User Manual DRAM Core Model

Thomas Vogelsang
tvogelsang@rambus.com

1. Copyright and Disclaimer

For additional information, please contact:
Rambus Inc.
1050 Enterprise Way, Suite 700
Sunnyvale, CA 94089
408-462-8000

2. Overview

The DRAM Core Model tool has been written to allow the assessment of a wide variety of DRAM architectures with respect to power consumption. To give maximum flexibility the program was written in Perl working from a description language specified to allow full description of DRAM architectures.
For a discussion of the model background and a selection of results achieved with the model see [1].

3. Program Usage

3.1. Contents of the Program Package

The program package contains in addition to this documentation and the reference [1] the program code (Perl code for UNIX), a sample input file and output files created using the sample input file.

3.2. Running the Program

The program is started with the UNIX line command `<path>dram_model.pl <-v> <-d<zero|one>> <name>`. The flag `-v` creates verbose output. The flags `-dzero` respectively `-done` force the loop dimension (see the programming of loops below). The input file `<name>.dram` is required to describe the DRAM to be evaluated. The description language is given below.
The program consists of the main program and a number of packages. The packages need to be stored in a subdirectory `perlsubs` under the directory where the main program is located. This directory needs to be defined in the main program's 'use lib' statement. Check if it corresponds to the correct location if the program aborts not finding its subroutines.

3.3. Important Usage Considerations

The DRAM core model can be only as accurate as its input files. The description of a typical 55nm DDR3 commodity DRAM is provided with the model. Extrapolating the model to technologies other than 55nm should take into account the technology roadmap discussion in [1]. As outlined there the ITRS roadmap is an important input. Any major modification of the input file should be followed by a sensitivity analysis. In such a sensitivity analysis parameters are varied by a certain amount, e.g. ±20% and then a Pareto is generated showing which parameters have the largest influence on the results of the model. The user must make sure that the values of the parameters having the largest influence are well understood. Miscellaneous logic circuitry is most difficult here. It can't be expected to be modeled absolutely correct without having access to an actual DRAM product design. This is also the area where there are the largest variations between DRAMs of the same type and generation but different vendors. The description of miscellaneous logic circuitry in the input file can be used as fitting parameter in a comparison with existing DRAMs. When evaluating completely novel architectures the most important goal is to come up with some reasonable assumptions (e.g. a certain number of gates per bit of internal data width) and then use these assumptions consistently over all the architectures being compared. In that way even if absolute accuracy is less good, the relative accuracy of the comparison will still be high.

3.4. *Description Language*

3.4.1. General

The DRAM is described in the form

```
[<Section>
[<Key> <Subkey>=<Value>]]
```

`<Section>`, `<Key>` and `<Subkey>` are defined words. `<Value>` can either also be a defined word or a number depending on the situation it is used in.
All lines or partial lines after # are ignored. Empty lines are ignored. Spaces are treated as word separators.
A backslash '\' signifies that the next line is a continuation of the current line. This can be used for very long values, especially lists in the '`GlobalLoop`' section. The backslash has to be before a comment, not after a comment at the line end, to be recognized.

3.4.2. Acronyms

SA	Primary or bitline sense-amplifier
SSA	Secondary or array data line sense-amplifier
BL	bitline
WL	wordline
MWL	Master wordline
SWL	Sub or local wordline
HV	High voltage
nch	N-channel MOSFET
pch	P-channel MOSFET

3.4.3. Language Definition

All words defined here are evaluated without regarding upper or lower case letters. Upper and lower case can be used to make words more easily readable.

Section	Key	Subkey	Unit	Comment
`Technology`	`Oxide_Main`	`t`	Å	Oxide thickness of standard periphery and low voltage transistors
		`C`	fF/µm^2	Oxide capacitance of standard periphery and low voltage transistors
	`Oxide_HV`	`t`	Å	Oxide thickness of high voltage transistors
		`C`	fF/µm^2	Oxide capacitance of high voltage transistors
	`Oxide_Array`	`t`	Å	Oxide thickness of cell access transistor
		`C`	fF/µm^2	Oxide capacitance of cell access transistor
	`Transistor_Main`	`Lmin`	µm	Minimum channel length of standard periphery and low voltage transistor
		`Csd`	fF/µm	Source-drain capacitance of standard periphery and low voltage transistor
	`Transistor_HV`	`Lmin`	µm	Minimum channel length of high voltage transistor
		`Csd`	fF/µm	Source-drain capacitance of high voltage transistor
	`Transistor_Array`	`L`	µm	Channel length of array access transistor
		`W`	µm	Channel width of array access transistor
	`Array`	`Cbl`	fF	Bitline capacitance including SA wiring but without devices, requires setting of SA device sizes
		`Cshare_BL_SWL`	%	Share of capacitance BL to SWL
		`Cs`	fF	Cell capacitance
		`MWLwire`	fF/um	MWL specific wire capacitance
		`MWLpredecode`		Number of pre-decoded signals in MWL decoder. Number of MWL nch is logarithm base 2 of that number
		`MWLdecNchW`	µm	Device width MWL nch
		`MWLdecNchL`	µm	Device length MWL nch
		`WLctrlLoadNchW`	µm	Device width nch load of wlctrl in SA hole
		`WLctrlLoadPchW`	µm	Device width pch load of wlctrl in SA hole

