a hour and the induced room temperature variation is reported. Although VRT is always present in the DRAM cells, in these cases the temperature has a predominant effect in their retention states changes. Moreover the 10 rounds are separated by only half a hour, covering for each test 4.5 hours a day. In reality, a typical VRT cell shows itself along a longer time of simulation, causing the failing of having more or less the same number of rows at the same temperature. For these reasons, three more days of simulations have been performed covering, with 10 repeated rounds separated by one hour, 8 hours of simulation. The SDRAM has been left idle among the rounds, refreshing the rows with the RRM.
Furthermore, the fact that the cells are grouped in rows could partially mitigate the effect of VRT, making the temperature variation the main reason of retention times change of the rows along the simulation. In fact, the retention time of a typical DRAM VRT cell has been verified to not fall below about the 2 s across 10 hours of simulations (Figure 11 [2]) and so these longer simulations could demonstrate this important consideration. Nevertheless, to avoid to extend too much this proof of consistency, only the last but one of the three simulations will be reported, where the maximum room temperature value has been reached. The results are presented in Figures 4.18 and 4.19 and Table 4.5.

The temperatures reached in the last two rounds are quite high and, indeed, there are lots of bit failures and then many rows at the retention time of 2.048 s. The results are again in agreement with the considerations done in previous simulations. Comparing the number of rows in this last simulation with the previous ones at the same room temperatures it is possible to notice some discrepancies,

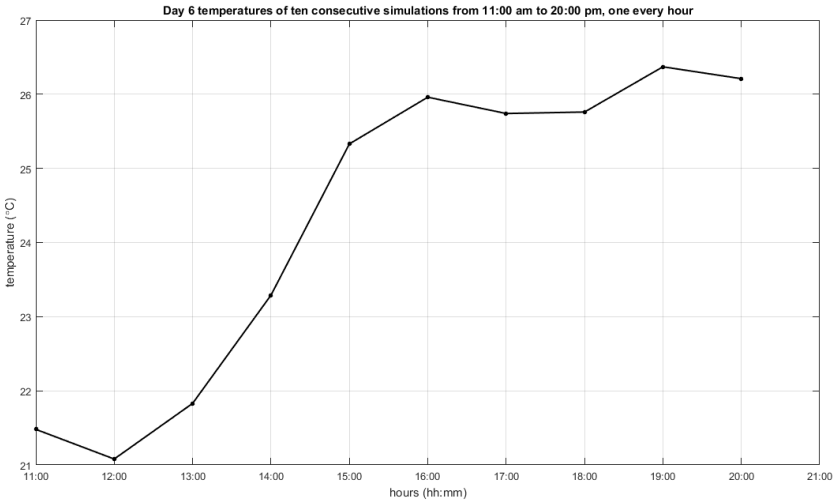


Figure 4.18: Day 6 - Temperature behavior across the simulations

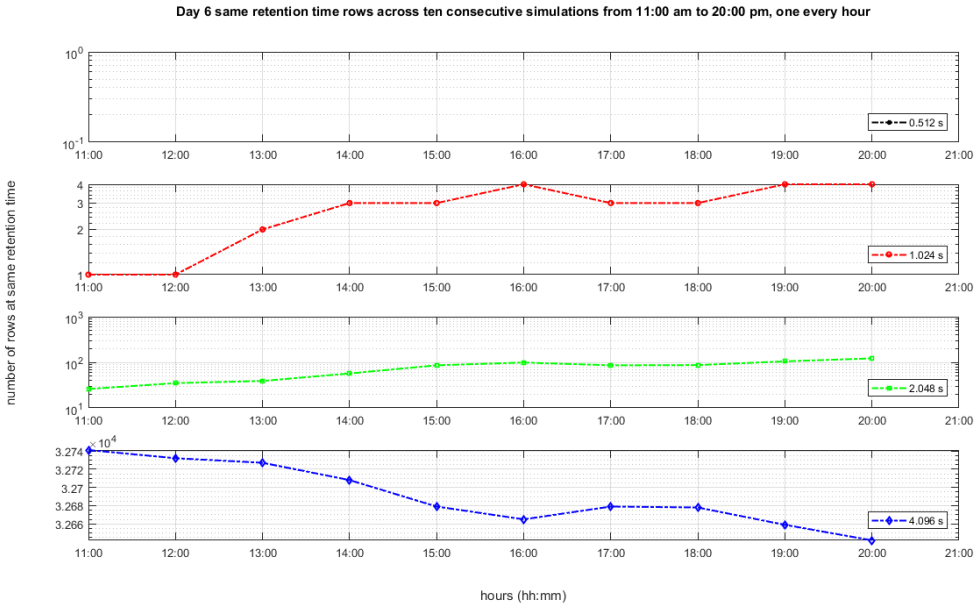


Figure 4.19: Day 6 - Number of rows variation at each retention time

Day 6 sim.	Retention time (s)				Room temperature (°C)
	0.512	1.024	2.048	4.096	
sim1	0	1	26	32741	21.48
sim2	0	1	35	32732	21.08
sim3	0	2	39	32727	21.83
sim4	0	3	57	32708	23.29
sim5	0	3	86	32679	25.33
sim6	0	4	99	32665	25.96
sim7	0	3	86	32679	25.74
sim8	0	3	87	32678	25.76
sim9	0	4	105	32659	26.37
sim10	0	4	122	32642	26.21

Table 4.5: Day 6 - Number of rows at each retention time across 10 rounds

which means that the longer simulation has been affected by multiple VRT retention states changes. Note that in the subplots sometimes a marker misses at one round: this is not an error but it simply means that the number of rows at that retention time and for that temperature is null. Hence, all the simulations show pretty much the same behavior between the trend of the number of rows at 2.048 s and the temperature change. Another important consideration is that at room temperature none of the rows has showed to have a retention time below 1.024 s: this is a strong result considering that all the rows are refreshed at 64 ms by default and this makes the idea of how much refresh latency can be saved with such modification. As a proof of this, in work [2] DRAM chips of different families with big capacities with respect to the tested memory have been simulated in this way and the result obtained for all the families is that none of the cells has a retention time lower than 1.5 s even at 45 °C (the retention time of 1.5 s was the first retention time of a finer analyzed span, as for these simulations is 512 ms in the coarser span). In conclusion, modern DRAMs of recent technology node and with such capacities (1 GB and 2 GB) compared with the used SDRAM (512 Mb) have been verified to retain data for 1.5 s at so high temperatures: then, considering the reduction of bit failures in lower-capacity devices and since the retention time of a typical VRT DRAM cell never falls under 1 s (Figure 11 of work [2]), it is possible to conclude that almost if not all the rows of the used SDRAM can be refreshed at 1.024 s refresh rate without incurring in losses of data.

Chapter 5

Controller architecture optimizations

Further optimizations of the controller architecture will be presented in this chapter and the final characterization and RRM machines will be showed together with the datapath of the architecture. The first applied modification is the so called *Selective Row Granular Refresh* (S-RGR [4]). Some applications could draw benefits from not refreshing some rows at all and further gains in terms of performances and power savings can be achieved. The second is an optimization regarding the temperature handling. The architecture, as it is till now, does not handle temperature variations that could cause heavy effects on the data integrity. Some considerations will be done before choosing the best solution that reduces as much as possible the latency of the controller in handling a temperature variation. An optimization of the *Row Access Strobe* (RAS) timing parameter will follow [4], making some background considerations on the physical refresh procedure that leads to restore the content in the issued row. Finally, clock gating technique has been applied to the controller architecture whenever the SDRAM enters in power-down mode.

5.1 Skipping refreshes solution

The architecture has been modified by adding a configurable SRAM in terms of memory depth. The data to store in the memory are the addresses of the SDRAM the user, intentionally, doesn't want to refresh at all. This kind of modification, that the system designer can decide to take advantage or not, is a way of calibrating the entire structure to the application in which the SDRAM will be used.
For some applications, there could be data that are not critical and their bit flips are not so destructive such to cause losses in performances: this could be the case of video coding or any other application that has a particular resilience to errors.
Another case in which this modification can provide considerable advantages is

in such applications that have dense accesses: in fact, in the memory locations that are sufficiently accessed, it would not be necessary to refresh them if they are continuously read or if their contents are constantly updated.
In some implementations present in literature, solutions with even the 50 % of non refreshed memory are provided: latency of the user requests is drastically reduced but, as obvious, the data integrity could be compromised if, for the target application, the memory spent most of the time in idle or without updating such locations. Hence a configurable signal is provided with the same name of the technique to decide using the SDRAM in this configuration and, then, a little on-chip memory is filled with the addresses of these locations not to be refreshed. But which addresses the machine will save? The higher is the retention time that a row is able to sustain, the stronger are the cells detected with the used procedure. So, as a consequence, according to the desired number of rows to not refresh, the same number of addresses of the locations founded out with the highest retention time are saved in this configurable SRAM. These addresses are saved during the characterization step when they are profiled and then they are skipped by the *Row Refresh Machine* when they are recognized to be the row addresses not to be refreshed, as will be reported in Figures 5.8 and 5.9.

5.2 Temperature effects handling

Although the controller architecture is aware of the distribution of the rows retention times and is able to refresh them accordingly, a temperature variation could affect drastically the behavior of the cells, as seen in previous simulations. Data integrity could not be ensured at all in this situation and the designed controller would be no more useful. As seen in Figure 1.2 from work [2], the distribution of cells retention times has an exponential behavior across the temperatures: so even a slight increase of temperature could prevent the controller to issue correct refreshes.
The authors of this experimental study have extracted the equation that fits with the worst case behavior of the retention times distribution with respect to room temperature, reported here:

$$x(T) = A * e^{-0.0625T} + C \tag{5.1}$$

The retention times distribution and then the equation is normalized to a reference room temperature of 45 °C, that is quite high and also higher than the maximum temperature reached during the simulations performed in section 4.3. This means that for our case this equation induces the worst case too, since such a high temperature has not been reached as a reference for the retention times distribution. Then, in order to allow the controller to be aware of the temperature variation a design modification has been provided. Let's call T_{REF} the reference

room temperature at which the retention times profiling is performed: looking at the worst case induced equation 5.1, whenever the temperature rises by about 10 °C the retention times have to be halved. If not done, most of the cells would have bit flips and the memory could incur in data corruption. Hence, a temperature resolution of 10 °C has been considered to be the value of temperature at which the variation could be consistent and some actions need to be taken. To do that, the user has to provide on-line temperature or some other mechanism to check for room temperature variation. The architecture provides an input signal, called *CURRENT_TEMPERATURE*, that is initially set to "00" at reference temperature of the profiling. Whenever the user detects an increase of temperature by 10 °C, simply changing this signal from "00" to "01" the retention times are halved. In practice, the controller has memory of the previous room temperature and according to the new value set by the user it can halve or double the retention times accordingly.
This 2 bits wide signal so allows to handle a maximum temperature variation of 30 °C where the retention times are reduced to one eighth of their initial values reaching T_{REF} + 30 °C or, if the temperature decreases, the user can change the input signal value and the controller detects a lower current temperature with respect to the previous one: the taken action is to double the retention times up to a reduction of the temperature until the initial T_{REF} where the starting profiled retention times are restored. Clearly, the temperature variation span could have been larger than 30 °C but the advice is that, due to the strong temperature effects on the retention times, over a temperature variation of 30 °C it would be better to perform again the characterization to provide safer values of retention times.
However this temperature handling span guarantees to change the retention times on the fly, without performing every time the profiling to have a correct refresh behavior and so loosing too much time in providing an acceptable memory latency response. In Figure 5.1, a simple representation of what happens in the case of a temperature variation is showed.

The controller, as said, has memory of the last temperature variation by using two registers and, using two comparators, detects if a temperature change has occurred (setting *TEMPERATURE_CHANGED* to '1') and in which direction (determining the divisor/factor), as an increased or decreased temperature. Finally, a FIFO temperature of depth one is added, to avoid loosing a temperature variation if the controller is issuing any other operation in the RRM. Whenever it has finished its operations, it can pay attention to a detected temperature variation and serve it. A FIFO of depth one is sufficient provided that the temperature doesn't change of 10 °C in 64 ms, which is quite improbable. When a temperature variation is detected, the taken action is to save in the FIFO temperature the divisor/factor that will be used to update the thresholds inside the homonym and current counters memories. So this value, out of the FIFO temperature, points to a multiplexer where the thresholds memory output, halved or doubled, is chosen as input write data to the memories: this value is written, as it is, inside the thresholds memory while it is

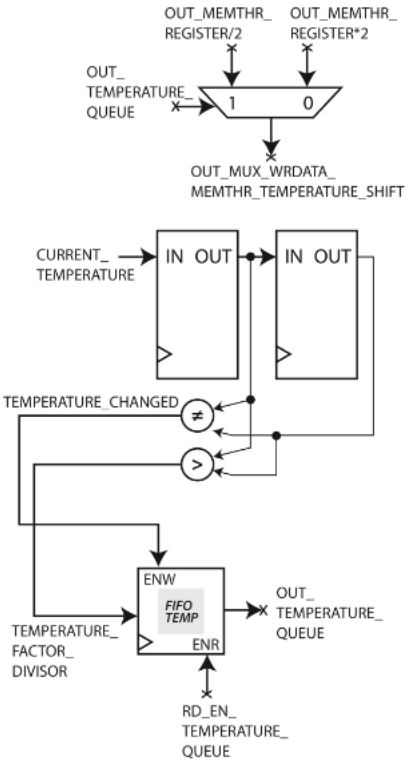


Figure 5.1: Temperature handling interface

subtracted by one for the counters memory before being written in, for the same reasons described in chapter 3. However an important consideration must be taken into account: if a temperature variation is detected in the very exact moment when a row needs to be refreshed, since the temperature change has the effect to restore the half or the double of each corresponding retention time, the row would lose the dummy reading in its elapsed counter cycle and incur in bit failures. Thence, keeping in this worst case situation, whenever a temperature change is handled, a dummy readings cycle is performed to every row ensuring so the rows data integrity. In the following, some simulations of the temperature handling procedure are provided.

In Figure 5.2, during the normal working, the user sends an increase of temperature of 10 °C over T_{REF} (*CURRENT_TEMPERATURE* changed from "00" to "01") and *TEMPERATURE_CHANGED* is asserted writing into the FIFO temperature. Since the RRM is not doing useful operations, the temperature variation is immediately served: the row counter is reset and for the next 64 ms refresh commands are asserted to each row. At the same time the thresholds are updated inside the memories and since the temperature has risen, the thresholds have to be halved: in fact, as the *INITIAL_WRITE_THRESHOLDS* signal states, all the

Figure 5.2: Temperature rise of T_{REF} + 10 °C detected

initial thresholds are set to 4 like in the simulations examples presented in chapter 3 and so they have to be halved to 2 and written back to the thresholds memory. The Q signal, standing for its output, demonstrates that actually this happens. The current counters memory output, instead, shows that the thresholds are correctly set to 1.

