MemoGen™ User Guide

MemoGen User Guide v1.3

Release Date: March 18th 2012

Contents

1. Introduction

The MemoGen software generates Algorithmic Memory™ cores by wrapping RTL-logic around physical memories in the specified target library. The memory cores are generated as RTL-IP along with the EDA-support script files required for integration of this core in the host chip.

The target audience for this document are chip-architects and engineers involved in the RTL, DFT and Physical (P&R, Timing, Power) integration of the host chip. It is recommended that the readers familiarize themselves with the functionality of Algorithmic Memory from the datasheet[1] provided by Memoir Systems Inc.

1.1. MemoGen Features

The MemoGen software has the following features:

- Algorithmic Memory core (RTL-IP) generation
- User specified Word-depth, Bit-width, Frequency parameters
- Ability to generate multiple memory cores per-run using batch-mode
- Area or Power optimization option for memory core generation
- Auto-generation of various EDA support files

1.2. MemoGen Scope

MemoGen generates memory cores for all Renaissance memory product families offered by Memoir Systems. The Renaissance product family offers up to 10X performance boost in memory performance. The **Renaissance 2X** product family offers memory cores which provide up to 2x MOPS[2] performance acceleration. The port capabilities supported are listed in Table 1-1:

Table 1-1: Renaissance 2X

Port-Capability	Description
1RW	Single read/write port memory core
1R1W	One Read Port AND One Write Port
1RW1W	(One Read/Write Port) AND (One Write Port)
1R1RW	(One Read/Write Port) AND (One Read Port)
2Ror1W	Two Read Ports OR One Write Port
2RW	Two Read/Write Ports (Dual Port)

[1] 'Renaissance mRnW Memory Core' datasheet; provided under NDA from Memoir Systems, Inc.

[2] Memory Operations Per Second

[3] The switch options used in extracting the various physical memory types included in the target library are specified in the

Renaissance 4X offers memory cores which provide up to 4x MOPS performance acceleration. The following additional port capabilities are supported:

Table 1-2: Renaissance 4X

Port-Capability	Description
1R2W	One Read Port AND Two Write Ports
1R3W	One Read Port AND Three Write Ports
2R1W	Two Read Ports AND One Write Port
2R2W	Two Read Ports AND Two Write Ports
3Ror1W	Three Read Ports OR One Write Port
3R1W	Three Read Ports AND One Write Port
4Ror1W	Four Read Ports OR One Write Port

Currently MemoGen supports only these two product families. Support for other families, including those which use eDRAM physical memories will be offered in the near future.

1.2.1. Functional Generator

Memogen also offers a mode ('Functional Generator') for architectural studies and early design exploration. A special binary is created to support this. This mode supports the same commands and user interface as in normal mode and also generates the same files for the specified user memory cores - only exception being that the synthesis capability of the generated RTL model(s) is disabled.

The Functional Generator will report area and power estimates for each requested memory. Hence it serves as a great tool for early design exploration. If a specific generator is required the user is urged to contact Memoir sales to purchase a fully functioning generator that supports synthesis.

1.3. MemoGen Output

The primary output of MemoGen is the RTL-IP model for the specified memory core. A specific datasheet describing the memory core is also generated. Sample scripts files required for incorporation of the core in the host chip are made available as well. The MemoGen deliverables are described below:

- RTL-model: Verilog file instantiating the underlying physical memories and an encrypted core-logic section.
- Datasheet describing the area, power, pins and other timing parameters specific to the generated memory core.
- Scripts: Script files to help with the physical design (synthesis, equivalence-check, post-layout static timing analysis) of the generated core. The scripts files are provided for **reference only** and cannot be used as is. Users are required to modify the script files to adapt them to their environment as appropriate.

2. MemoGen Installation

The MemoGen software is delivered as a tarball to the user.

2.1. O/S and Disk-Space Requirements

MemoGen requires a Linux environment for operation. It has been tested on Redhat Linux 5.8. The list of Linux versions supported is described in the '*INSTALL.txt*' file in the *doc* directory (refer next section).

The delivered tarball can be as large as 100 MB depending on the functionalities supported. Its uncompressed directory structure may require up to 250 MB.

2.2. Installation

The user should designate a suitable directory for installation of MemoGen, herein referred to as **'installation directory'**. The delivered tarball can be copied to this directory and uncompressed by typing the command:

$ ***tar xvfz*** *<tarball fn>*

The '*version-id*' string is part of the tarball file name and identifies the version number of the software. The command above, results in the creation of the '*<version-id>/memogen*' directory-tree with sub-directories as listed below.

rt, bin, LICENSES, flexlm, doc, src, libs

The *doc* directory contains MemoGen's documentation files. The generic datasheet ('Renaissance mRnW Memory Core') and the User-Guide (this document) for MemoGen are present here. The user should also read the Release-notes and Install.txt files present in this directory. The latter contains information related to the user's environment setup required to run MemoGen.