Section	Key	Subkey	Unit	Comment
		SWLdrvNchW	µm	Device width nch at master wordline
		SWLdrvPchW	µm	Device width pch at master wordline
		SWLrstNchW	µm	Device width SWL restore nch
		SAnchW	µm	Device width SA nch
		SAnchL	µm	Device length SA nch
		SApchW	µm	Device width SA pch
		SApchL	µm	Device length SA pch
		SAeqlW	µm	Device width SA equalize
		SAeqlL	µm	Device length SA equalize
		SAshrW	µm	Device width SA BL share (only folded BL)
		SAshrL	µm	Device length SA BL share (only folded BL)
		SAbswW	µm	Device width SA bit switch
		SAbswL	µm	Device length SA bit switch
	BEOL	CperiWire	pF/mm	Wire capacitance of periphery signal wiring
BasicElectrical	Voltages	Vcc	V	External supply voltage
		Vperi	V	Standard periphery voltage
		Varray	V	Array voltage
		Vpp	V	Wordline voltage
	Efficiency	Vperi	%	Current multiplier external supply to internal current standard periphery voltage
		Varray	%	Current multiplier external supply to internal current array voltage
		Vpp	%	Current multiplier external supply to internal current wordline voltage
FloorplanPhysical	Vertical	Blocks	[[A|P]<n>]]	Description of block arrangement in horizontal direction. A is array block, P is periphery block. At the moment only one array block is supported.
	Horizontal	Blocks	[[A|P]<n>]]	Description of block arrangement in horizontal direction. A is array block, P is periphery block. Only one type of array block, A1, is supported.
	SizeVertical	[A|P]<n>	µm	Vertical size of block [A|P]<n>
	SizeHorizontal	[A|P]<n>	µm	Vertical size of block [A|P]<n>
	CellArray	BL	[h|v]	Direction of bitline
		BitsPerBL		Bits per bitline

Section	Key	Subkey	Unit	Comment
		`CellsPerSWL`		Bits per sub (local) wordline
		`BLtype`	[open|folded]	folded bitlines (used with $8f^2$ cells) have twice as many local wordlines as bits per BL, open bitlines (used with $6f^2$ and $4f^2$ cells) have cross point cells but need a dummy array at the edge
		`CslBlocks`		Number of array blocks across which one physical CSL extends
		`ECCFactor`		This factor multiplies the multiplies the internal page size to show the impact of ECC on the array and sense-amplifier. Additional power due to circuitry overhead needs to be implemented as a logic block. ECCFactor does not need to be specified and defaults to 1.
		`BLpitch`	nm	Pitch of bitlines; Optional input,, used to calculate array block size bottom up and compare with size given in SizeVertical and SizeHorizontal
		`WLpitch`	nm	Pitch of wordlines; Optional input,, used to calculate array block size bottom up and compare with size given in SizeVertical and SizeHorizontal
		`SenseampWidth`	um	Width of sense-amplifier; Optional input,, used to calculate array block size bottom up and compare with size given in SizeVertical and SizeHorizontal
		`SWLdriverWidth`	um	Width of SWL driver stripe; Optional input,, used to calculate array block size bottom up and compare with size given in SizeVertical and SizeHorizontal
		`RowRedundancy`	%	Amount of row redundancy; Optional input,, used to calculate array block size bottom up and compare with size given in SizeVertical and SizeHorizontal
		`ColumnRedundancy`	%	Amount of column redundancy; Optional input,, used to calculate array block size bottom up and compare with size given in SizeVertical and SizeHorizontal
		`ArchitectureFactor`		This factor does not change the calculation results. It can be used to snap the physical size to a desired as-pect ratio in case this cannot be done by changing the starting values of the vertical and horizontal array block sizes.

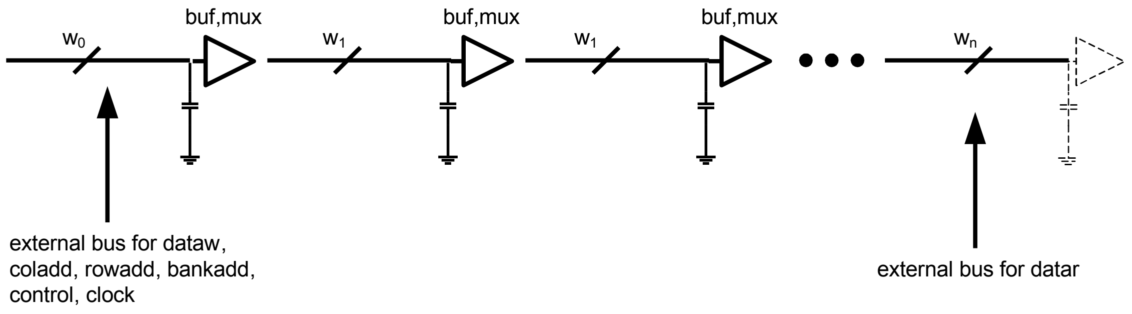
Section	Key	Subkey	Unit	Comment
FloorplanSignaling	<DataR| DataW| ColAdd| RowAdd| BankAdd| Control| Clock><n>	Direction	[h|v]	The key defines the type of the signal segment: data used for read, write, column, row or bank address, control and clock. Each signal segment is numbered. The numbering of each path needs to be consecutive.
		Start	<m>_<n>	<m>_<n>are the coordinates of the block from the physical description where the signal segment starts (based on block numbering beginning with 0). Start / End combinations go from one block to another (center to center).
		End	<m>_<n>	<m>_<n>are the coordinates of the block from the physical description where the signal segment starts (based on block numbering beginning with 0). Start / End combinations go from one block to another (center to center).
		Inside	<m>_<n>	<m>_<n>are the coordinates of the block from the physical description where the signal segment is located if it does not extend to another block. Use in combination with Fraction
		Fraction	%	Fraction is the percentage of the extension of the block with a signal segment inside which is used as segment length. Use in combination with Inside.
		NchW	µm	Device width of nch load at start
		PchW	µm	Device width of pch load at start
		Load	fF	Device load at start
		Mux	<m>:<n>	Serialize or deserialize ratio at start
		Swing	<Full|Half|<x>V>	Signal swing on segment (full, mid level or specified voltage), defaults to full at Vperi.
		P_Eff	%	Power efficiency: the external power at Vcc is calculated as Vcc*Current/P_Eff. Default is 100% if not defined.
		Toggle	%	Toggle rate as fraction of maximum data rate
	CoreBoundary	Read		Last segment of read data path considered to be in