In Figure 5.3, a further temperature increase of 10 °C is detected (input signal

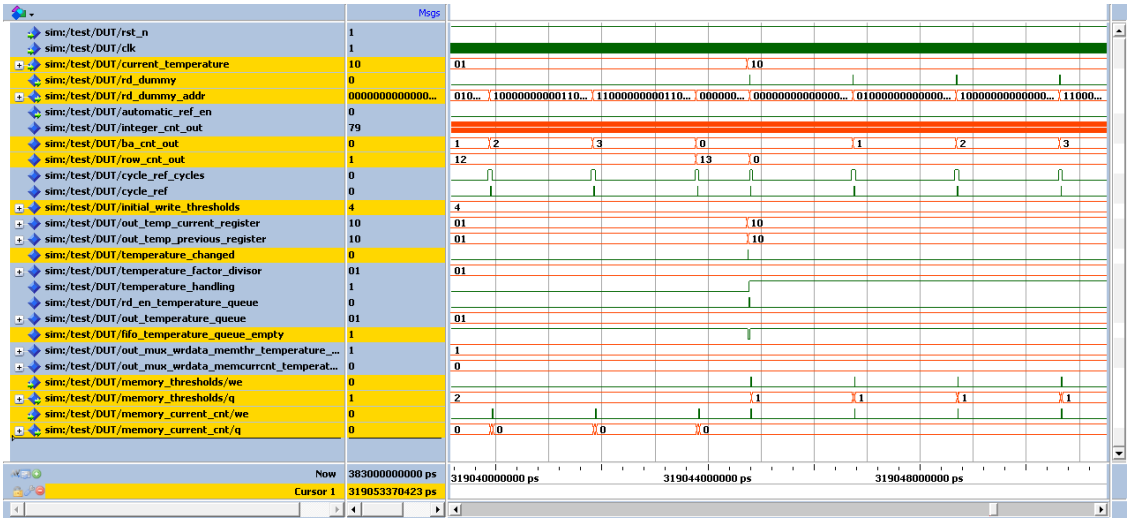


Figure 5.3: Temperature rise of T_{REF} + 20 °C detected

changed from “01” to “10”), reaching a temperature variation of T_{REF} + 20 °C.

Also in this case the thresholds are halved, then resulting to be one fourth of the initial thresholds at the reference profiling room temperature. This means that if the starting thresholds were all fixed to 4 (tREF = 256 ms), now the thresholds are all set to 1 (tREF = 64 ms) and this means that now the rows are refreshed every 64 ms as for the standard *Auto-Refresh*. It is an example where all the thresholds are fixed to the same initial value, but the profiling provides a different retention times distribution this temperature handling takes advantage from. However, in the worst case where a temperature halving leads to obtain a value lower than 64 ms, although the memory is not in the extended temperature range the thresholds updating is skipped as will be showed in Figure 5.10.
In Figure 5.4 it is possible to notice what happens at the end of the previous temperature change handling: as described before, now all the thresholds correspond to a refresh window of 64 ms and so, after the end of the temperature variation handling (*TEMPERATURE_HANDLING* is de-asserted), dummy readings are immediately asserted as refresh commands. The RRM terminates issuing dummy readings caused by the temperature change and then continues to issue refresh commands as the row counters are elapsed. So the thresholds continue to be correctly managed by the controller architecture.
Finally, in Figure 5.5, the same previous 64 ms refresh window of dummy readings

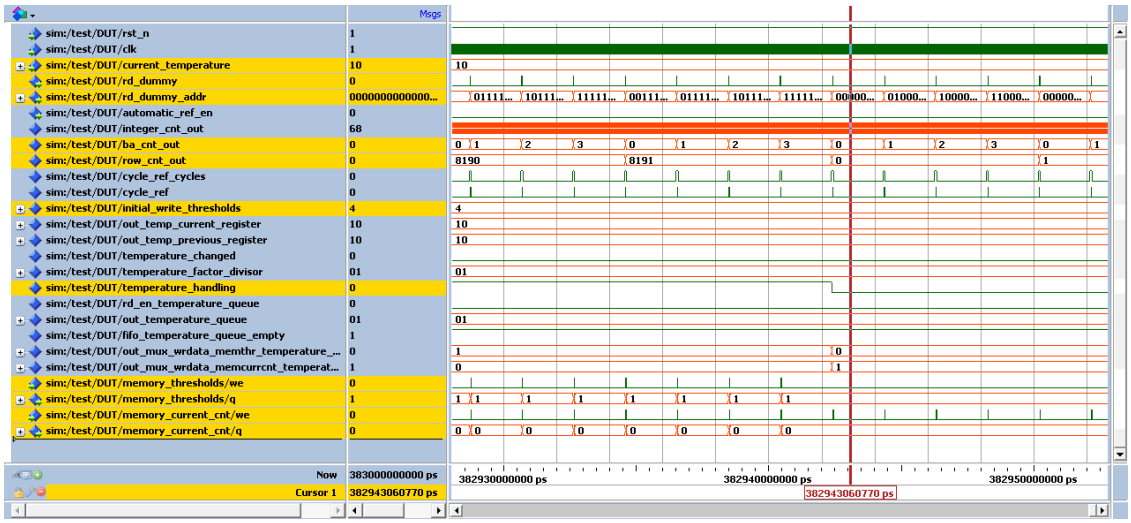


Figure 5.4: Previous temperature rise of T_{REF} + 20 °C end of handling

is interrupted by a temperature change, actually a temperature fall of 10 °C (*CURRENT_TEMPERATURE* changed from “10” to “01”). This situation describes the reason for which it is necessary to continue issuing dummy readings whenever a temperature change is detected: if the controller is refreshing the rows whose counters are elapsed and a temperature variation occurs, the change of thresholds could cause the skipping of the refreshes for these rows incurring, probably, in data

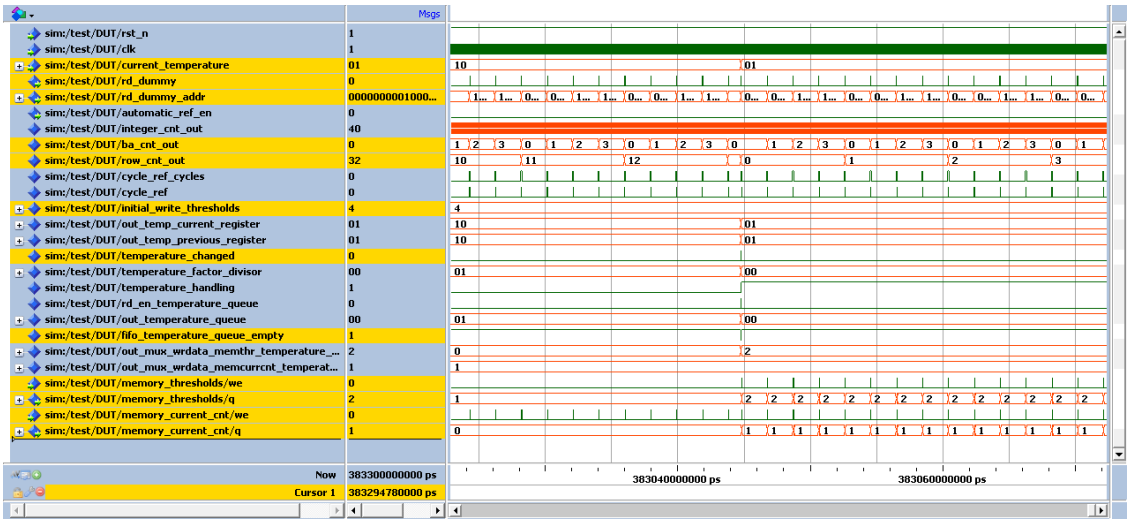


Figure 5.5: Temperature fall of 10 °C, reaching again T_{REF} + 10 °C

corruptions. This because a temperature change must have the highest priority in the RRM to fastly update the thresholds and, then, this worst case situation induces the controller to perform a refresh window cycle to continue guaranteeing the data integrity of the rows content whenever a temperature variation is detected. In conclusion, also a challenging effect like the temperature dependence of the retention times has been handled, providing a smarter controller aware of the surrounding working condition. In Figures 5.6, 5.7, 5.8 5.9 and 5.10 are reported the final ASM charts of the entire controller architecture that has been designed. Figures 5.8 and 5.9 yield also the solution for the S-RGR [4] feature while Figure 5.10 refers to the ASM chart states needed to handle a temperature variation detection.
Lastly, in Figure 5.11, the datapath of the final controller architecture is presented. The picture is minimized to show only the most important RTL blocks.