This directory may also contain a 'libnotes' file which describes the list of commands and the associated swithches/arguments used to extract library cells from the base memory compiler(s). This file is will NOT be present in case the base memory is of a generic type like 'Scaled 28nm'.

2.3. Working Directory

The user should also create a separate '**working-directory**' for generating memory cores. Typically it is placed somewhere in the user's chip design file hierarchy. It is strongly recommended that the user NOT generate memory cores in the installation directory.

2.4. Environment Variable Setting

The user is required to set **two environment variables** to appropriate values before running MemoGen. These are described in the *INSTALL.txt* file in the *memoir/doc* directory.

The **first** variable sets the **path for the target library**. This library (which is licensed by the user directly from the lib-vendor) contains Verilog and other views of the target library.[3]

The **second** variable (MEMOIR_LICENSE_FILE) sets the **path for license server**. The 'portnumber@my_host' setting for this variable should reflect the port-number/hostname of the machine

[3] The switch options used in extracting the various physical memory types included in the target library are specified in the 'libnotes' document in the *doc* sub-directory of the installation

where the user's FlexLM software is located. More details on license server management are explained next section.

Finally, the user should add '*memogen/bin*' to the Unix command search path. That allows the tool to be run from anywhere in the system, specifically from the working directory.

2.5. License Server Management

MemoGen uses FlexLM software for license management. FlexLM tools required to operate the license server are packaged in the original tarball file and get installed in the '*flexlm*' sub-directory (in the installation directory). The FlexLM license-file for the software is shipped separately. This file should be stored at a suitable place on the user's system. The tools provided are:

a) **lmgrd** : FlexLM license server daemon. This is used to start license server:

$ ***lmgrd –c*** *<path to license file>* ***-l*** *<logfilename>*

The header of the log file, created after the daemon starts, shows the port at which the license server is operating. This number should match the value of the port-number set in the LM environmental variable. Improper operation of the license server will cause Memogen to fail with a 'license checkout failed' exception.

b) **lmdown** : This is used to stop the license server using the following command

$ ***lmdown -c*** *<path to license file>*

3. Running MemoGen

This section provides information on using MemoGen to generate memory cores. The user should preferably be in the working directory to do this. It is strongly recommended that the user NOT run the software in the installation directory. MemoGen can be invoked by typing the command below:

$ ***memogen –help***

This results in the various switch options available for running MemoGen to be displayed on the screen. These are listed in Table 3-1. The user has to type the command and the associated switches/arguments exactly as shown in the table.

Table 3-1: MemoGen Switch Options

Switch[4]	Name	Description
help	Help	Prints list of all switch options
list	List	Lists Algorithmic Memory types available for generation
ver	Version	Displays MemoGen's version number
a*	Memory Core Type	Memory core type to be generated
f*	Frequency	Frequency (in MHz) of the memory core
w*	Words	Word count (or depth) of the memory core
b*	Bit-width	Bit-width of the memory core
name*	Memory core name	Name of the memory core (Alphanumeric characters only)
dir	Output Dir.	Directory where the memory core files are generated
fr	Force	Force overwrite of existing files if –name is repeated
in	File Input	Name of input file (.csv) containing multiple core specifications
nortl	No RTL	Estimate memory core parameters only, no RTL-IP model generation
Advanced Options		
ul	Use Libs	List of physical memory types to be used (Default is to use all available library memory cells)
el	Exclude Libs	List of physical memory types to be excluded (Default is to NOT exclude any memory cell types)
c	Cost-function	Cost-function used to optimize the generated memory core
tl	Target Latency	Generate memory cores with the specified (or lower) latency value
flopin	Input Flops	Parameter to control insertion of flops on all inputs of the core
flopout	Output Flops	Parameter to control insertion of flops on all outputs of the core
flopmem	Phy. Memory Flops	Parameter to control insertion of flops on all physical memory outputs within the core

Using the information above a sample 1R1W memory core of Words = 4K (4096) and Bit-width = 32, operating at a frequency of 500 MHz can be generated as follows (assume the user in in working-directory):

$ ***memogen*** ***–a*** *1R1W* ***-w*** *8192* ***-b*** *64* ***-f*** *500* ***-name*** *my_core* ***-dir*** *my_dir*

The resulting screen output is shown in Figure 3-1.