Section	Key	Subkey	Unit	Comment
				core for current output
		Write		First segment of write data path considered to be in core for current output
Periphery	Logic<n>	Toggle	%	Toggle rate as fraction of base frequency (see component)
		Gates		Number of logic gates
		NchW	um	Typical nch width in gate
		PchW	um	Typical pch width of gates
		DevicesPerGate		Number of transistors per gate
		AreaFactor	%	Percentage of transistor gate area of total block area used to calculate block size
		Operation	<all|activate|read|write|precharge|readwrite|row>	Operation in which current is generated
		Component	<row|column|data|control|clock>	Part of DRAM circuitry in which current is generated. Row and column will be counted under 'core' in the power usage summary, the rest under 'periphery'. row, column and control base frequency is the control frequency, clock and data base frequency is the data clock frequency.
		Swing	V	To be set if voltage is different from Vperi
		P_Eff	%	Power efficiency: the external power at Vcc is calculated as Vcc*Current/P_Eff. Default is 100% if not defined.
	Sink<n>	Vcc	mA	Constant current sink on Vcc
		Vperi	mA	Constant current sink on Vperi
		Varray	mA	Constant current sink on Varray
		Vpp	mA	Constant current sink on Vpp
Specification	IO	Width		Number of I/O bits
		DataRate	Gb/s/pin	Data rate
		Type	<single|differential>	Type of I/O (single ended or differential) – not yet implemented as difference in results
		Burstlength		Allows specification of a burst length different from the default which is a burst of the full data granularity
	Control	BankAdd		Number of bank address bits

Section	Key	Subkey	Unit	Comment
		RowAdd		Number of row address bits
		ColAdd		Number of column address bits
		Miscellaneous		Number of additional miscellaneous bits
		Banks		Number of banks (alternative to bankadd)
		Rows		Number of rows (alternative to rowadd)
		Columns		Number of columns (alternative to coladd)
		Frequency	MHz	Frequency of control bus – this is the frequency used for the pattern and control signals
	Dataclock	Number		Number of data clocks
		Frequency	MHz	Data clock frequency
	Pattern	Loop	string of act, pre, rd, wrt, nop	Calculates percentage spent in each operation. Row-cycle is calculated from pattern. IMPORTANT: the clock used for the pattern is the control clock
GlobalLoop	Loop<n>	Section		Section of variable which will be used in loop
		Key		Key of variable which will be used in loop
		Subkey		Subkey of variable which will be used in loop
		Value	linear <start> <end> <step> or logarithmic <start> <end> <multip-lier> or list <list values separated by comma>	Value of variable which will be used in loop
	Type	Dimension	<one|multi|zero>	Type of loop: one-dimensional requires that the value-entries of all Loop<n> statements are lists and that all lists have the same number of entries. Each calcula-tion then varies the combination of all Loop<n> statements at the same list index. Multi-dimensional loops vary all possible combinations of Loop<n> changes. Default if omitted is multi. Zero bypasses the loop by evaluating only the first setting. This is useful for debug and when creating the floorplan drawing.

3.5. Wiring Topology Definition

The description of the wiring follows the topology shown below. Each bus is split into segments which have a buffer / multiplexer at the end which can provide a load and can also change the width and frequency of the bus. If the parameter mux is e.g. 2:1, between segment 0 and segment 1 then the width $w_0 = 2\ w_1$ and the frequency $f_0 = ½\ f_1$. Adding a buffer / multiplexer to the last segment has no meaning as there is no wiring after it. The width and frequency of each segment is calculated from the external pin information given in the specification section. It is therefore calculated backward for the read data path and forward for all other signal paths.


buf,mux
buf,mux
buf,mux
w_0
w_1
w_1
w_n
external bus for dataw, coladd, rowadd, bankadd, control, clock
external bus for datar

3.6. Presentation of Results

3.6.1. Physical and Signaling Floorplan

Follow these steps to create a PowerPoint Slide showing the architecture and signal path layout

- Open <name>_physical.txt in Excel.
- Select the range of x and y coordinates.
- Insert a chart, type xy-scatter sub-type lines with no data points; deselect all options like axes, grid lines, legend etc.
- Insert on new sheet.
- Format plot area and set both border and area to none.
- Select data series and set color to black.
- Save in Excel workbook format.
- Open <name>_signals.txt and move worksheet to workbook with physical drawing.
- Create a line below the data provided by Perl, enter 0 for each column and add this line to each value of the Perl values. The purpose is to provide an offset which can be used to separate visually different signals which would otherwise overlay and hide each other.
- Copy all signals with applied offset to the drawing using 'Paste Special' and selecting 'New Series, x-axis in first column'.
- Change colors and line styles as desired, use offsets to move apart.
- Copy the drawing and paste special into PowerPoint as enhanced metafile.
- Ungroup by first confirming to ungroup it even when it is a picture and the ungroup again.

- Delete the background so that only the lines are left and regroup.
- Resize to represent the correct aspect ratio in two steps: first format the size of the object by setting height and width to the y and x dimension of the die without locking the aspect ratio, then lock the aspect ratio and resize again to have the desired size to fit the presentation.