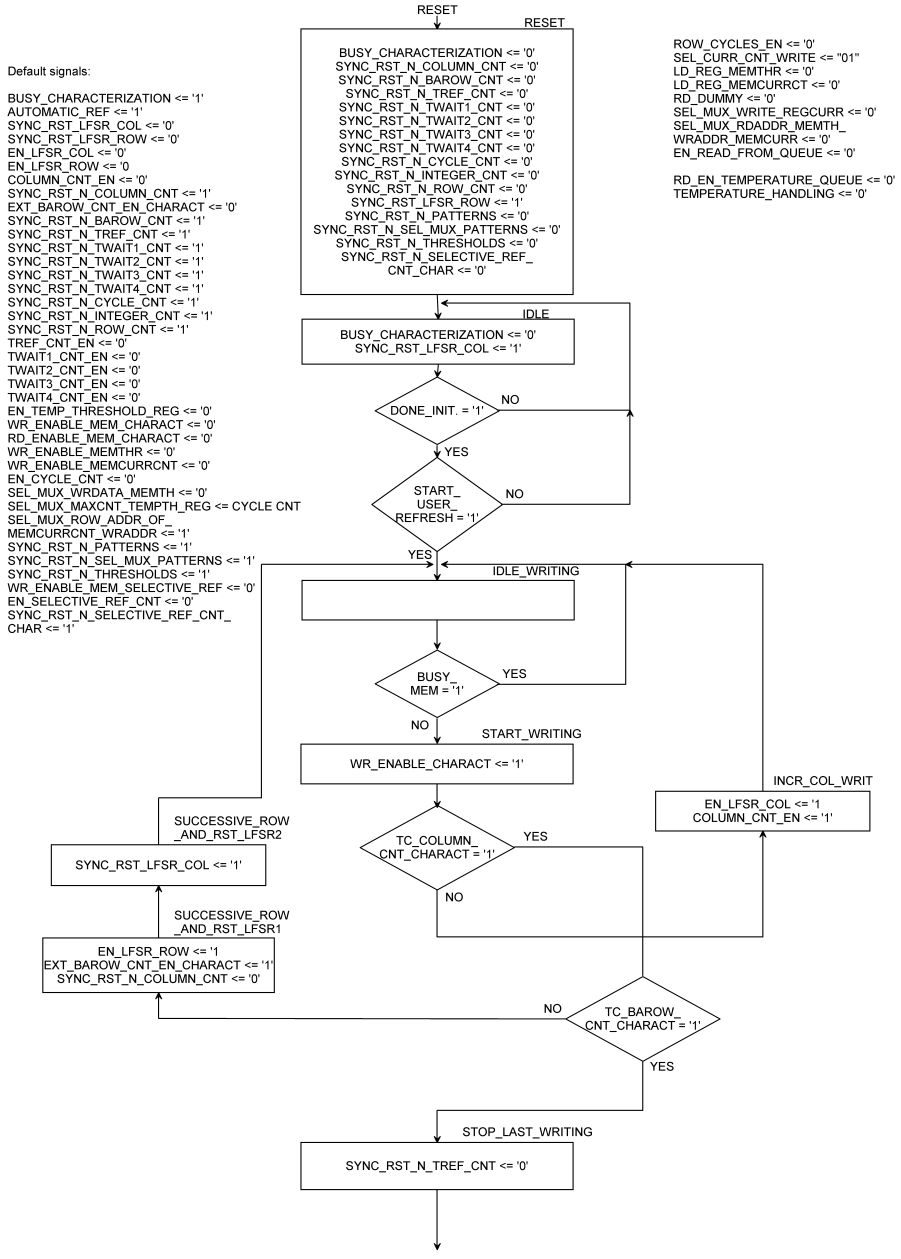


Figure 5.6: Final characterization and RRM machine ASM chart - Writings step

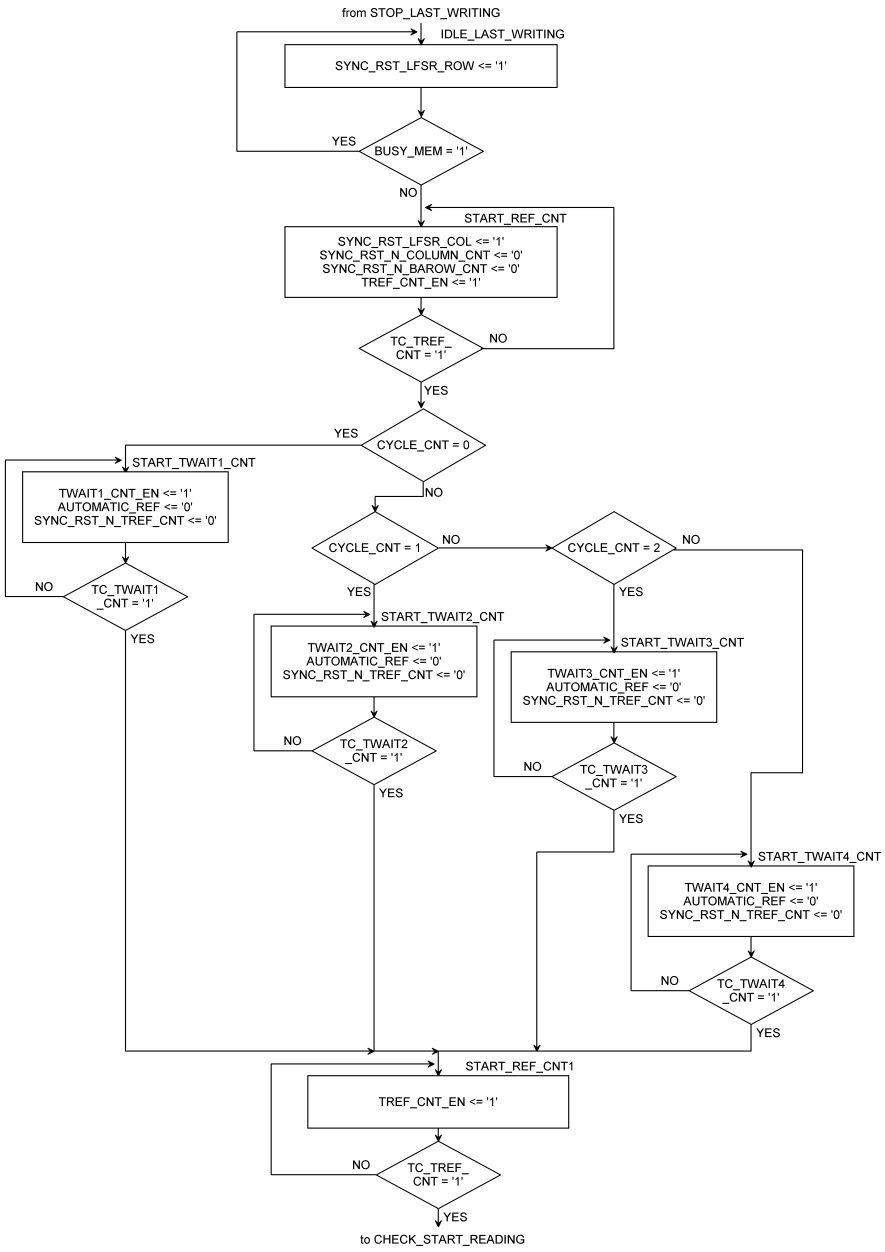


Figure 5.7: Final characterization and RRM machine ASM chart - tREF + tWAIT step

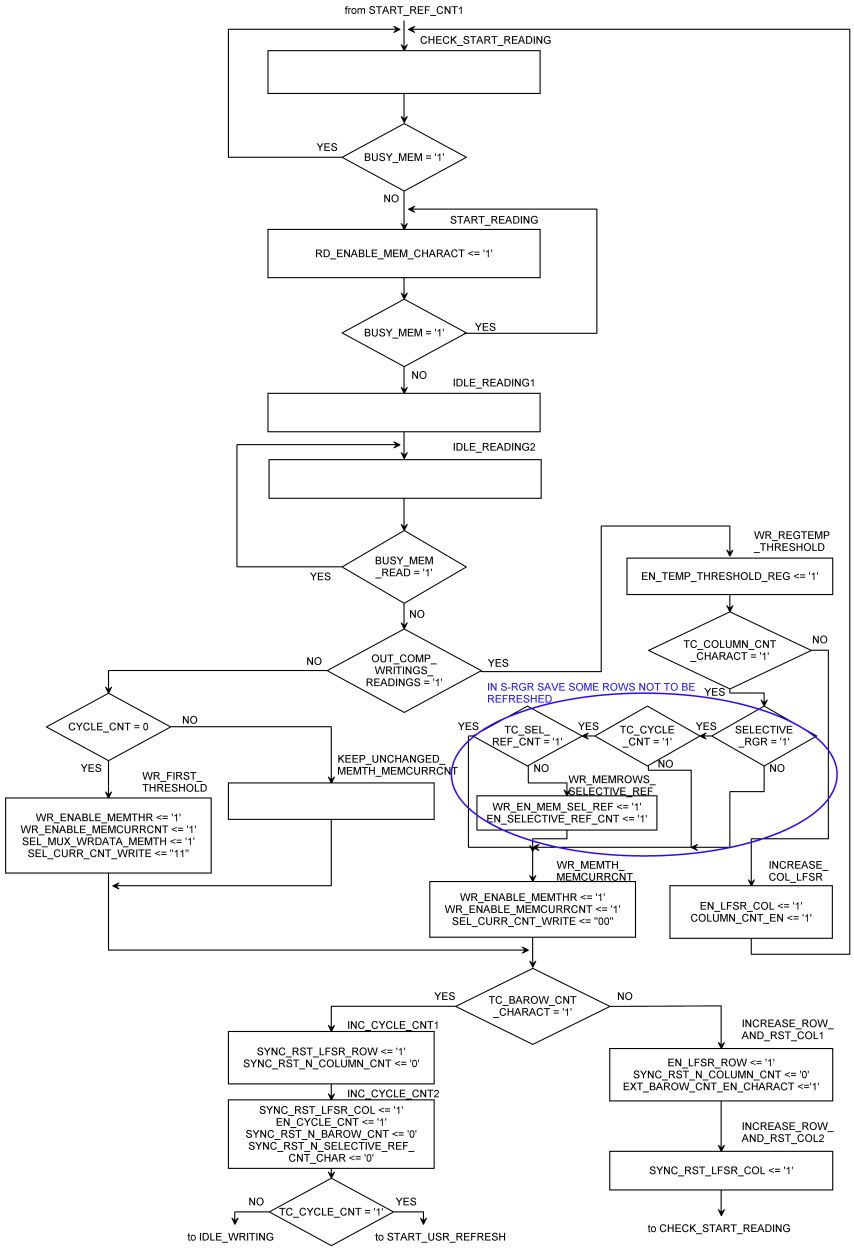


Figure 5.8: Final characterization and RRM machine ASM chart - Readings-comparisons step

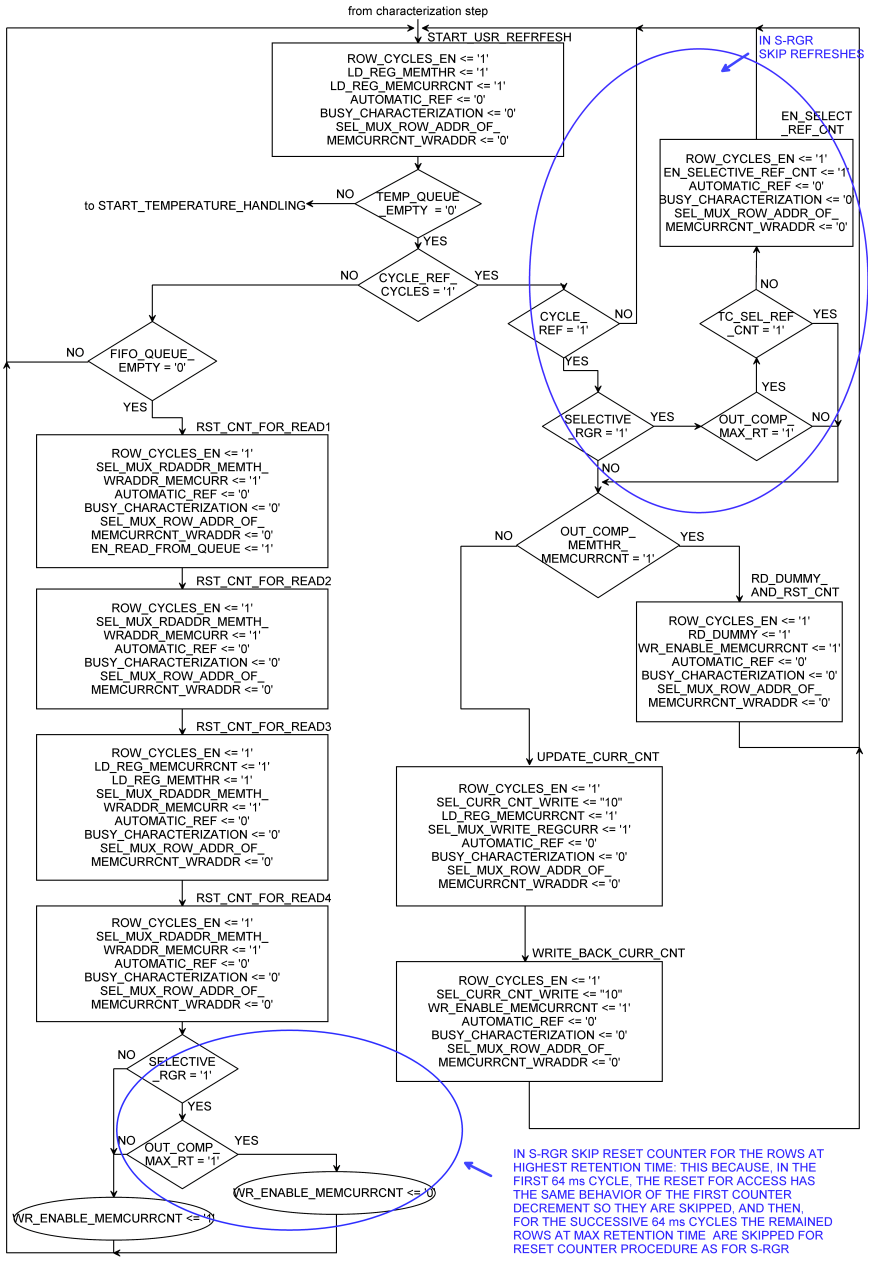


Figure 5.9: Final characterization and RRM machine ASM chart - Skipping refreshes handling

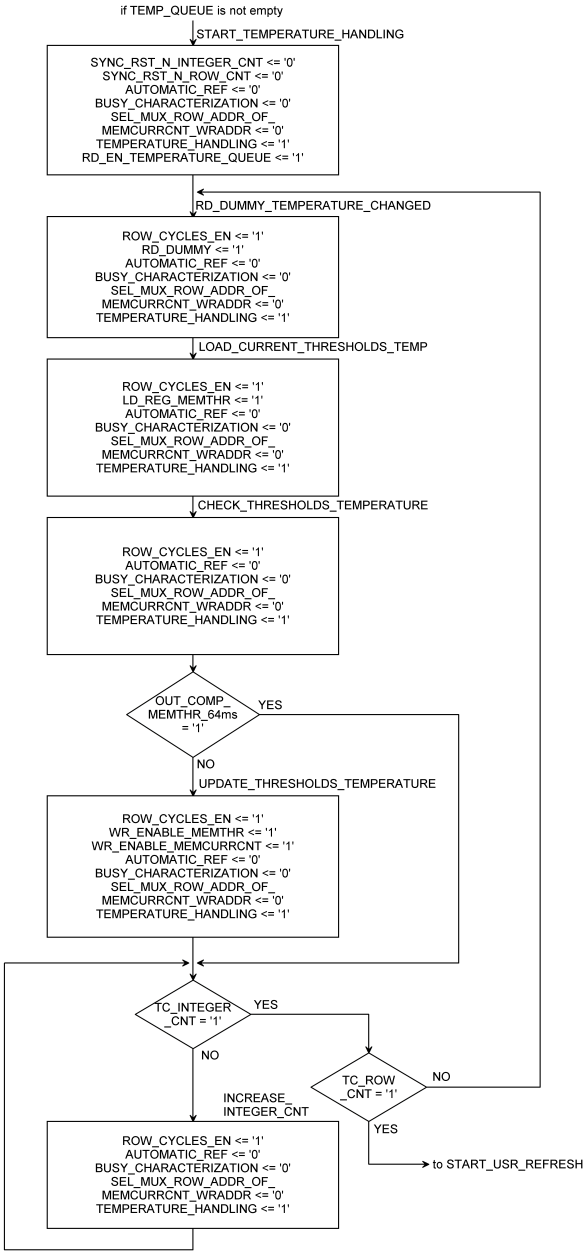


Figure 5.10: Final characterization and RRM machine ASM chart - Temperature handling

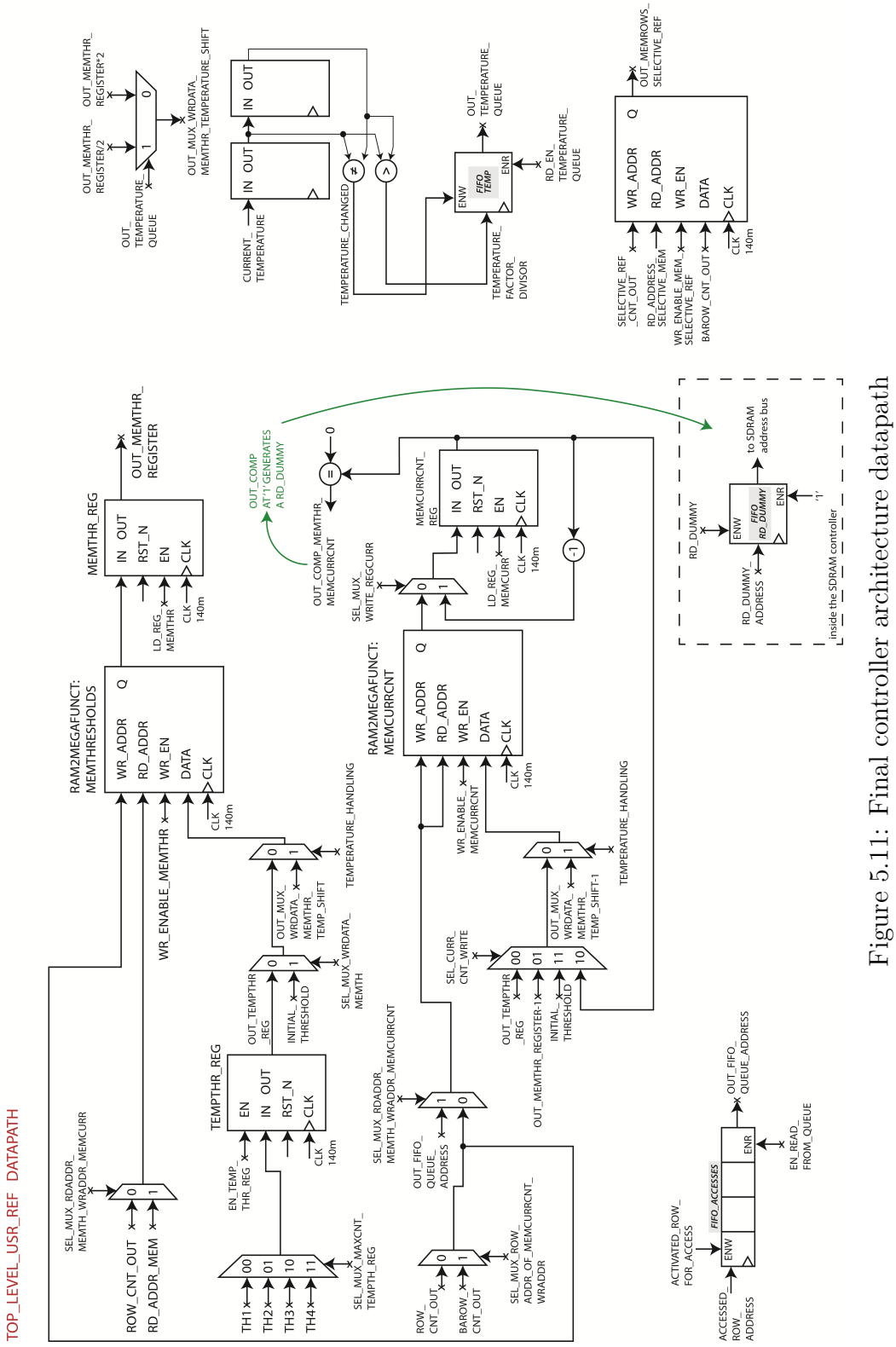


Figure 5.11: Final controller architecture datapath

5.3 RAS time delay reduction

In this section the controller architecture has been optimized making some considerations about SDRAM timing parameters. The effect of applying a row-by-row based refresh is to lose all the optimizations that come with the use of *Auto-Refresh*, where internally multiple rows are refreshed at a time. Following the idea of work [4], a controller can find out the optimized timing parameters used by the *Auto-Refresh*. In this way the controller can still issue row-by-row refreshes but having the same advantages of *Auto-Refresh* in issuing refresh commands, that means energy and performances improvements. In simulations conducted in the same work, it has been found out that this mechanism is even more efficient than the standard optimized *Auto-Refresh* when applying *Row Granular Refresh* [4], then it is worth doing some considerations to understand if it is applicable to this implementation. The followed idea is the same and requires a controller customization to handle the sequence of testing. The idea that comes from work [4] concerns mainly the reduction of the tRAS timing delay that the DRAM experiences between an active command (ACT) and a precharge command (PRE): in fact, whenever a row needs to be refreshed, a row level refresh issues an ACT command that places the entire row in the *row buffer* of sense amplifiers and then, after a PRE command is sent, it restores the content of the row in the same location. An active command, hence, specifies the bank and the row on which this operation has to be conducted and a precharge command closes the row and precharges the bitlines for the next request. However, this duet doesn't need to wait for bitline sensing, because no voltage difference is detected on the bitlines since no column address of a specific location has been sent on the address bus. Therefore, apart from a standard read or write operation where the sensing is needed to correctly terminate this operation, in row level refreshes such long tRAS is not necessary and so can be reduced and still obtain reliable refreshes. In modern DDR3 memory chips, the measurements conducted through the FPGA-based platform developed in work [4] have found that the optimized tRAS shortens the refresh cycle period tRFC up to 45% with respect to a such optimized *Auto-Refresh*. This makes the idea of how much access response latency can be saved especially in upcoming DRAM chips, that are expected to become denser and denser, increasing then the refresh cycle time tRFC to cover all the rows. The timing parameters involved in tRFC computation are:

- tRAS → Row Access Strobe Delay
- tRP → Row Precharge Delay
- tRRD → Row-to-Row Delay
- tFAW → Four Activate Window Delay

where tRRD stands for the time delay between active commands to different selected banks and tFAW stands for the delay window where at maximum only

four consecutive active commands can be issued. tFAW constraint can be neglected for the used SDR SDRAM, since it is valid especially for modern chip memories working at double data rate. To reduce refresh latency, the idea of the cited work has been followed but customizing the controller. Instead of reducing all the timing parameters, tRAS, tRRD and tRP, the attention has been focused on tRAS only, since tRP and tRRD intervene also during standard read/write operations and then JEDEC [19] standard values have been kept for these parameters. For the considerations made before, an active command is not followed by a read command waiting for bitlines sensing, but it is followed by a precharge command and so total tRAS can be reduced for refresh operations without affecting read/write ones.
The procedure has been executed by adding a down counter, counting $\text{tRAS}_{\text{curr}}$ + tRP + CAS LATENCY cycles, that at terminal count resets to its starting value in that test minus one if the memory passed the test for that current $\text{tRAS}_{\text{curr}}$. The SDRAM core controller, instead, is modified by adding a down counter as well to the tRAS value that is decremented every time the memory passes the test, in agreement with the previous down counter updating letting tRP and CAS LATENCY unmodified. Moreover, the sequence of active and precharge states has been added and executed when the starting minimum tRAS evaluation command is triggered. Therefore, 6 more states have been added to the control unit just after the characterization step to perform this evaluation. A known pattern is first written to a random memory location, then the sequence of active and precharge command is followed and, after the precharge delay, a read command is issued to the same location: the comparison on the data bus says if the memory has passed the test for that tRAS, since data bus is not driven and remains in high impedance if the read command is not triggered in the correct way (recognized as illegal operation in SDRAM commands table). In Figures 5.12, 5.13 and 5.14 the timings of the sequence of operations are reported, where the first and second figures represent the two passed tests cases for 6 tRAS cycles and 5 tRAS cycles respectively, while the last one represents the case in which SDRAM did not pass the test for 4 tRAS cycles and, hence, its value is not updated.