3.6.2. Power Results

All power results are written to tab-separated files readable with Excel. If there is no GlobalLoop section in the input file, then there will be only one file created named <name>_power.txt. This file contains separate entries for all contributors to power listed separately to create an overall power usage Pareto as well as grouped results which can be used to create pie-charts. When there is a GlobalLoop section each data point will have a separate consecutively numbered file of this kind. The not numbered power file is then a summary file which has one entry line for each data point to allow the plotting of sensitivity charts.

4. Program Details

4.1. Program Structure

The script is built as a series of modules to keep the file sizes manageable and to allow flexibility in development. The main modules are

- BasicElectrical: subroutines for section
- Current: current calculator
- FloorplanPhysical: subroutines for section
- FloorplanSignaling: subroutines for section
- GlobalLoop: subroutines for section
- Parser: reads input file and converts into internal hash storage
- Periphery: subroutines for section
- Power: power calculator and output routines
- Specification: subroutines for section
- Syntax: syntax checker using subroutines from each section
- Technology: subroutines for section

Further checking of parameters is done during evaluation when the context of a parameter is known. The program is aborted with an error message in case of errors in the input file.

4.2. Internal data structures

The parsed input data are in all subroutines available in a hash %input which has keys of the form <section>-<key>-<subkey>. The value of each hash entry is the value of the specific parameter. This hash is set by parsing the input file and then used for all further operations.

4.2.1. FloorplanPhysical

The physical Floorplan is described with five variables:

- $bl_dir holds the direction of the bitline (horizontal or vertical)

- @blk_coord_h is an array storing all x-coordinates
- @blk_coord_v is an array storing all y-coordinates
- @blk_type_h is an array storing either a or p for the block type (array or periphery) going horizontally across the die
- @blk_type_v is an array storing either a or p for the block type (array or periphery) going vertically across the die

Blocks are defined by the grid built from the horizontal and vertical coordinates. A block which has array both as horizontal and vertical designator is a true array block, a block having only one array designator is a periphery block whose size is defined in one direction by an array block, i.e. an on-pitch block.
See section 4.5.3 for more details on the calculation and input of array block size.

4.2.2. FloorplanSignaling

The signaling allows describing a read data path, writing data path, read-write data path, column, row and bank address bus, a control bus and the clock. There are therefore 8 variables (@data_r, @data_w, @coladd, @rowadd, @bankadd, @control, @clock). All of them are arrays of hashes. Each array record describes one segment of the path. The segments must be continuous. There are two types of segments. The first type goes from one block to a different one. It is always assumed to go from block center to block center. The second type is within one block and extends either in horizontal or vertical direction across a part of the block. For the purpose of connection it is also assumed to have the connection point in its center. Each segment is described by a hash with the keys 'length' (always provided as calculated from the geometry description) and 'bufstartnchw', 'bufstartpchw', 'bufendnchw', 'bufendpchw', 'bufstartload', 'bufendload', 'bufstartmux', 'bufendmux', 'swing' and 'toggle'. The latter are provided when they are included in the description file.

4.2.3. Technology and Specification

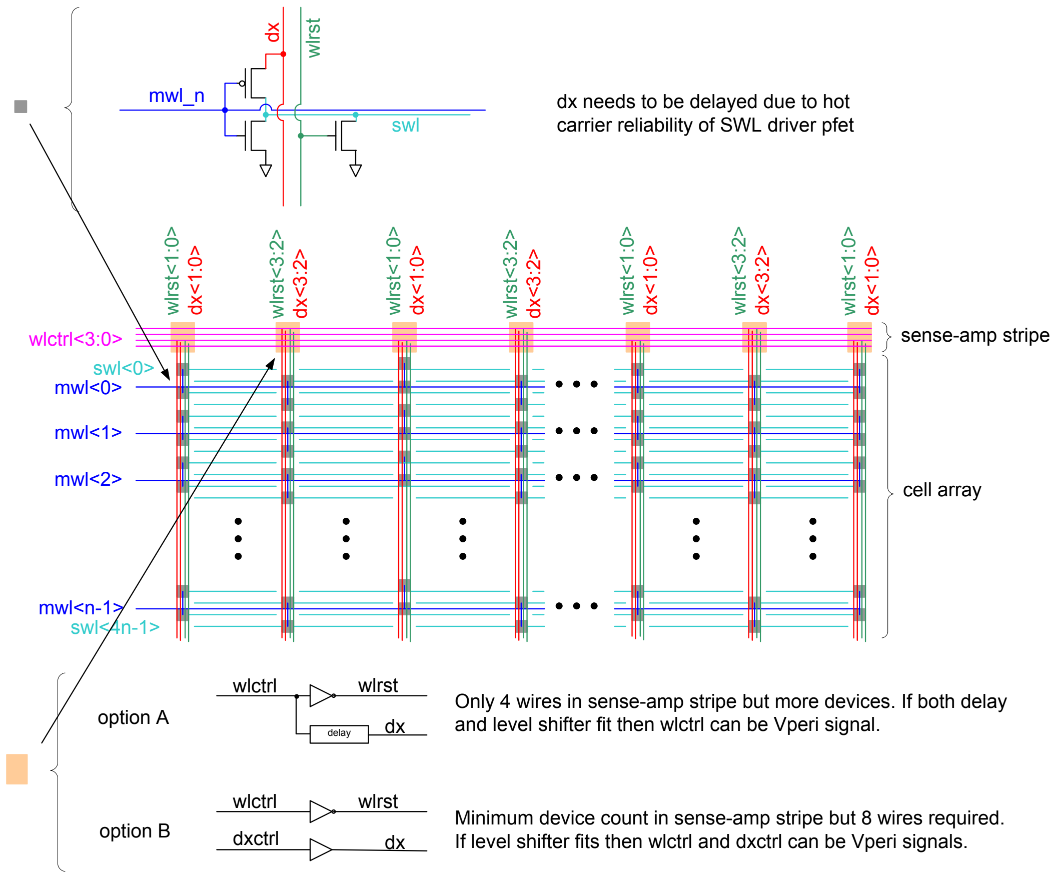
Information from the sections on technology and specification is combined to populate a number of variables which describe the DRAM in more detail and are used to calculate power consumption. These variables are $prefetch (ratio between bits fetched from core to width of the IO), $pagesize (number of bits which can be fetched from an open row), $wl_act_overhead (multiplier for open BL architectures to account for additional edge arrays), $numberSWLstripes (number of stripes which drive SWLs), $n_subarray_par_bl (number of array sub-blocks parallel to the BL direction in an array block), $n_subarray_par_wl (number of sub-blocks parallel to the WL direction in one array block), $n_subbanks (number of physical subunits in one bank), $opshare_act (share of activate operations of total pattern length), $opshare_pre (share of precharge operations of total pattern length), $opshare_rd (share of read operations of total pattern length), $opshare_wrt (share of write operations of total pattern length), $row_period (repeating time of activate commands) and $core_frequency (frequency of operation of the DRAM core).
Use the program with the –v flag and pipe to more to read the output showing intermediate calculations when defining a new architecture to make sure that no errors occurred in the specification.