The datasheet of the used SDRAM fixes the number of cycles for tRAS timing parameter to 6 clock cycles. This means that, for the used speed grade and the actual used clock frequency of 140 MHz, the total tRAS delay corresponds to about 42.86 ns. After the evaluation sequence, the value has been reduced of one cycle, hence to about 35.71 ns. So a reduction of about 7 ns has been obtained on the tRAS value, providing benefits over *Auto-Refresh* and RGR [4] in terms of refresh latency and saved energy. The reduction is not significant for one reason: the cycle period of the used frequency corresponds to about 7.14 ns, that means that the scaling of the tRAS value comes for about 7.14 ns each and the minimum tRAS can be reached early. The choice to not use another FSM at higher frequency to get a higher resolution stands in the fact that control units at different frequencies are typically a bad design implementation choice and the conversions of long to short

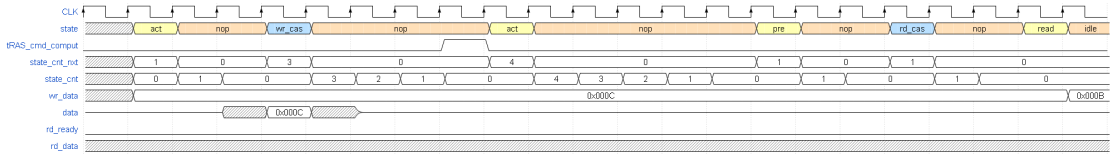


Figure 5.12: tRAS passed test for 6 cycles delay (JEDEC [19] standard value)

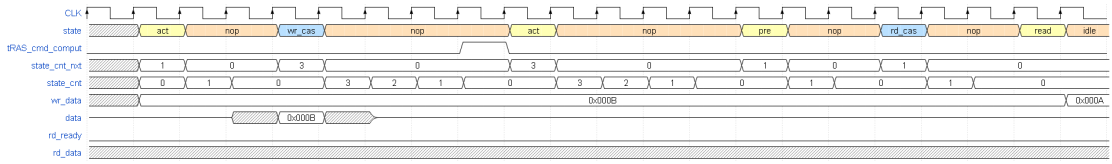


Figure 5.13: tRAS passed test for 5 cycles delay

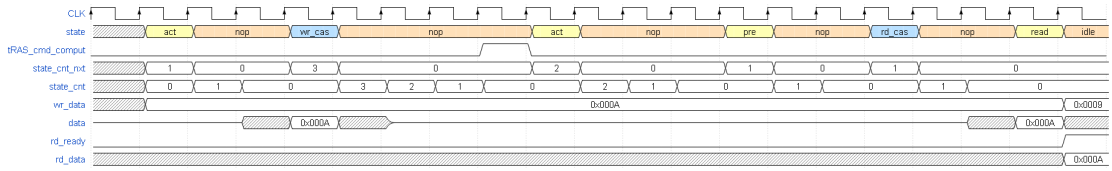


Figure 5.14: tRAS failed test for 4 cycles delay

strobes and vice versa could require inverter chains whose delays are technology dependent. In fact, the problem is mainly due to the used SDRAM that does not support higher frequencies causing so a minimal tRAS reduction. The problem is solved and further reductions and benefits are achieved when using a faster SDRAM: a modern DDR reaches frequencies of the order of GHz, hence providing scaling resolutions of ns per cycle. Then, replacing the used SDR with a modern DDR and its core controller, the reduction in tRAS value could be considerable and then a more reliable minimum value can be obtained. So, the same frequency has been kept for this procedure, addressing further improvements for modern DRAM chips. In fact, the FPGA-based experimental setup of work [4] was tested on a DDR3 working at the order of GHz providing so a scaling resolution of ns: the obtained reduction is of 10 ns, achieving about 35 % reduction over JEDEC [19] standard tRAS value of 28.3 ns for that memory.
Nevertheless, an improvement is obtained also at the used frequency of 140 MHz. In fact, compared to the RGR with standard tRAS in a tREF cycle of 64 ms, the time spent in refreshing is reduced from 2.11 ms to 1.87 ms, obtaining so a reduction in tRFC delay and then an improvement of system performances of about 11.4 %, due to the reduced access response latency. This, as a consequence, reflects in a more energy efficient solution.

5.4 Power-down mode clock gating

Power-down mode allows to save power consumed by the memory whenever active operations are not going to be performed for a certain amount of time. The clock is suspended and the SDRAM can experience either precharge power-down, where all the banks are kept precharged, or active power-down, where at least one bank is activated when the user puts the memory in this low power mode. The problem is that the realized controller continues to perform useful operations while flowing through RRM states even if the memory is in power-down mode and is not able to listen to any kind of operation. In order to approach to a low power solution for the realized controller architecture, clock gating technique has been used to suspend the controller clock whenever the user puts the SDRAM in power-down mode. Classic clock gating scheme used is showed in Figure 5.15.

A D latch active on the low level clock allows to filter out possible glitches coming with the power-down mode signal provided by the user. The use of a D latch allows to have full period to sample the enable signal with respect to a D flip flop negative-edge triggered, avoiding so to halve clock frequency only for this glitches filtering. When the controller is acknowledged by the memory entering in power-down mode, the latch samples the power-down signal and suspends clock for the controller: this means that all the controller operations are stalled till the memory exits from this low power condition. In Figure 5.16 is reported a piece of simulation in which the user puts the SDRAM in power-down mode for 30 µs: as visible, all the controller

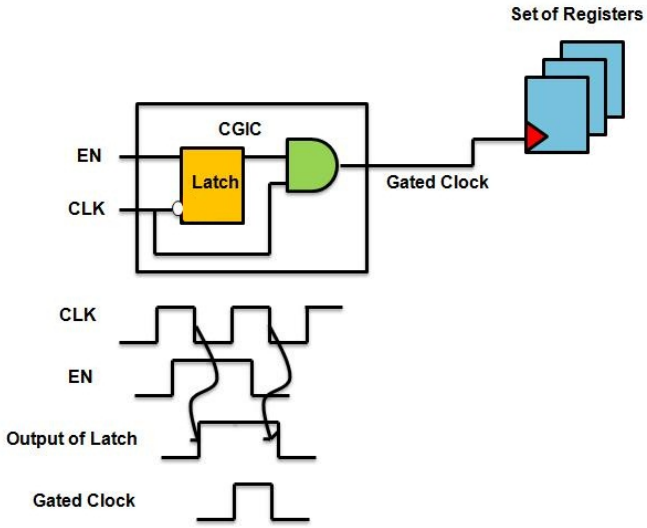


Figure 5.15: Clock gating scheme [18]

signals are interrupted for the entire duration of *POWER_DOWN* signal at '0' logic. This results in further power savings for the memory controller: however, as for *Auto-Refresh*, is the user's concern to exits from power-down mode as soon as possible since no refreshes are issued in this condition.

The power consumptions that are saved for the SDRAM in power-down mode will

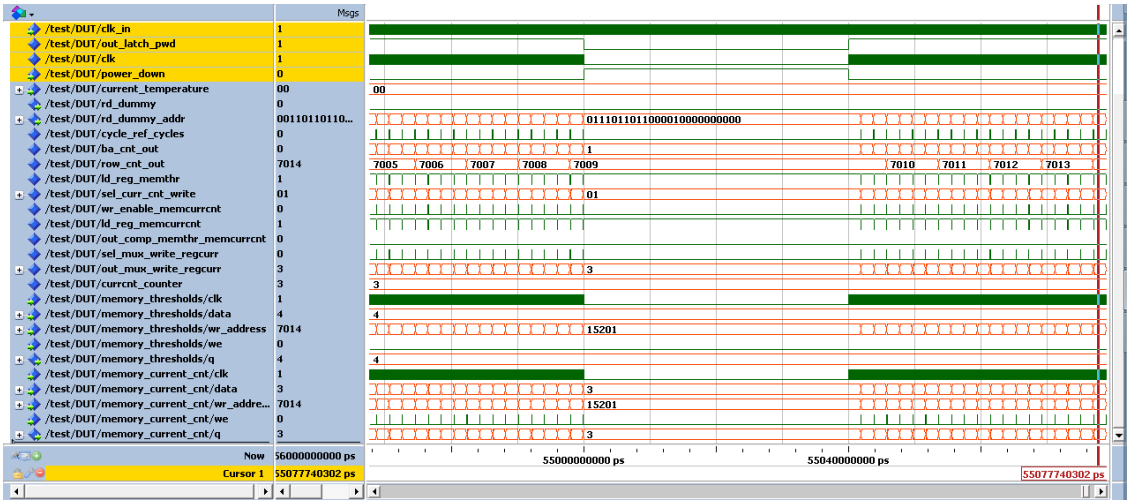


Figure 5.16: Controller architecture simulation in power-down mode

be reported in following results analysis chapter 6.

Chapter 6

Results analysis

In this section, some computations about the SDRAM power consumptions and area occupation will be conducted. As mentioned, from this designed controller architecture it is expected to save power and latency, especially due to the removed *Auto-Refresh* feature, while the area occupied is necessarily higher due to the retention times storage. For what concerns the performances of the controller itself, it is quite useless to test for the maximum frequency since the controller works in parallel with the SDRAM when issues *RAS-only refreshes*, so it could be useful, instead, to have an idea of the reduced response latency obtained with this implementation. Finally, a validation simulation will be reported where, in the worst case situation, the memory is left idle refreshing with this controller architecture: after some time, the SDRAM will be entirely read to see if it is able to retain data through the profiled retention times with minimum errors.

6.1 Power estimation

To estimate the power consumptions, the technical note details [15] of how SDRAM consumes power have been followed. With these calculations, one can estimate the SDRAM consumptions in a given system by simply referring to the datasheet. For the power calculations, the most important parameters and commands sent to the memory will be considered, that are:

- ACT $\longrightarrow$ Active
- BL $\longrightarrow$ Length of burst
- PRE $\longrightarrow$ Precharge
- READ $\longrightarrow$ Reading
- REF $\longrightarrow$ Refresh

- WRITE $\longrightarrow$ Writing

All the operations of the SDRAM are controlled by a clock enable signal. Whenever CKE is de-asserted, the input buffers to the memory are turned off and the SDRAM enters in power-down mode. In order to correctly issue commands, clock enable must be asserted and then the memory can decode commands and addresses inside the memory logic. To better understand which portions of the SDRAM consume power whenever a command is sent, the block diagram of the used memory will be showed taken from its datasheet [7]. During an active command the row content is brought from the memory array to the sense amplifiers of the selected bank: all the blocks shown in Figure 6.1 consume power. The data in the sense amplifiers stay there until a precharge command is issued: this action rewrites the data in the same cells array exactly like a dummy reading refresh performs. If a precharge command is not issued, the SDRAM can execute readings and writings to the same opened bank. A RD requires to place on the bus the correct column address to be accessed in the same row of the array whose content is still stored in the *row buffer* of sense amplifiers. The data is redirected to the DATA OUT BUFFER and then it is available on DQ pins: so in this step, only the output buffers and input/output (I/O) circuitry are used.
In a WRITE operation data spreads in the opposite direction: the data placed on the bus is stored in the DATA IN BUFFER and brought to the I/O structure before being sensed on the bitlines and written in the selected location. To perform power computations, the values of the IDDi currents both for stand-by and power-down mode are needed and these values can be easily obtained from the datasheet. Although the power computations performed in the technical notes are referred to a DDR3 SDRAM, the way in which these contributes are computed is exactly the same. Here, for completeness, the followed methodology reported in [15] is listed:

1. Compute the power components from datasheet.
2. Scale the power according to system command scheduling.
3. Scale the power to the used supply voltage and frequency.
4. Sum all the components to compute the total power.

When the memory is in power-down state and all banks are in a precharge state, the current that flows is referred to as IDD2P. If the memory is in active power-down where a bank is open, the current consumed is IDD3P. In active states, the memory is performing requests sent by the user and the power consumption rises since all the blocks are involved in the operation. The current consumed when all banks are precharged is referred to as IDD2N while if any bank is opened the current consumed is given by IDD3N. These values, taken from the datasheet for the used configuration and for the selected speed grade (corresponding to a clock frequency

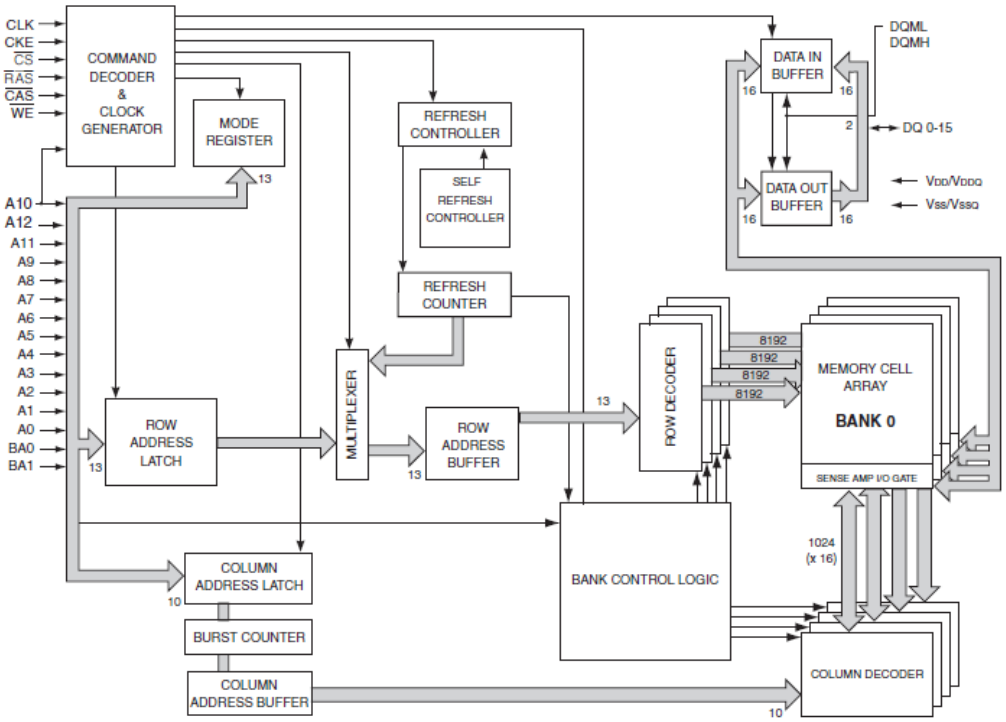


Figure 6.1: SDRAM block diagram (for 8MX16X4 banks configuration) [7]

of 143 MHz), are reported in Table 6.1.