4.3. Calculation of Results

The calculation of results is done in the two packages ‘Current’ and ‘Power’. All powers are calculated using the steps capacitance – charge – current and power.

4.3.1. Row current

The row current lumps together all currents contributing to wordline operation. It has two main operational parts: activate and precharge. The wordline topology used is shown below. This segmented wordline is standard in today’s DRAMs. There are variations at some vendors; they do however not change the current significantly.

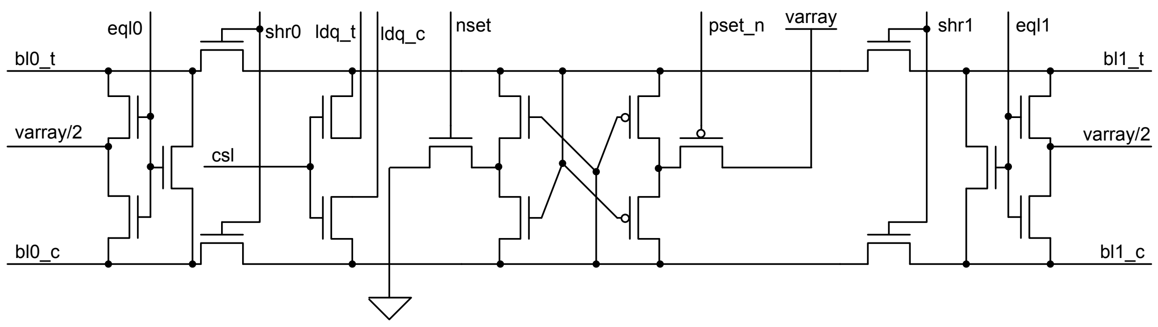

dx
wlrst
mwl_n
swl
dx needs to be delayed due to hot carrier reliability of SWL driver pfet
wlrst<1:0>
dx<1:0>
wlrst<3:2>
dx<3:2>
wlctrl<3:0>
sense-amp stripe
swl<0>
mwl<0>
mwl<1>
mwl<2>
cell array
mwl<n-1>
swl<4n-1>
option A
wlctrl
wlrst
delay
dx
Only 4 wires in sense-amp stripe but more devices. If both delay and level shifter fit then wlctrl can be Vperi signal.
option B
wlctrl
wlrst
dxctrl
dx
Minimum device count in sense-amp stripe but 8 wires required. If level shifter fits then wlctrl and dxctrl can be Vperi signals.

According to the chart, modeled capacitive loads are the array access transistor gate oxide, the parasitic wordline-to-bitline capacitance (mostly due to the bitline contact), master wordline, wlrst, dx, wlctrl and sub-wordline wire capacitances and the device load of the different drivers of these signals. In addition the row address bus in the periphery is modeled as described in the signaling floorplan.
The row current is calculated using the row cycle time. At the final current power evaluation when data pattern are used, the row cycle time currents are modified to correspond to the clock frequency and the number of activate / precharge commands in the pattern. Current caused by address change is only counted during activate. All row power is counted as core power when adding up power contributors by location.

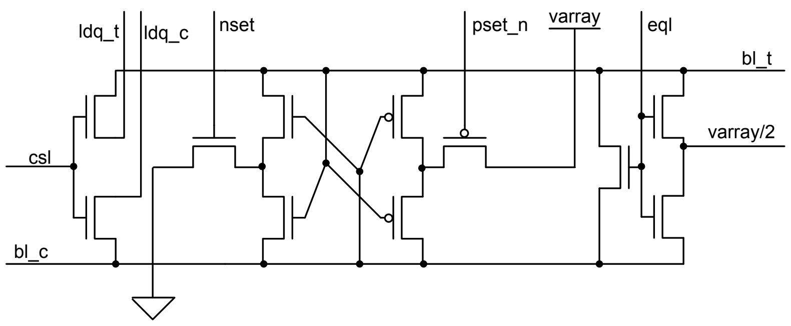
4.3.2. Sense-amplifier current

Sense-amplifier current is calculated as device current based on all sense-amplifier device loads and the wire load of the control signals. The device load is different for folded bitline architectures used in $8f^2$ and above cell technologies and for open bitline architectures used in $6f^2$ and below cell technologies. The figure below shows the sense-amplifier structure for both cases.

Folded bitline sense-amplifier



Open bitline sense-amplifier



Sense-amplifier current is mostly drawn during activate when the sense-amplifier is set from precharge at mid-level to the bitline pair at complement levels and at write when the bitline pair is flipped into the opposite state. In the program the full bitline capacitance current is calculated as part of the sense-amplifier current. Sense-amplifier current is counted as part of the core current when adding up power contributors by location.

4.3.3. Signal current

Signal current is calculated from the device load of the buffers and the wire load of the signal segments using the periphery wire capacitance value. Data current is assigned to the core and periphery respectively as defined in the input file, row and column address current is assigned to the core, all other signal current is assigned to the periphery when adding up power contributors by location.

4.3.4. Logic blocks

Current due to constant current sinks (e.g. generator bias currents) are added to the periphery current. Current of logic blocks is added to the operation for which it has been defined. The device load part is calculated using the defined number of logic gates, devices per gate and device width and length for each logic block. A wire length of each block is calculated according to the following equations:

$$\text{area} = 3 * \text{gate area} / \text{area factor}$$
$$\text{block length} = \text{sqrt}\ (\text{area})$$
$$\text{wire length} = (\text{block length} / \text{transistor length}) * \text{wire factor}$$

In this way assumptions about wiring density and device coverage can be parameterized for an estimate of logic block load.

4.4. Sample Input File

This is the input file `ddr3_55nm_6f2_2g.dram`.

```
# DRAM description for architecture, power and performance model

# Device: Reference DDR3 2G 55nm

# (c) 2010 Rambus Inc.
# Author: Thomas Vogelsang

# Section: Description of physical layout
# -----------------------------------------

FloorplanPhysical

# Arrangement of blocks (starting in lower left corner)

# Array information
CellArray BL=v BitsPerBL=512 CellsPerSWL=341 BLtype=open CslBlocks=1
CellArray WLpitch=165nm BLpitch=110nm SenseampWidth=19um SWLdriverWidth=9um

Horizontal blocks = A1 P1 P1 A1 A1 P1 P1 A1
# A1: array block
# P1: row drivers

Vertical blocks = A1 P1 P2 P1 A1
# A1: array block
# P1: column driver and core data path
# P2: main periphery logic and pads

# Block sizes
SizeHorizontal A1=2244um P1=120um
SizeVertical A1=3396um P1=200um P2=530um

# Section: Description of signal wiring
# ---------------------------------------

FloorplanSignaling

# Core boundary in data path
CoreBoundary Read=3 Write=4 # last respectively first segment considered to be in core

# Write data path
DataW0 inside=0_2 fraction=5% direction=v mux=1:4 PchW=19.2 NchW=9.6 Toggle=50%
DataW1 inside=0_2 fraction=25% direction=h Toggle=50%
DataW2 start=0_2 end=6_2 mux=1:2 PchW=19.2 NchW=9.6 Toggle=50%
DataW3 start=6_2 end=6_3 Toggle=50%
DataW4 start=6_3 end=7_3 PchW=19.2 NchW=9.6 Toggle=50%
DataW5 start=7_3 end=7_4
DataW6 inside=7_4 fraction=50% direction=v
```

```
# Read data path
DataR0 inside=7_4 fraction=50% direction=v swing=1.0V
DataR1 start=7_4 end=7_3 swing=1.0V PchW=19.2 NchW=9.6
DataR2 start=7_3 end=6_3 Toggle=50%
DataR3 start=6_3 end=6_2 PchW=19.2 NchW=9.6 mux=2:1 Toggle=50%
DataR4 start=6_2 end=0_2 PchW=19.2 NchW=9.6 Toggle=50%
DataR5 inside=0_2 fraction=25% direction=h mux=4:1 Toggle=50%
DataR6 inside=0_2 fraction=5% direction=v Toggle=50%

# Row address path
Rowadd0 start=4_2 end=1_2 PchW=19.2 NchW=9.6
Rowadd1 start=1_2 end=1_4 PchW=19.2 NchW=9.6
Rowadd2 inside=1_4 fraction=50% direction=v PchW=19.2 NchW=9.6

# Column address path
Coladd0 start=4_2 end=2_2 PchW=19.2 NchW=9.6
Coladd1 start=2_2 end=2_3 PchW=19.2 NchW=9.6
Coladd2 start=2_3 end=3_3
Coladd2 inside=3_3 fraction=50% direction=h

# Bank address path
Bankadd0 start=4_2 end=1_2 PchW=19.2 NchW=9.6
Bankadd1 start=1_2 end=1_3

# Control path
Control0 inside=7_2 fraction=50% direction=h
Control1 start=7_2 end=0_2 PchW=19.2 NchW=9.6
Control2 inside=0_2 fraction=50% direction=h

# Clock path
Clock0 inside=7_2 fraction=50% direction=h
Clock1 start=7_2 end=0_2 PchW=19.2 NchW=9.6
Clock2 inside=0_2 fraction=50% direction=h

# Section: Description of specification (data sheet)
# --------------------------------------------------

Specification

IO width=16 datarate=1.6Gbps
Dataclock number=1 frequency=800MHz
Control frequency=800MHz bankadd=3 rowadd=14 coladd=10 miscellaneous=6

# Section: Description of technology related parameters
# -----------------------------------------------------

Technology

Oxide_Main t=26A
Oxide_HV t=49A
Oxide_Array t=49A
Transistor_Main Lmin=0.08um csd=0.8fF/um
Transistor_HV Lmin=0.27um csd=0.8fF/um
```

```
Transistor_Array L=0.085um W=0.055um
Array Cbl=85fF Cs=24fF Cshare_BL_SWL=30% BitsPerCsl=4 # Cell and bitline
Array MWLwireCap=0.3pF/mm MWLpredecode=8 MWLdecNchW=0.65um MWLdecPchW=1.85um MWLdecToggle=50% # MWL
Array WLctrlLoadNchW=3.2um WLctrlLoadPchW=6.4um SWLdrvNchW=1.9um SWLdrvPchW=5.4um SWLrstNchW=1.4um SWLwireCap=0.3pF/mm # SWL
Array SAnchW=1.9um SApchW=1.33um SAnchL=0.16um SApchL=0.16um SAeqlW=0.9um SAeqlL=0.16um SAbswW=1.9um SAbswL=0.08um # sense-amp
Array SAnsetW=0.22um SAnsetL=0.27um SApsetW=0.22um SApsetL=0.27um # SA set devices
BEOL CperiWire=0.25pF/mm
Leakage SD_Main=3nA/um Gate_Main=0.1nA/um2 GIDL_HV_nch=0.6nA/um GIDL_HV_pch=0.005nA/um
Leakage column_width=8.5um MWLdrvNchw=8.5um BLjunction=27fA/cell

# Section: Description of basic electrical parameters
# ----------------------------------------------------

BasicElectrical

Voltages Vcc=1.5V Vperi=1.3V Varray=1.2V Vpp=2.9V
Efficiency Vperi=100% Varray=100% Vpp=37%

# Section: Peripheral logic
# -------------------------

Periphery

Logic0 Gates=200 NchW=3.2um PchW=6.4um DevicesPerGate=4 AreaFactor=10% WireFactor=33% Operation=row Component=control
Logic1 Gates=200 NchW=3.2um PchW=6.4um DevicesPerGate=4 AreaFactor=10% WireFactor=33% Operation=readwrite Component=control
Logic2 Gates=500 NchW=3.2um PchW=6.4um DevicesPerGate=4 AreaFactor=10% WireFactor=33% Operation=readwrite Component=column
Logic3 Gates=600 NchW=3.2um PchW=6.4um DevicesPerGate=3 AreaFactor=10% WireFactor=33% Operation=readwrite Component=data
Logic4 Gates=200 NchW=3.2um PchW=6.4um DevicesPerGate=3 AreaFactor=10% WireFactor=33% Operation=readwrite Component=control
Logic5 Gates=200 NchW=3.2um PchW=6.4um DevicesPerGate=3 AreaFactor=10% WireFactor=33% Operation=all Component=control
Sink0 Vcc=30mA

# Section: Description of global loop variables
# ---------------------------------------------

GlobalLoop
Type Dimension=one

Loop0 Section=Specification Key=Pattern Subkey=loop Value=list \
  act pre nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop \
                                                                                                             nop nop nop nop nop, \ # Idd0
   rd nop nop nop, \                                                                                                                # Idd4R
  wrt nop nop nop, \                                                                                                                # Idd4W
   rd act pre nop  rd act pre nop  rd act pre nop  rd act pre nop  rd nop nop nop  rd nop nop nop  rd nop nop nop  rd nop nop nop, \# Idd7R
  wrt act pre nop  rd act pre nop wrt act pre nop  rd act pre nop wrt nop nop nop  rd nop nop nop wrt nop nop nop  rd nop nop nop   # Idd7RW
```