To compute the power consumed by the SDRAM operating in these two different

Current	Value (mA)
IDD2P	8
IDD3P	15
IDD2N	35
IDD3N	40

Table 6.1: Power-down and stand-by SDRAM currents

conditions, it is necessary to multiply the current by the applied VDD voltage. Here are reported the datasheet power contributes due to these currents:

$$\begin{aligned} Pds(PRE\_PDN) &= 8\,mA * 3.3\,V = 26.4\,mW \\ Pds(PRE\_STBY) &= 35\,mA * 3.3\,V = 115.5\,mW \\ Pds(ACT\_PDN) &= 15\,mA * 3.3\,V = 49.5\,mW \\ Pds(ACT\_STBY) &= 40\,mA * 3.3\,V = 132\,mW \end{aligned} \tag{6.1}$$

The background power contribute is always consumed by the SDRAM according to the condition in which the memory is working. These contributes have to be averaged considering how much time the memory spends in each of them and this ratio is determined providing to know the time in which the DRAM banks are all precharged rather than one of them is activated. So, as the notes suggest [15], these following information are needed to perform computations:

- BNK_PRE(%) : Time percentage all banks precharged
- CKE_LO_PRE(%) : Time percentage of power-down bank in precharge state
- CKE_LO_ACT(%) : Time percentage of power-down bank in active state

So the actual scheduled power is determined from the background one taking into account the system usage percentages. Hence:

$$\begin{aligned} Psch(PRE\_PDN) &= Pds(PRE\_PDN) * BNK\_PRE(\%) * \\ &\quad * CKE\_LO\_PRE(\%) \\ Psch(PRE\_STBY) &= Pds(PRE\_STBY) * BNK\_PRE(\%) * \\ &\quad * [1 - CKE\_LO\_PRE(\%)] \\ Psch(ACT\_PDN) &= Pds(ACT\_PDN) * [1 - BNK\_PRE(\%)] * \\ &\quad * CKE\_LO\_ACT(\%) \\ Psch(ACT\_STBY) &= Pds(ACT\_STBY) * [1 - BNK\_PRE(\%)] * \\ &\quad * [1 - CKE\_LO\_ACT(\%)] \end{aligned} \tag{6.2}$$

What about the active power? When an active command is issued, the power consumed by the memory is pretty much high due to the current needed to interpret the command and the address. Datasheet current is referred to as IDD0 or IDD1 as the operating current in this case, that is averaged over the tRC period between successive commands. For the used SDRAM, this current refers to one bank active condition, *CAS Latency* (CL) of 3 and *Burst Length* (BL) of 1. An example of the IDD0 current profile for a given DDR3 DRAM taken from notes [15] is reported in Figure 6.2.

The consumed background portion, represented by the orange line, is always present and refers to the previously computed power consumptions, in stand-by

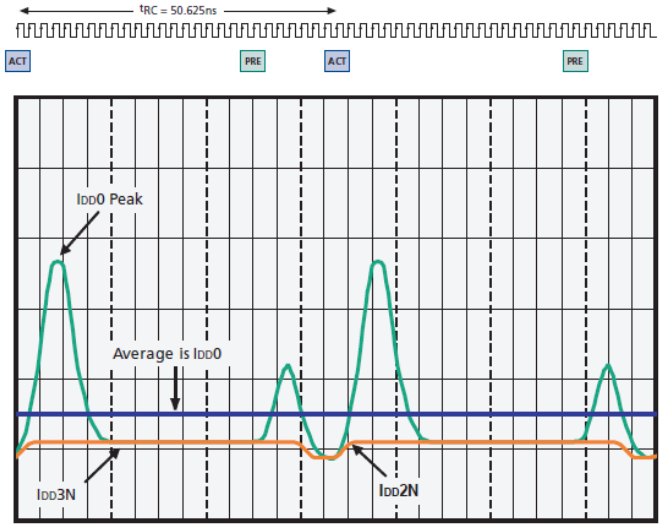


Figure 6.2: Typical IDD0/IDD1 current profile [15]

mode due to IDD3N (active) or IDD2N (precharge). In order to find the power due to active or precharge commands for the used SDRAM, this current value has to be subtracted from IDD1 and multiplied by the supply voltage as follows:

$$Pds(ACT) = \left[140\,mA - \frac{40\,mA * 37\,ns + 35\,mA * (60-37)\,ns}{60\,ns}\right] * 3.3\,V = \\ = 336.33\,mW \tag{6.3}$$

This corresponds to the worst case when working at tRCmin as time interval. In a real case an active command is not issued every tRCmin and so to obtain the correct Psch(ACT) is necessary to derate this value to the average value of row-to-row activation time delay. In the notes it is referred to as tRRDsch, where tRRD is the command period from activating a bank to activating a different bank. Supposing to use the SDRAM with a row-to-row activating time higher than tRC, let's say tRRDsch = 75 ns, then the actual value is computed:

$$Psch(ACT) = 336.33\,mW * \frac{60\,ns}{75\,ns} = 288.28\,mW \tag{6.4}$$

So in a true scheduling the Psch(ACT) is reduced. Clearly, if more banks are opened at a time, the scheduled time between active rows could be very small and the active power can grow further.

Whenever a bank is opened, it can be scheduled with a WRITE operation. The current that is consumed when all banks are active, a BL equal to 4 is selected

and with a CL equal to 3, is referenced in the datasheet to as IDD4. Again, as for active power, to obtain only the power associated with the WRITE operation, the stand-by current IDD3N has to be removed. This contribute is computed as:

$$Pds(WR) = (160\,mA - 40\,mA) * 3.3\,V = 396\,mW \tag{6.5}$$

Then this value has to be derated considering the actual writings scheduling. It is referred to as WRsch(%), standing for the ratio between the cycles the write data is on the bus with respect to the cycles between successive active requests to different banks. Reporting the computation from the technical notes, supposing to have an average of 36 cycles between active commands and the data on the bus for 8 cycles:

$$WRsch(\%) = \frac{no.\ of\ writing\ cycles}{no.\ of\ active\ cycles} \simeq 22\,\% \tag{6.6}$$

Then the scheduled power for writings operation can be computed:

$$Psch(WR) \;=\; 396\,mW * 22.22\,\% \;=\; 88\,mW \tag{6.7}$$

As it is computed for a BL = 4, if a different length is selected, then it is suggested to multiply the scheduled power by a factor equal to (4/BL) to account for a different length of the burst operation.

During a READ operation the power consumed is quite similar to the one needed for a WRITE one whenever a row is activated. In fact, as showed in the notes, the profile of the current that flows in this case is almost identical to the writing one. The power contribute taken from datasheet for a READ operation is computed as follows:

$$Pds(RD) = (160\,mA - 40\,mA) * 3.3\,V = 396\,mW \tag{6.8}$$

Also in this case the actual scheduling of RD operations has to be used to derate the previous value to take into account for real bandwidth carried out in a read operation. It is specified as RDsch(%) standing, as for writing one, for the ratio of the cycles which hold data to read on the bus over the total cycles between two active requests. Supposing as well 8 cycles for data on the bus over an average of 32 cycles, RDsch(%) = 25 %. Hence, as for equation 6.7, the actual scheduled power consumed for a READ operation is:

$$Psch(RD) \;=\; 396\,mW * 25\,\% \;=\; 99\,mW \tag{6.9}$$

These last two contributes determine the consumed power for reading and writing operations. However the total power has to consider also the one consumed by output driver that is specified as on-die termination power. As mentioned in the notes, this requirement is not present in the datasheet but however must be taken into account since depends on the given system in which this SDRAM is used and cannot be neglected. Nevertheless this refers mainly to the used resistance values for the on-die termination when interfacing with the SDRAM and this is a configurable parameter for DDR technology on by programming internal registers, while for the used SDR SDRAM this contribute is not considered in the datasheet itself.

Finally, the power needed for refresh operations has to be computed. This is the major component that is interested in this designed controller architecture and this is the last contribute that must be computed for the total SDRAM power. The datasheet indicates both *Auto-Refresh* current, given by IDD5, and distributed *Self-Refresh* current, referred to as IDD6. The current interested in power consumptions is referred to IDD5, whose value and profile is obtained in the worst case condition considering minimum tRFC delay between active or refresh commands and at the same speed grade at which the device is used. As previously stated, in this operation also the stand-by current IDD3N is consumed and so it must be considered to get correct Pds(REF):

$$Pds(REF) = (210\,mA - 40\,mA) * 3.3\,V = 561\,mW \tag{6.10}$$

To get the actual power consumed for refresh operation, it is necessary to consider that the refreshes commands are issued in a refresh cycle window at a time interval given by tREFI, computed as in 1.1 considering the refresh cycle tREF of 64 ms and the number of rows in a bank to refresh (R). So the actual power is computed:

$$Psch(REF) = 561\,mW * \frac{60\,ns}{7.8\,\mu s} = 4.32\,mW \tag{6.11}$$

The power contributes have been conducted according to the worst-case specifications conditions at which the different currents have been profiled. Actually, indeed, the FPGA based-system works with a LVTTL of 3.3 V instead of the maximum voltage of 3.6 V for which the currents have been measured. Moreover, the used clock frequency is generated through a PLL circuit to obtain a frequency of 140 MHz, slightly different from the speed grade at 143 MHz at which the timing parameters have been obtained. So the computed power contributes have to be scaled to the operating conditions.
The power contribute, referred to the used VDD with respect to the maximum VDD specified in the absolute maximum ratings depends on voltage square, on the input

capacitance C and on the frequency f. Hence, the power derated to the used voltage is:

$$Psys = Psch * \left(\frac{applied\ VDD}{max\ VDD} \right)^2 \tag{6.12}$$

Also a frequency scaling is needed for the previously computed power contributes that change with frequency.
Active power components need to be scaled while contributes in which the memory is in power-down mode don't need to be scaled because of disabled clock in this condition. Scheduled power for refresh operations does not need to be derated since it refers to the time period between active commands and not to frequency. The frequency derated power, considering the linear dependence of the power with the frequency itself, is computed as:

$$Psys = Psch * \left(\frac{used\ frequency}{specified\ frequency} \right) \tag{6.13}$$

The specified frequency refers to the one at which currents profiles have been obtained. The conditions of the test, as indicated in the datasheet, refer also to a rate that is minimum for a given latency specified by the CAS parameter, especially for the operating current IDD0/IDD1 whose profile has been showed previously.
When all the power contributes have been derated for both actual frequency and supply voltage, the final SDRAM power for any usage condition is computed summing up all the components. Also the power due to the on-die termination and the power related to the data bus DQ have to be added according to the system in which the SDRAM is used. To avoid to compute all these values, Micron® has provided an available worksheet [16] to automatically obtain the power consumptions. The spreadsheet allows to specify the speed grade for the actual frequency and latency for CAS parameter, the number of data strobes on the bus per DRAM and the memory density. After inserting all the parameters, timings and currents over all, then the usage conditions must be provided, such as system supply voltage, frequency and the percentages of usage conditions. The output load has to be specified as well, to automatically compute the power for output or termination on the DRAM when the memory itself drives the bus (for a READ operation). This power contribute per DQ must be multiplied by the number of strobes on the device. To compute tRRDsch, the technical notes take into account a new entry called "PageHit(%)". The rate of this value corresponds to the amount of readings and writings operations successfully performed to a row which already was read or written previously over the number of total writings and readings commands performed on that row. This depends on the amount of accesses performed on every single location, hence is mainly due to the running application. Notebook

applications and desktop as well experience high percentages of this parameter, while server applications demonstrate to have low percentages experiencing low hit rates to the same page location. The correct equation for tRRDsch [15] is reported in the following:

$$tRRDsch = \left[\frac{BL/2}{(RDsch\% + WRsch\%) * fclk}\right] /(1 - PageHit\%) \tag{6.14}$$

Then an example of the usage of the SDRAM in a system environment is provided. The memory integrated circuit on the DE1-SoC development board has one rank only, then RD and WR percentage of termination scheduling is null since always the same rank is used. Data bus is fixed to 16 bits, as in the used configuration for the realized FPGA-based architecture and the controller issues operations only to this unique rank. Taking the data example for a mobile sample application from the notes, where two ranks of DRAM are used, the total utilization of data bus is set to 80 %, divided in the percentage of cycles which are yielding data from the DRAM (50 % of the bandwidth for reading operations) and the percentage of cycles which are writing to the DRAM (30 % of the bandwidth for writing operations). Since the memory hierarchy has two ranks, the bandwidth is supposed to be equally distributed for each of them. To have such a high value of bandwidth, a 50 % hit rate is considered by the example, and this value can be acceptable if one considers the latency added by the refresh overhead that causes lots of row misses. Moreover, in this first case power-down condition is never entered, hence the percentage of time that all banks of the DRAM are precharged is assumed to be 20 % of the total operating time. According to these chosen values and considering a maximum BL of 4, average tRRDsch for successive active commands is equal to 71.43 ns.
In Figures 6.3 and 6.4, the "Device Spec" for the used SDRAM and the "Usage Conditions" tabs of the worksheet [16] are showed.

Provided all the information, the single power contributes are reported in the "Power Calcs" tab, derated to the used conditions. The results are shown in Figure 6.5.

The "Summary" tab, reported in Figure 6.6, shows the final power consumption contributes plotted in a bar graph.

The SDRAM rank consumes 130.7 mW background power (including the refresh power too), 182.9 mW activate power and 294.8 mW for write and read operations, including data bus power consumption. Hence, the total SDR SDRAM power consumption in the system environment is about 608.4 mW and this value has to be multiplied by the number of DRAM ranks to obtain the total power consumed by the memory system. As visible, the refresh power P(REF) corresponds to 12.1 mW: in the designed controller the power consumed by the *Auto-Refresh* is null since this command is never issued, so the total background power is reduced to 118.6 mW, having so background power savings for 9.2 %. This is a low density

SDRAM Configuration and Data Sheet Parameters

DRAM Density	512	M
Number of DQs per DRAM	16	
Speed Grade	-7E	

Parameter	Condition	Value	Units
	Maximum Vcc	3,6	V
	Minimum Vcc	3	V
IDD0	Maximum Active Current (calculated)	112	mA
IDD1	Maximum Operating Current	140	mA
IDD2	Maximum Precharge Power-Down Standby Current	8	mA
IDD3	Maximum Active Standby Current	40	mA
IDD4	Maximum Read Burst Current	160	mA
IDD6	Maximum Distributed Refresh Current	12	mA
	^{t}CK for current measurements (see current notes)	7	ns
^{t}RRD	Minimum activate-to-activate timing (different bank)	14	ns
^{t}RC	Minimum activate-to-activate timing (same bank)	60	ns
	Minimum ^{t}CK cycle rate	7	ns

Figure 6.3: SDRAM device specifications [16]

DRAM Usage Conditions in the System Environment

System Vcc	3,3	V	
System CK Frequency	140	MHz	
Output Load in System	50	pF	
Percentage of time that all banks on the DRAM are in a precharged state	20%		
The percentage of the all bank precharge time for which CKE is held LOW	0%		
The percentage of the at least one bank active time for which CKE is held LOW	0%		
The average time between ACT commands to this DRAM (includes ACT to same or different banks in the same DRAM device)	71,43	ns	
The percentage of clock cycles which are outputting read data from the DRAM	25%		
The percentage of clock cycles which are inputting write data to the DRAM	15%		

Figure 6.4: SDRAM usage conditions [16]

memory, but in a modern DRAM chip the power consumed for refresh operations requires a higher current, hence the power savings are considerable. Due to the

System Derating for 512Mb SDRAM (x16, -7E Speed Grade)

Power is calculated after after scaling for VDD. Next, some powers are scaled for frequency effects to derate from test condition to use condition. Finally, these numbers are scaled for actual system usage. All parameters are calculated and require no user input.

		Worst-Case Power Based on Data Sheet	Power Derated for System Usage Conditions Input Into this Model
p(PRE_PDN)	Background power used during precharge power-down	28,8 mW	0,0 mW
p(PRE_STBY)	Background power used during idle standby	144,0 mW	28,8 mW
p(ACT_PDN)	Background power used during active power-down	28,8 mW	0,0 mW
p(ACT_STBY)	Background power used during active standby	144,0 mW	115,2 mW
p(REF)	Background power to complete refreshes	14,4 mW	14,4 mW
p(ACT)	DRAM power for ACT/PRE commands	259,2 mW	217,7 mW
p(WR)	DRAM write power	432,0 mW	64,8 mW
p(RD)	DRAM read power	432,0 mW	108,0 mW
p(DQ)	DQ output power	740,6 mW	185,1 mW

	Power Scaled for Actual System CK Frequency and VCC
P(PRE_PDN)	0,0 mW
p(PRE_STBY)	23,7 mW
P(ACT_PDN)	0,0 mW
P(ACT_STBY)	94,9 mW
P(REF)	12,1 mW
P(ACT)	182,9 mW
P(WR)	53,4 mW
P(RD)	88,9 mW
P(DQ)	152,5 mW

Total DRAM Power Consumed ≥ **734,1 mW** **608,4 mW**

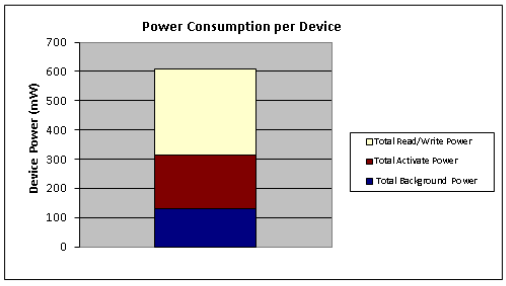
Figure 6.5: Power computations derated to the system conditions [16]

512Mb SDRAM with 16 DQs and a -7E Speed Grade

System is operating at 140 MHz at VCC = 3,3V. Read bandwidth is 70MB/s with write bandwidth of 42 MB/s. The data bus is idle 60% of the time. ACT commands are separated by 71,43ns on average. All parameters are calculated and require no user input.

Power Consumption Summary

P(PRE_PDN)	0,0	mW
P(PRE_STBY)	23,7	mW
P(ACT_PDN)	0,0	mW
P(ACT_STBY)	94,9	mW
P(REF)	12,1	mW
Total Background Power	**130,7**	**mW**
P(ACT)	182,9	mW
Total Activate Power	**182,9**	**mW**
P(WR)	53,4	mW
P(RD)	88,9	mW
P(DQ)	152,5	mW
Total Read/Write Power	**294,8**	**mW**
Total SDR SDRAM Power	608,4	mW



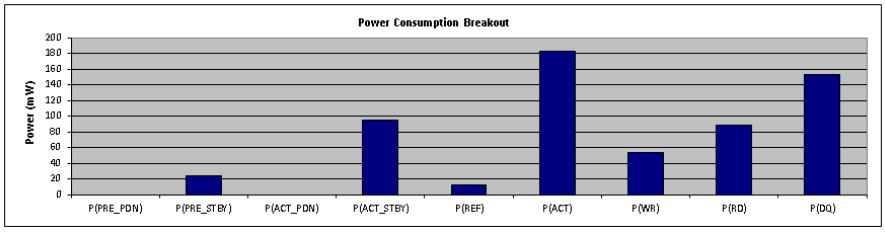


Figure 6.6: Power consumption summary overview [16]

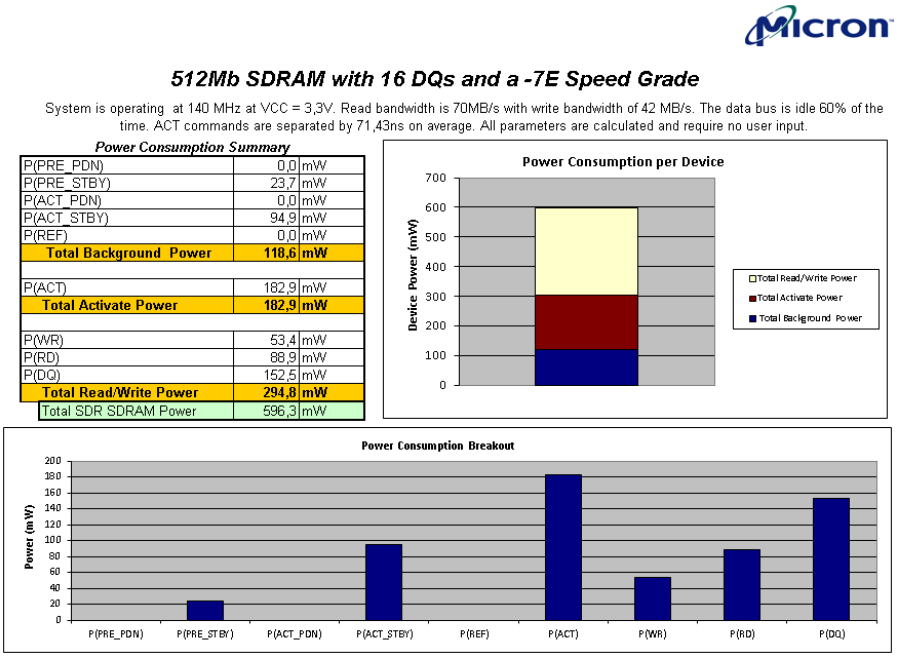


Figure 6.7: Power consumption summary overview without refresh power [16]

long retention times, the power for activation is quite the same since row-by-row refreshes can be seen as accesses for read bandwidth and it is not too much affected considering that almost the 99 % of rows are activated only at the longer retention time as seen in profiling simulations, where the maximum chosen one was 4.096 s: this means that the rows are almost all activated one time every 64*tREF, which is quite negligible in terms of power. Finally, in Figure 6.8, are reported the power consumptions for the case in which the SDRAM enters in power-down mode for a certain time period. To compute power contributes in this situation, it is necessary to consider the equation 6.2 for scheduled power-down mode power in precharge and active conditions. In the analyzed application case, the rank of the used SDRAM is supposed to perform active operations in the 40 % of time usage while in the remaining 60 % of time the memory is left idle. Then, the time percentage in which all the banks are precharged is set to 20 % of the total idle time, meaning that one third of the total idle time the banks are precharged. Now, assuming that of this 20 % of the precharge state time in which no active operations are performed, 15 % of the time has at least one bank active (*CKE_LO_PRE(%)*) and remaining 5 % has all banks precharged (*CKE_LO_ACT(%)*) for the time in which clock enable signal is de-asserted, the power consumptions considering power-done mode condition are summarized in Figure 6.8.

As expected, background power is reduced since the currents flowing in precharge power-down (IDD2P) and active power-down (IDD3P) are lower as reported in

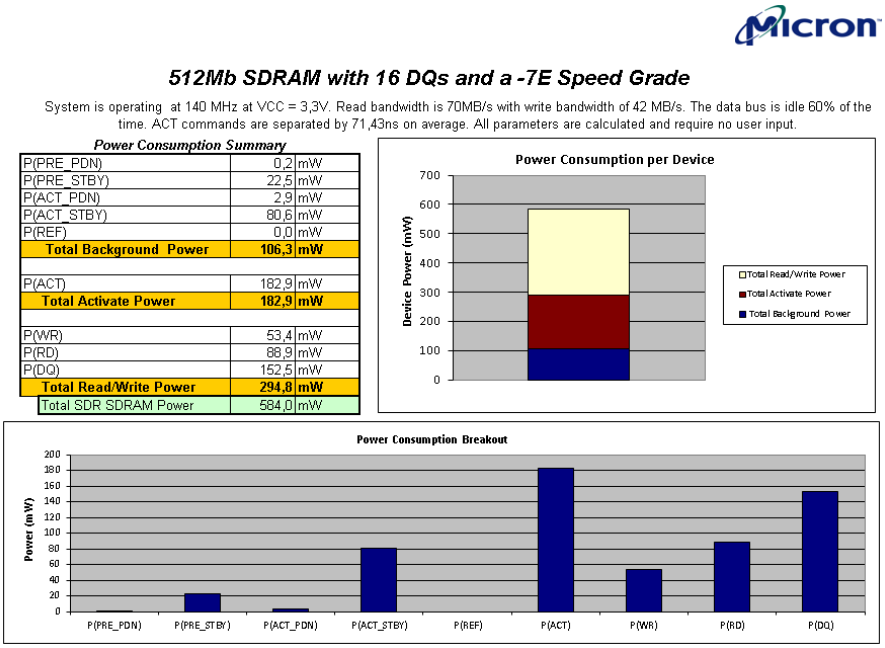


Figure 6.8: Power consumption summary overview with power-down mode [16]

Table 6.1. Hence, total background power is reduced from 118.6 mW to 106.3 mW, so having power savings for 10.4 % with respect to previous analyzed conditions where power-down mode is never entered.

6.2 Area estimation

Concerning the area occupied by the entire architecture, an estimation of the area on the used Cyclone® V FPGA device family will be provided. In the minimized datapath structure showed in Figure 5.11, only the most important blocks have been reported: used counters and some combinational logic are not displayed for a better clarity. Nevertheless, it is quite important to understand which is the overhead, in memory bits, of the two SRAMs used to store the thresholds and current counters. In the actual configuration with 8-bits thresholds, that define a retention times range from 64 ms to about 16 s, the memory occupation depends on:

- ADDRESS_WIDTH = BANK_WIDTH + ROW_WIDTH = 13 bits
- DATA_WIDTH = NBITS_THRESHOLDS = 8 bits

Hence each memory has 32768 rows (all the SDRAM ones) by 8 bits each, covering so 32 KB. Considering the presence of both thresholds and current counters

memories, the total storage overhead corresponds to 64 KB.
Then, if one considers the area occupied by one of the banks of the used SDRAM, that has in the configuration 8192 rows by 1K page and a data parallelism of 16 bits, the area in memory bits corresponds to 16 MB: hence, the thresholds storage overhead corresponds to only the 0.2 % of one bank area, that is quite negligible. If compared to the total SDRAM area, the overhead corresponds to a paltry percentage of 0.05 %. The advantages of this architecture come with modern DRAM chips with higher densities, where the thresholds storage contribute is even more reduced compared to the entire DRAM area. The additional contribute is given by the controller usage for skipping refreshes, where an additional SRAM has to store the addresses of the rows that must not be refreshed at all: the overhead added from this feature depends on the application and from the user choices according to the system requirements. To give an idea, if the user selects 1K addresses to not refresh, the additional overhead is equal to 15 Kb, that is still negligible. Then it is advisable to choose, in this configuration, a limited number of rows compared to the total SDRAM capacity to avoid incurring in considerable overheads.
In order to provide more quantitative results of the area occupied in FPGA, the *Fitter Resource Usage Summary* report has been extracted from Quartus® synthesizer, which displays the percentage of a given resource used in the design. This report shows a detailed analysis of the logic utilization based on Adaptive Logic Module (ALM) usage, that is the basic building block of these device families and is designed to have best performances and resources usage. In practice, the logic utilization to build the design refers to a portion of the ALMs available on the device (ALMs needed over total ALMs) and this is the used metric to find the area occupied on the chip in FPGA.
The report summarizes also the combinational ALUT usage for logic: an Adaptive Look-Up Table (ALUT) represents a logical construct which can be realized by the combinational logic of an Adaptive Logic Module (ALM) in the device. Moreover, in an ALM the Fitter can select from different paths to control a register input signal. As mentioned in the guide [17], there are paths that are dedicated which bypass the logic defined by the LUTs; direct paths, instead, move across the LUTs. The way in which the path is chosen depends on the critical paths in the designed structure: in some cases it might be better to have a register driven using a direct path if the logic given by the LUT is not utilized for another one in the implementation. Finally, the dedicated logic registers entry is reported: this refers to number of registers in the design realized with the same logic of ALMs. To understand the area occupied by the designed controller architecture, a comparison between the area of the basic SDRAM core controller discussed in chapter 2 and the one of the final implementation with the optimizations reported in chapter 5 will be displayed. In the following figures, Fitter Summary and Resource Usage obtained from the Compilation Report for the two cases are explored.

Fitter Summary	
Fitter Status	Successful - Sun Mar 24 13:02:33 2019
Quartus II 64-Bit Version	14.0.0 Build 200 06/17/2014 SJ Web Edition
Revision Name	top_level
Top-level Entity Name	sdram_controller_del
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	83 / 32,070 (< 1 %)
Total registers	84
Total pins	153 / 457 (33 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0

Figure 6.9: Fitter Summary for the basic controller

Fitter Resource Usage Summary		
Resource	Usage	%
Logic utilization (ALMs needed / total ALMs on device)	83 / 32,070	< 1 %
ALMs needed [=A-B+C]	83	
[A] ALMs used in final placement [=a+b+c+d]	84 / 32,070	< 1 %
[a] ALMs used for LUT logic and registers	39	
[b] ALMs used for LUT logic	42	
[c] ALMs used for registers	3	
[d] ALMs used for memory (up to half of total ALMs)	0	
[B] Estimate of ALMs recoverable by dense packing	5 / 32,070	< 1 %
[C] Estimate of ALMs unavailable [=a+b+c+d]	4 / 32,070	< 1 %
[a] Due to location constrained logic	0	
[b] Due to LAB-wide signal conflicts	0	
[c] Due to LAB input limits	4	
[d] Due to virtual I/Os	0	
Difficulty packing design	Low	
Total LABs: partially or completely used	10 / 3,207	< 1 %
-- Logic LABs	10	
-- Memory LABs (up to half of total LABs)	0	
Combinational ALUT usage for logic	132	
-- 7 input functions	3	
-- 6 input functions	25	
-- 5 input functions	43	
-- 4 input functions	21	
-- <=3 input functions	40	
Combinational ALUT usage for route-throughs	0	
Dedicated logic registers	84	
-- By type:		
-- Primary logic registers	84 / 64,140	< 1 %
-- Secondary logic registers	0 / 64,140	0 %
-- By function:		
-- Design implementation registers	84	
-- Routing optimization registers	0	

Figure 6.10: Fitter Resource Usage Summary for the basic controller

```
; Fitter Summary                                                                ;
+---------------------------------+-----------------------------------------+
; Fitter Status                   ; Successful - Sun Mar 24 13:16:52 2019      ;
; Quartus II 64-Bit Version       ; 14.0.0 Build 200 06/17/2014 SJ Web Edition ;
; Revision Name                   ; top_level                                 ;
; Top-level Entity Name           ; advanced_sdram_controller                 ;
; Family                          ; Cyclone V                                 ;
; Device                          ; 5CSEMA5F31C6                              ;
; Timing Models                   ; Final                                     ;
; Logic utilization (in ALMs)     ; 2,589 / 32,070 ( 8 % )                    ;
; Total registers                 ; 4676                                      ;
; Total pins                      ; 161 / 457 ( 35 % )                        ;
; Total virtual pins              ; 0                                         ;
; Total block memory bits         ; 458,752 / 4,065,280 ( 11 % )              ;
; Total DSP Blocks                ; 0 / 87 ( 0 % )                            ;
; Total HSSI RX PCSs              ; 0                                         ;
; Total HSSI PMA RX Deserializers ; 0                                         ;
; Total HSSI TX PCSs              ; 0                                         ;
; Total HSSI PMA TX Serializers   ; 0                                         ;
```