4.5. *Sample Output*

4.5.1. Zero-Dimension Run

Run with `dram_model.pl -dzero ddr3_55nm_6f2_2g`
Comments are inserted in green.

```
Physical floorplan:
die width = 9456um, die height = 7722um
```

Die size calculated from block size in input file, not from bitline and wordline pitch. See section 4.5.3 for the iterative procedure to find the correct array block size.

```
Device Architecture Overview:
Density 2048Mb
8 Banks, 16384 rows, 1024 columns, 16 IOs
Aggregate bandwidth 3.200GB/s
Prefetch 8 bits
Granularity 16 Bytes
Subbanks per bank: 1
Subarrays in array block parallel BL: 32
Subarrays in array block parallel WL: 48
```

The program calculates a number of generic figures characterizing the DRAM. These figures can be used to check the correctness of the input file. Wrong numbers of columns or rows or an incorrect physical description will show up here.

```
Page size from specification: 2.0kB
Sanity check: minimum page size estimated from architecture: 2.0kB
```

A further sanity check: the page size from the specification should be the same or larger than the minimum page size from architecture.

```
Number of SWL driver stripes: 49
Row activation overhead (>1 for open bitline): 1.031
```

Open bitline architectures have a row activation overhead since only half of the cells in the two edge segments can be used, requiring simultaneous activation of two wordlines if a row address falls into an edge segment. The power program assumes random row address usage to calculate an average overhead.

```
Full burst in control clock cycles: 4
```

Calculated from specification

```
Row period: 45.0ns
```

Calculated from data pattern and specification. A pattern without activate and precharge will show here a default value of 60ns which has no influence on the power. The model uses an effective row period that might be achieved in a real DRAM only by bank interleaving, e.g. a row period of 60ns in a single bank could become an effective row period of 15ns for the DRAM by interleaving 4 banks.

```
Calculated array block size:
Horizontal: calculated 2244um, defined 2244um (difference -0.0%)
Vertical:   calculated 3396um, defined 3396um (difference -0.0%)
```