Figure 6.11: Fitter Summary for the advanced controller

```
; Fitter Resource Usage Summary                                                                ;
+---------------------------------------------------------------------+-----------------------+-------+
; Resource                                                            ; Usage                 ; %     ;
+---------------------------------------------------------------------+-----------------------+-------+
; Logic utilization (ALMs needed / total ALMs on device)              ; 2,589 / 32,070        ; 8 %   ;
; ALMs needed [=A-B+C]                                                ; 2,589                 ;       ;
;     [A] ALMs used in final placement [=a+b+c+d]                     ; 3,249 / 32,070        ; 10 %  ;
;         [a] ALMs used for LUT logic and registers                   ; 757                   ;       ;
;         [b] ALMs used for LUT logic                                 ; 1,163                 ;       ;
;         [c] ALMs used for registers                                 ; 1,329                 ;       ;
;         [d] ALMs used for memory (up to half of total ALMs)         ; 0                     ;       ;
;     [B] Estimate of ALMs recoverable by dense packing               ; 677 / 32,070          ; 2 %   ;
;     [C] Estimate of ALMs unavailable [=a+b+c+d]                     ; 17 / 32,070           ; < 1 % ;
;         [a] Due to location constrained logic                       ; 0                     ;       ;
;         [b] Due to LAB-wide signal conflicts                        ; 16                    ;       ;
;         [c] Due to LAB input limits                                 ; 1                     ;       ;
;         [d] Due to virtual I/Os                                     ; 0                     ;       ;
;                                                                     ;                       ;       ;
; Difficulty packing design                                           ; Low                   ;       ;
;                                                                     ;                       ;       ;
; Total LABs:  partially or completely used                           ; 413 / 3,207           ; 13 %  ;
;     -- Logic LABs                                                   ; 413                   ;       ;
;     -- Memory LABs (up to half of total LABs)                       ; 0                     ;       ;
;                                                                     ;                       ;       ;
; Combinational ALUT usage for logic                                  ; 2,833                 ;       ;
;     -- 7 input functions                                            ; 12                    ;       ;
;     -- 6 input functions                                            ; 1,530                 ;       ;
;     -- 5 input functions                                            ; 171                   ;       ;
;     -- 4 input functions                                            ; 454                   ;       ;
;     -- <=3 input functions                                          ; 666                   ;       ;
; Combinational ALUT usage for route-throughs                         ; 1,323                 ;       ;
; Dedicated logic registers                                           ; 4,676                 ;       ;
;     -- By type:                                                     ;                       ;       ;
;         -- Primary logic registers                                  ; 4,171 / 64,140         ; 7 %   ;
;         -- Secondary logic registers                                ; 505 / 64,140           ; < 1 % ;
;     -- By function:                                                 ;                       ;       ;
;         -- Design implementation registers                         ; 4,597                 ;       ;
;         -- Routing optimization registers                           ; 79                    ;       ;
```

Figure 6.12: Fitter Resource Usage Summary for the advanced controller

Looking at the utilization in ALMs, the final advanced controller occupies only 7 % more logic than the initial basic controller on the device chip. The number of dedicated registers is higher as well, while the number of the used pins is almost the same since both the implementations interface, in principle, with the SDRAM pins only. The significant result is given by the total block memory bits entry: the final controller architecture has 11 % more of total block memory bits on the device over the basic controller, as a result of the usage of thresholds and current counters memories (32768x8 configuration each) and of the SRAM for skipping refreshes (256x15 configuration). No synthesis has been performed through Synopsys® design compiler or similar, since the controller architecture would reside on the FPGA chip in a more complex system where the SDRAM is used and it is not needed to realize a stand alone integrated circuit for such implementation.

6.3 System performances estimation

When this controller architecture is brought to a real system, the advantages come from the removed *Auto-Refresh* feature in favor of a reduced rate of refreshing, exploiting retention times distribution. To understand the increase of performances due to the usage of *RAS-only refreshes* together with actual retention times, it would be necessary to execute several benchmarks that stress the memory usage in different conditions, when trying to replicate real system applications. Unfortunately, simulators that support SDR SDRAMs are difficult to find today, so a future work could be involved in replacing the SDR SDRAM core controller with a DDR one, without modifying the surrounding realized architecture that would continue to work. However, to give an idea of the improvements some considerations will be done.
Supposing to choose as refresh rate, after the profiling procedure, the minimum found retention time at room temperature of 1.024 s for all the rows to reduce VRT state changes and to have a more reliable refresh process, this means that in 16 tREF refresh cycles of 64 ms only in the very last one all the rows will be refreshed. So, considering the refresh command duration, that in *Auto-Refresh* takes 60 ns (tRFCmin) while in *RAS-only refresh* takes 52 ns (minimum activation time tRAS plus minimum precharge time tRP), the total time that the SDRAM spends refreshing for the two cases is reported in the following Table 6.2, where the values are derated to the actual clock frequency of 140 MHz.

A controller that issues *RAS-only refreshes* saves 6.32 ms in 16 refresh cycles,

Refresh type	Time spent (ms)
Auto-Refresh	8.43
RAS-only refresh	2.11
Improved RAS-only refresh	1.87

Table 6.2: Refresh types latency comparisons in 16 tREF cycles

that corresponds to a refresh time saving of about 75 %. All this time that the SDRAM does not spend in refreshing is time gained in serving user requests, then this results in a reduced access latency and improved performances. Although the tRAS optimization does not improve the area and not even the power consumptions (for *RAS-only refreshes* the power consumption refers to the activate power, whose accesses are determined by tRRDsch but due to the reduced refresh rate, the average value of tRRDsch is mainly fixed by reading and writing operations according to the system application bandwidth), a reduction of the *Row Access Strobe* delay results in an improvement of the time spent in doing refreshes. In fact, having reduced

tRAS of one clock cycle, that at 140 MHz has fallen from about 42.86 ns to about 35.71 ns, now a refresh command (activate plus precharge) lasts about 57.14 ns with respect to the previous 64.29 ns, providing further refresh overhead reductions. In particular, comparing with the time spent in the same 16 tREF by the other solutions, the total time spent in refreshing corresponds to 1.87 ms providing so 11.4 % more refresh reduction over *RAS-only refresh* with standard timing parameters at the actual clock frequency.
In Figure 6.13 an indicative comparison with other implementations present in literature has been done, in terms of occupied logic area (as % of a DRAM bank area) and refresh reduction.

IMPLEMENTATION	LOGIC AREA (% of DRAM bank area)	REFRESH REDUCTION
VRL (4 bits)	1.85 % for a DRAM bank of size 8192x32	23 % reduction over RGR thanks to low-latency tRFC
VRL-Access	Same as VRL: it's just an optimization on accesses as for our implementation	Average 13 % reduction over VRL and 34 % reduction over RGR across the applications
ORGR	No area overhead (only a modification on tRAS delay)	Up to 45 % w.r.t *Auto-Refresh* and RGR
Our implementation (4 bits thresholds RGR, datasheet value for tRAS)	≈ 32.5 KB (0.2 % for DRAM bank of size 8192x16) working as Selective RGR where 256 rows are not refreshed at all	Average reduction up to 75 % w.r.t *Auto-Refresh* at room temperature and 50 % of reduction in best case of alternated refreshes-accesses

Figure 6.13: Comparison with other implementations [1], [4], [5]

6.4 Validation test

In this section a validation of the controller has been performed. The memory is written with pseudo-random data in the same way done in the profiling step. After a time delay of 4 hours where the memory, in the worst case, is left idle without issuing accesses, the locations have all been read and compared with the data written at the beginning: the comparison determines if the rows are able to retain data with their profiled retention times and, then, if the controller is reliable enough in performing actual retention times-based refreshes. The room temperature has been tracked during the simulation time and possible variations of ± 10 °C have been issued to the controller. The results of the test are reported in Figures 6.14 and 6.15.
The room temperature has been acquired every 5 minutes, for a total of 48 samples

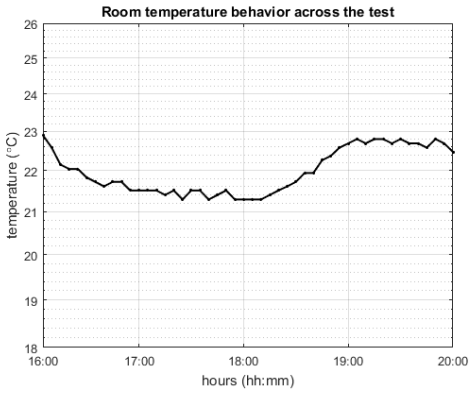


Figure 6.14: Room temperature trend

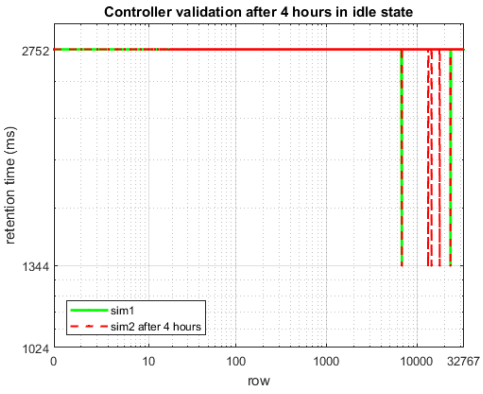


Figure 6.15: Refresh rates distribution

across the simulation time. The temperature has not reached a variation of ± 10 °C, hence no retention times updating has been issued. The chosen retention times for the simulation test are 512 ms, 1024 ms, 1344 ms and 2752 ms.
Figure 6.15 reports the distribution of the retention times obtained after 4 hours by reading back the content of the rows. The green dashed line represents the retention times distribution at the beginning of the test, while the red dashed line reports the retention times distribution after the 4 hours of simulation: the result is that only 3 rows out of 32768 have failed the test, reporting bit failures in some cells. This means that, even in a worst case condition where the memory is left completely idle, the controller is reliable enough at such retention times distribution. This is corroborated by the fact that a typical DRAM cell is affected by VRT especially for retention times over about 3 seconds where the state changes are, instead, very frequent. In conclusion, choosing a correct profiling, the user can finally rely on a safe memory usage through a controller that issues retention times-based refreshes.

Chapter 7

Final conclusions and future work

7.1 Conclusions

The controller architecture is now able to issue *RAS-only refreshes* according to the profiled retention times distribution using the best pattern for the adopted SDRAM. The core controller of the SDR is able to provide single word accesses or burst-based accesses, whose length is configurable. The accesses are handled by the RRM to further improve the refresh reduction, as an access corresponds to fully restore the content in the selected row. Then, the architecture has been modified to include a configurable SRAM to save addresses of some rows to not refresh at all and this could allow to achieve further benefits in terms of refresh overhead reduction and power savings when consented by a specific application. For example the presence of non critical data whose bit failures are not destructive in terms of performances or a high flow of data that are continuously read and written in these locations make refreshes unnecessary. The feature of skipping refreshes is configurable and the number of addresses to save, at the maximum retention time, is configurable as well, providing to work at room temperature where the number of rows at the maximum retention time is pretty much high. Then, temperature variations of $\pm$ 10 °C with respect to the reference profiling room temperature have been handled, where the retention times are halved or doubled accordingly. This implementation choice comes from the need of handling the temperature effects on the retention times distribution but also to avoid weighing operations down. Finally an optimization on the *Row Access Strobe* timing parameter has been conducted [4] to further reduce the refresh overhead in RGR mode and clock gating technique has been applied to the controller whenever the SDRAM enters in power-down mode. Power and area estimations have been provided to broadly understand the power savings and the storage overhead produced by this implementation, which in general

is quite irrelevant. The logic required for the entire implementation does not affect execution critical path, since the refreshes are generated in parallel with the normal functionality of the memory controller and its frequency is however smaller than a processor clock frequency could be.
A final consideration can be done on the challenging problem of VRT. Experimental studies conducted in work [2] have demonstrated that none of the DRAM cells of all of the tested chip families had a retention time lower than 1.5 seconds at a room temperature of 45 °C. The result is that there are lots of cells, and then rows, that could be refreshed at very low rates. The analysis done in this thesis work has demonstrated that, for the four tested retention times, most of the rows can be refreshed at 1.024 s as minimum retention time, then setting thresholds and current counters memory locations to 16. However these results are valid and true for the tested SDRAM on the DE1-SoC development board and they *could* be in part or totally different for another memory: taking into account all these factors and the considerations made until now, the behavior changes from memory to memory depending from the architecture too, thus it is always necessary and strongly recommended to profile the memory first before using it in an attempt to reduce the refresh overhead. This is the reason for which the control has been built in such a way that, to start the row-by-row refresh mode, you have to pass through characterization states before activating the RRM. Although the DPD is bypassed by checking for the worst-case data pattern to apply and taking some precautions in doing the profiling, VRT instead is always present and is a real problem in retention times characterization. Fixed the pattern, the performed simulations have been executed under room temperature variation and cells retention times mainly followed the temperature trend. The FPGA-based architecture has not built for studying VRT but it is obvious that, at fixed temperatures, the effects of VRT on profiling would be remarkable. In work [2], in all the devices that have been tested most of the cells stayed in the high retention time state, in that case equal to 6.2 s. Although the state changes are very marked, as visible in Figure 11 of that work and reported here for clarity in Figure 7.1, it has been found that almost the totality of the cells were found to have a minimum retention time period of about 1.5 s, in agreement with the seven days simulations performed in this work.
Moreover, in the same work [2], it has been found that most cells stayed in high retention states for about 4 hours and only some cells stayed in high retention states for the entire day of the experiment: this means that repeated simulations on more days are needed to perform a correct profiling of the minimum retention time of any cell. That's what has been observed in the performed analysis: apart from temperature variation, when the experiments were conducted for the first four days for a total of about four hours, at intervals of half a hour, it has been observed that the rows almost faithfully followed the temperature variation along the test. Most of them ($\simeq$ 99 %), in fact, stayed in high retention time state of 4.096 s in agreement with what mentioned in the experimental study. On the contrary, in

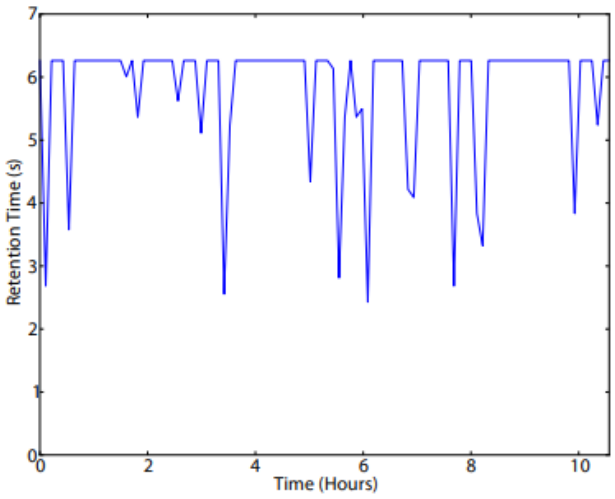


Figure 7.1: Retention time behavior of a typical VRT cell [2]

the last three days simulations of the week performed for about 8 hours a day at intervals of a hour, it was found that for some temperatures equal to ones in the first four days simulations the behavior was in some cases different in terms of the number of rows found in the various retention time states. This means that the effects of VRT are more visible along a day of simulation. However, since these simulations have verified that none of the rows exhibited a retention time lower than 1.024 s, thus setting this value as refresh rate for all the rows could minimize the errors due to VRT.
As a final consideration, one should account that even if this procedure could provide a limited number of errors, a particular application, in which this design can be involved, could require a memory completely free from errors: the use of error correcting codes (ECC) could be a good way to solve the problem and to handle VRT [10], except that this mechanism would require too much energy and capacity during the execution time for DRAMs, making this solution not generally applicable. So, in short, this controller architecture solution has been realized and thought for resilient applications, like video coding ones for example, where a limited number of failures can be accepted provided that they do not affect significantly the performances.

7.2 Future work

The problem encountered in this work mainly concerns the lack of a simulator to properly test the SDR SDRAM controller running some benchmarks. Although it has not been possible to perform such tests, lots of works in literature have

demonstrated that similar architectures that aim to reduce the refresh overhead in DRAMs in general, in the end get huge benefits in terms of performances. In all the cited works different boards have been exploited, in most cases provided with a channel for testing external SO-DIMMs or custom PCB have been realized conformed with standards. In this way different chips from different vendors could be tested and, especially, they could work with modern DDR chip of memories.
Simulations performed on DDR3 from different manufacturers have demonstrated that the refresh reductions are more and more marked in modern memories due to the high density of cells. Refresh overhead is expected to aggravate in next technology nodes due to more rows to refresh at a time and so the power consumption is expected to reach the limits imposed by that technology, making impossible to further scale in frequency. In this context, the idea proposed in this work could help to reduce the problem of refresh by relaxing power constraints and increase frequency, having thus a reduced response latency at the cost of a small loss of performances. Other works, orthogonal to this, have focused on reducing refresh in different manners, like providing VDD scaling or reducing the data bus traffic exploiting different kinds of data compressions.
The advantage of the proposed solution is that is quite general and can be applied to different types of SDRAMs, by simply setting the proper configuration. In fact, apart from the sequence of activate and precharge for *RAS-only refreshes* and the sequence of commands used for determining the minimum tRAS value, the core controller has not been further customized. Surrounding this, the designed controller handles the environment operating conditions and addresses proper commands to the core controller in order to reduce the unnecessary number of refreshes. In this sense, the controller architecture can be modified by replacing the SDRAM core controller of the SDR with the one of a DDR by simply configuring in the proper way also the surrounding top level architecture. In this way simulations can be addressed by launching benchmarks of different types and with different accesses distributions to understand all the benefits given by this solution. As demonstrated by the cited works and stated here, the benefits on DDR SDRAMs are remarkable and we are convinced that same benefits can be hopefully obtained with this implementation when applied to a modern memory device.

Appendix A

User specifications

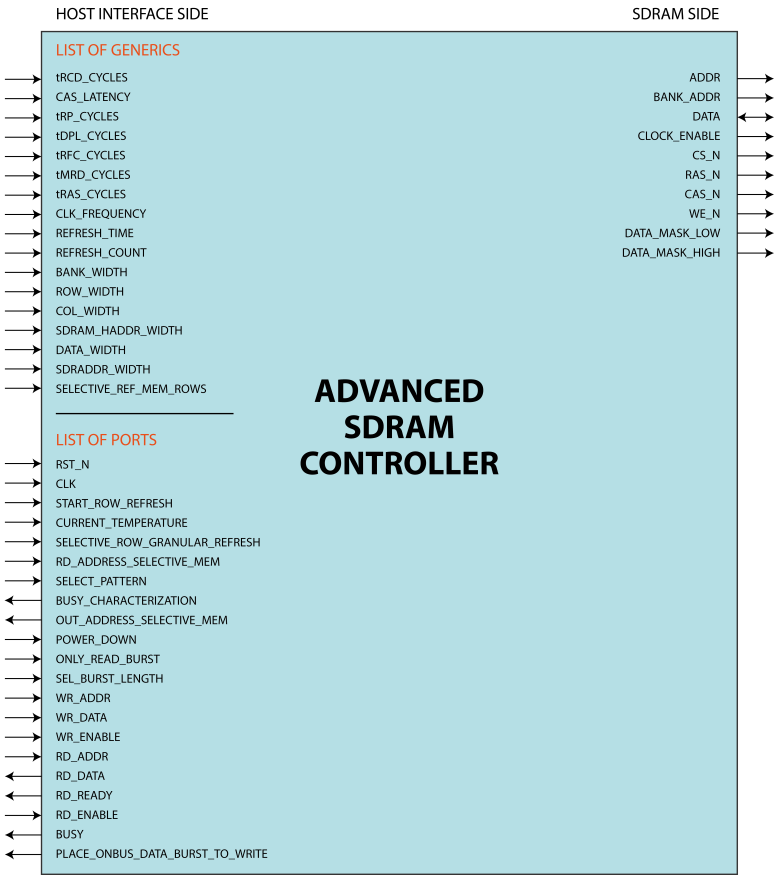


Figure A.1: Advanced SDRAM controller top entity

Figure A.1 shows the top level entity of the designed controller architecture. In the host interface side, in *LIST OF GENERICS* are reported the timing parameters

that the user must set in clock cycles, the used clock frequency (in MHz), the refresh time period (in ms that is 64), the refresh count (number of rows in a bank, here 8192), bank, row and column width as address bits and data width as data bus bits parallelism. Then there are other three parameters:

- SDRAM_HADDR_WIDTH = sum of bank, row and column address bits
- SDRADDR_WIDTH = max[ROW_WIDTH, COL_WIDTH]
- SELECTIVE_REF_MEM_ROWS = # of rows refreshes are skipped to

All these parameters depend on the selected configuration for the SDRAM. For the used memory, here is a setup list of all the generic parameters:

- tRCD_CYCLES = 3
- CAS_LATENCY = 3
- tRP_CYCLES = 3
- tDPL_CYCLES = 2
- tRFC_CYCLES = 9
- tMRD_CYCLES = 2
- tRAS_CYCLES = 6
- CLK_FREQUENCY = 140
- REFRESH_TIME = 64
- REFRESH_COUNT = 8192
- BANK_WIDTH = 2
- ROW_WIDTH = 13
- COL_WIDTH = 10
- SDRAM_HADDR_WIDTH = 25
- DATA_WIDTH = 16
- SDRADDR_WIDTH = 13
- SELECTIVE_REF_MEM_ROWS = 256

Provided the configuration, the SDRAM is ready to perform operations. In *LIST OF PORTS* are reported the signals available to the user. Here is reported the list of the signals used to communicate with the controller:

- *INPUTS*
  - START_ROW_REFRESH
  - CURRENT_TEMPERATURE
  - SELECTIVE_ROW_GRANULAR_REFRESH
  - RD_ADDRESS_SELECTIVE_MEM
  - SELECT_PATTERN
- *OUTPUTS*
  - BUSY_CHARACTERIZATION
  - OUT_ADDRESS_SELECTIVE_MEM

START_ROW_REFRESH asserted by the user triggers the beginning of the SDRAM profiling and the use of the memory in a row-by-row refresh mode.
CURRENT_TEMPERATURE is a 2 bits wide signal that, at reset, has to be set to "00". When the profiling has finished, the memory will be refreshed according to the distribution of retention times, obtained at the current reference room temperature (T_{REF}). As described in section 5.2, the memory controller is able to handle temperature variations at steps of 10 °C from T_{REF} to T_{REF} + 30 °C. The user has to change this signal value whenever an increase or a decrease of about 10 °C is detected, having established that a temperature increase or a decrease of about 10 °C requires retention times halving or doubling respectively (Figure 1.2 taken from work [2]). An increase of 10 °C over TREF (changing the signal value to "01"), causes the halving of the retention times. A further increase of 10 °C over current temperature (changing from "01" to "10") causes the retention times to be one fourth of the starting T_{REF} retention times values. At T_{REF} + 30 °C (from "10" to "11") the retention times are one eighth over T_{REF} ones. At any temperature in the range [T_{REF} + 10 °C, T_{REF} + 30 °C], if the user detects a reduction of the temperature of 10 °C, then can change the signal value at steps of 10 °C where the retention times values are doubled each time and so, passing from "11" to "00" in 3 steps the user can restore the initial retention times values obtained at T_{REF}. The halving is done without any rounding factor, since an integer value is needed and then the truncation towards minus infinite is just fine to provide safe retention times conversions. Note that over T_{REF} + 30 °C the controller does not handle temperature variations anymore and it is necessary to perform again the retention times profiling, since the exponential behavior of retention times with respect to temperature makes the memory less reliable at such high temperatures. Moreover

a temperature change requires a dummy readings cycle, then increasing accesses memory latency: this is one of the main reasons for which only a span of 30 °C has been handled. Note also when starting with "00" at T_{REF} or when returning to this value after successive temperature variations, you cannot double the retention times if the temperature drops 10 °C below T_{REF}: this, clearly, would make the starting retention times characterization useless.
SELECTIVE_ROW_GRANULAR_REFRESH, asserted at reset, tells the memory controller that the user wants to skip refreshes to some rows, as a system application requires it or simply because the application has a set of non critical data whose integrity is not destructive in terms of performances.
RD_ADDRESS_SELECTIVE_MEM is the address signal to point to this memory of not refreshed rows to know where these locations are actually present in the SDRAM.
OUT_ADDRESS_SELECTIVE_MEM corresponds to the data output of this memory and refers to the actual SDRAM row addresses. The reading is asynchronous, hence the output is immediately available.
SELECT_PATTERN is a 2 bits wide signal that allows the user to configure the profiling of the retention times distribution using one among four different patterns:

- "00" $\longrightarrow$ all 1s
- "01" $\longrightarrow$ all 0s
- "10" $\longrightarrow$ checkerboard
- "11" $\longrightarrow$ pseudo-random

The choice, as previously mentioned, stands in the SDRAM architecture complexity and in the type of application that will be run. A quasi dynamic pattern usually provides a good coverage of bit failures but if the memory has a small capacity and the application has a sparse memory filling, a pattern like all 0s could also provide a good result.
Finally, BUSY_CHARACTERIZATION is an output signal warning the user that the memory controller, and hence the SDRAM, is busy performing the retention times characterization.
The following signals of the host interface side, instead, are used to communicate with the SDRAM core controller. Explanatory signals like addresses and data ones to communicate with the memory are skipped for simplicity.

- *INPUTS*
  - POWER_DOWN
  - ONLY_READ_BURST
  - SEL_BURST_LENGTH

- *OUTPUTS*

  - RD_READY
  - BUSY
  - PLACE_ONBUS_DATA_BURST_TO_WRITE

POWER_DOWN, if asserted when SDRAM is idle, puts the memory in power-down mode by suspending the clock. Note that in power-down mode, the rows are not refreshed and any dummy reading command cannot be received, hence causing bit failures. It is the user concern to leave this state as soon as possible, where for the *Auto-Refresh* feature is before tREF elapses, otherwise data integrity could not be guaranteed. Applied clock gating technique suspends clock also for the controller.
ONLY_READ_BURST set to '1' at power-on configures the SDRAM to provide burst readings and single word writings. When set to '0', burst mode is selected for both readings and writings.
SEL_BURST_LENGTH is a 2 bits wide signal to select the burst length in both readings and writings. "00" corresponds to 1 word burst length, "01" to 2 words, "10" to 4 words and "11" to a burst 8 words long.
For the output signals, BUSY signal is asserted to '1' whenever the SDRAM is performing active operations, that are initialization, readings, writings, *Auto-Refreshes* or *RAS-only refreshes*.
RD_READY is asserted to '1' the next clock cycle to which the data is on the bus when a reading operation is performed. When a burst mode is selected, this signal is asserted for all the entire duration of the burst length.
PLACE_ONBUS_DATA_BURST_TO_WRITE is asserted by the memory controller during a write burst operation: when this signal is set to logic '1', by the next clock cycle the user has to place on the bus, in successive clock cycles, the data to be written in memory for as many clock cycles as the burst length. Timings of the usage of these last two signals are omitted here, since they have already been showed in Figures 2.6, 2.7 and 2.8 and during retention times profiling readings step in Figure 4.8.

Bibliography

[1] J.Liu et al., *"RAIDR: Retention-Aware Intelligent DRAM refresh,"* in ISCA-39, 2012.

[2] J.Liu, B.Jaiyem, Y.Kim, C.Wilkerson, and O.Mutlu, *"An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms"*, in Proc. 40th Annu. Int. Symp. Comput. Archit., 2013.

[3] T.Hamamoto, S.Sugiura and S.Sawada, *"On the retention time distribution of dynamic random access memory (DRAM)"*, IEEE TED, vol. 45, no. 6, 1998.

[4] Deepak M. Mathew, Eder F.Zulian, Matthias Jung et al., *"Using Run-Time Reverse-Engineering to Optimize DRAM Refresh"*, MEMSYS 2017, October 2-5, 2017, Alexandria, VA, USA

[5] Anup Das, Hasan Hassan and Onur Mutlu, *"VRL-DRAM: Improving DRAM Performance via Variable Refresh Latency"*, 2018, 55th ACM/ESDA/IEEE Design Automation Conference (DAC)

[6] Mrinmoy Ghosh and Hsien-Hsin S.Lee, *"Smart Refresh: An Enhanced Memory Controller Design for Reducing Energy in Conventional and 3D Die-Stacked DRAMs"*, MICRO 40 2007, Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture, Chicago, Illinois, USA

[7] IS42/45R86400D/16320D/32160DIS42/45S86400D/16320D/32160D: 16Mx32, 32Mx16, 64Mx8 512Mb SDRAM Datasheet, September 2012

[8] Arnab Raha, Soubhagya Sutar, Hrishikesh Jayakumar and Vijay Raghunathan, *"Quality Configurable Approximate DRAM"*, IEEE Transactions on Computers, Vol. 66, No. 7, July 2017

[9] Jan Lucas, Mauricio Alvarez-Mesa, Michael Andersch and Ben Juurlink, *"Sparkk: Quality-Scalable Approximate Storage in DRAM"*, The Memory Forum

[10] Moinuddin K. Qureshi, Dae-Hyun Kim, Samira Khan, Prashant J. Nair and Onur Mutlu, *"AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for*

DRAM Systems", 2015, 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks

[11] R.K. Venkatesan, S. Herr and E. Rotenberg, *"Retention-Aware Placement in DRAM (RAPID): Software Methods for Quasi-Non-Volatile DRAM"*, The Twelfth International Symposium on High-Performance Computer Architecture, 2006

[12] Song Liu, Karthik Pattabiraman, Thomas Moscibroda and Benjamin G. Zorn, *"Flikker: Saving DRAM Refresh-power through Critical Data Partitioning"*, ASPLOS XVI Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems

[13] Arnab Raha, Soubhagya Sutar, Hrishikesh Jayakumar and Vijay Raghunathan, *"Quality-Aware Data Allocation in Approximate DRAM"*, 2015, International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)

[14] *"SDRAM Memory Controller"*, https://github.com/stffrdhrn/sdram-controller

[15] *"Technical Note: Calculating Memory System Power for DDR3"*, Micron®

[16] *"SDRAM Power Calculator - Micron Technology, Inc."*, https://www.micron.com/~/media/documents/products/power-calculator/sdram_power_calc_10.xls

[17] *"Fitter Resource Usage Summary Report"*, https://www.intel.com/content/www/us/en/programmable/quartushelp/17.0/mapIdTopics/mwh1465496451103.htm

[18] *"Clock gating integrated cell"*, http://vlsi-soc.blogspot.com/2012/08/clock-gating-integrated-cell.html

[19] *"JEDEC Global Standards for the Microelectronics Industry"*, https://www.jedec.org/standards-documents