This compares the calculated array block size and the size defined in the input file. An iterative process should be used to achieve a 0% difference.

```
Pattern current and power:
Burst length is 4 clock cycles, used as multiplier for internal share of column and data operation in read
and write.
```

The burst length determines how many nop cycles in the pattern following a read or write command are really read or write commands, not nop.

```
Activate current:     32.1mA (  3% of  1156.9mA )
Precharge current:     6.2mA (  3% of   223.1mA )
Write current:         0.0mA (  0% of     0.0mA )
Read current:          0.0mA (  0% of     0.0mA )
NOP current:          60.3mA ( 94% of    63.8mA )
```

Pattern power attributed to the different operations. The rightmost number is a fictive calculation value assuming one operation of a certain type per clock cycle. The example DDR3 part is running at 800MHz, a clock cycle is therefore 1.25ns. A row cycle time of 45ns means one row cycle every 36 clock cycles or a utilization of about 3%. For read and write every fourth clock cycle which leads to a full burst, the breakdown will say 25%, but the current will consider the burst length which has active data lines over all 4 clock cycles.

```
Total current:        98.6mA
Power:               147.9mW
```

Current as sum over components and power a s current multiplied with supply voltage

```
Power by location and voltage
core-varray        34.0
core-vperi          7.1
core-vpp            9.6
periphery-vcc      45.0
periphery-vperi    52.2
```

This output breaks down the contributors in large buckets. A more detailed breakdown is available in the file `<name>_power.txt`

```
Power by location
core               50.7
periphery          97.2
```

This output breaks down the contributors in large buckets. A more detailed breakdown is available in the file `<name>_power.txt`

```
Successful run of 'dram_model.pl'.
```

4.5.2. One-Dimension Run

Run with `dram_model.pl -done ddr3_55nm_6f2_2g`
Comments are inserted in green.

```
Evaluating specification-pattern-loop = act pre nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop
nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop
```

Parameters used in first instantiation of loop

```
Physical floorplan:
die width = 9456um, die height = 7722um
```

Control output

```
Power:   147.9mW (row period 45.00ns)
```

Power of first instantiation of loop

```
Evaluating specification-pattern-loop = rd nop nop nop
Power:   351.6mW (row period 60.00ns)
```

Parameters and power of second instantiation of loop

```
Evaluating specification-pattern-loop = wrt nop nop nop
Power:   364.1mW (row period 60.00ns)
```

Parameters and power of third instantiation of loop. The sample input file shows a reduced swing on the master array data lines during read as it is typical for DRAMs. The miscellaneous logic does not use different settings for read and write, so the read power is lower than the write power. In a real DRAM read and write circuitry are partially different, so the difference in data line power might be overcompensated leading to a higher read power than write power. Data sheets of different vendors show there are parts on the market which have higher read power, higher write power or the same power.

```
Evaluating specification-pattern-loop = rd act pre nop rd act pre nop rd act pre nop rd act pre nop rd nop
nop nop rd nop nop nop rd nop nop nop rd nop nop nop
Power:   586.4mW (row period 10.00ns)
```

Parameters and power of fourth instantiation of loop

```
Evaluating specification-pattern-loop = wrt act pre nop rd act pre nop wrt act pre nop rd act pre nop wrt
nop nop nop rd nop nop nop wrt nop nop nop rd nop nop nop
Power:   592.7mW (row period 10.00ns)
```

Parameters and power of fifth instantiation of loop

```
Successful run of 'dram_model.pl'.
```

4.5.3. Array Block Size Determination

Calculating the array block size has not been fully automated due to the ambiguity which is possible if different aspect ratios are allowed. Page size is not a sure differentiator as a logical page can either be the same as a physical page or it can be broken up into two or more physical pieces. An iterative process, shown below in an example, can be used to put the exact array block size in the input file.

Guess 1

For the first guess any reasonable number of a few thousand by a few thousand µm can be used. The program aborts whenever the deviation between the guess and the calculated value is more than 5%

```
SizeHorizontal A1=4000um
SizeVertical A1=4000um
```

Output:

```
die width = 16480um, die height = 8930um
Horizontal: calculated 2244um, defined 2000um (difference 10.9%)
```

```
Vertical:   calculated 3396um, defined 3000um (difference 11.7%)
```

Guess 2

For the right number of bits in an array block there are different solutions possible depending on the aspect ratio. For this example a guess of 4mm by 1.5mm is assumed.

```
SizeHorizontal A1=4000um
SizeVertical A1=1500um
```

Output:

```
die width = 16480um, die height = 3930um
Horizontal: calculated 4478um, defined 4000um (difference 10.7%)
Vertical:   calculated 1740um, defined 1500um (difference 13.8%)
```

The resulting block size of 4.5mm by 1.7mm is close to the starting point. The die itself becomes however very long and thin (16.5mm by 3.9mm), so this is not the aspect ratio one would implement when building a DRAM which should be between square and a 2:1 aspect ratio.

Guess 3

A starting point of 2mm by 3mm gives a good aspect ratio. The calculated block size can be put into the input file to achieve a 0% difference.

```
SizeHorizontal A1=2000um
SizeVertical A1=3000um
```

Output:

```
die width = 8480um, die height = 6930um
Horizontal: calculated 2244um, defined 2000um (difference 10.9%)
Vertical:   calculated 3396um, defined 3000um (difference 11.7%)
```

5. References

[1] T. Vogelsang, "*Understanding the Energy Consumption of Dynamic Random Access Memories*", Proceedings 43rd Annual IEEE/ACM International Symposium on Microarchitecture, Atlanta December 2010.