[4] "*" in the entry implies these options MUST be specified on the command line for MemoGen to run

```
Renaissance 4X (c) 2012 Memoir Systems. - memogen [v3.3.4191]  - All rights reserved.
[2012-07-26 16:20:12] INFO  - Loading libraries ..
[2012-07-26 16:20:15] INFO  - Searching for designs ..
[2012-07-26 16:20:17] WARN  - Clearing caches --> hup=83.3
[2012-07-26 16:20:18] INFO  -
Args: -a 1R1W -f 500 -w 8192 -b 64 -dir my_dir -name my_core
Libctx: scaled, 28nm, jsn, [rf_d_1rw,rf_s_1r1w,rf_s_1rw,sram_d_1rw,sram_d_2rw,sram_s_1rw], [1MHz-1000MHz], [1-524288], [1-2048]
+-----------------+-----------------------------------------+--------+--------+--------+--------+--------+---------+
| Memory core     | Instances                               | Area   | Gates  | Static | Max Rd | Max Wr | Latency |
|                 |                                         | (sqmm) | (k)    | (mW)   | (mW)   | (mW)   |         |
+-----------------+-----------------------------------------+--------+--------+--------+--------+--------+---------+
| C1 1R1W 8192x64 | scaled_28nm_rf_d_1rw_512x64_2@500MHz:16 | 0.1559 | 11.991 | 118.26 | 53.51  | 86.62  | 1.0     |
| 500MHz a=1      | scaled_28nm_rf_s_1r1w_512x79_2@500MHz:2 |        |        |        |        |        |         |
+-----------------+-----------------------------------------+--------+--------+--------+--------+--------+---------+
```

Figure 3-1: MemoGen Output

The user specified memory core information is reflected in the leftmost column under ‘Memory Core’. The ‘C1’ tag reflects the cut-id assigned to this memory core by MemoGen. Tags are needed when the tool generates multiple cores in a single command (refer –in option described later).

The next column (‘Instances’) shows all the physical memories from the target library and the respective instance counts used in the generation of the memory core. For instance, the memory core in Figure 3-1 has 16 instances of ‘rf_sp_hde’ memory and 2 instances of ‘rf_2p_hse’ memory.

The total estimated area is shown next. This consists of the areas of all the physical libraries plus the area of the overhead logic. An estimated gate-count (in k-gates) for the latter is shown in the ‘Gates’ column.

The next column shows the total static power (in mW) of the generated core. This is the sum of static powers of all the underlying physical memories.

The next two columns show the maximum read and maximum write dynamic power for the core. A note on calculating the total maximum dynamic power taking into account read and write activity-factors (AFs) is presented in Sec 4.3.

The last column shows the total latency (from inputs to outputs) of the generated memory core. It should be noted that Memogen’s algorithm RTL logic (used to generate a given core type) does not add any latency to the solution. The inherent latency of the generated core is simply the latency of the embedded physical memory library used within the core. The user can further influence this latency value by setting of the flop switch settings (described later).

The command above also results in the creation of directory ‘*my_dir*’ under the working directory. All MemoGen output files are generated in the appropriate sub-directories below that (see Section-4).

A note on –name values: MemoGen generates output Verilog file with the same name same as the character-strings specified for the –name option. The names can consist of only alphanumeric values, with no other special characters like hyphens, dots, commas (except underscore) included. Also the name cannot begin with a number.

The ***–dir*** option allows for the creation of a directory (if one does not exist already) for organizing the generated memory core files. For instance, the user may choose to keep all ‘1R1W’ memory cores in the same directory. Successive runs of the tool with –a 1R1W and the same –dir <name> results in the creation of memory core files in the same directory. Although not required, it is a good practice to always specify ***–dir*** option to better organize the generated results (Not doing so results in output files being dumped in the same directory where the command was invoked from).

The ***–fr*** option when used in conjunction with the same –name (and -dir) value as some prior run, allows for overwriting of generated files for that memory core name.

To see only the various estimated parameters for the generated memory core without generating RTL & other files, the user should use the ***–nortl*** option. (Note: -name value is still required when using this option).

Other relevant MemoGen switch options are described next.

3.1. Use-Library and Exclude-Library Options

These options allow the user to conveniently include (or exclude) certain physical memory types from the generated memory cores. This feature is especially useful if, for instance, the user wishes to generate the memory cores strictly out of a 'golden set' of physical memories. Such a set could be included as one of the target libraries. The –ul feature then allows the user to direct MemoGen to generate memory cores only from that library. The default for –ul option is to include all memory types.

The following example explains the usage of this option.

Assume that the user's target library consists of four different memory types: *sram_hd, sram_hs, rf, dram*. If the user wants to generate memory cores only from sram memory type then it can be done by specifying '-ul sram' switch option in the MemoGen command. It should be noted that '*sram*' is interpreted by the tool as '**sram**' – meaning any physical library with the string 'sram' in its name. Hence both *sram_hd, sram_hs* are included. To specify strictly *sram_hd*, the specification would be as shown below:

$ ***memogen*** ***–a*** *1R1W* ***-w*** *4096* ***-b*** *32* ***-f*** *500* ***-ul*** *sram_hd* ***-name*** *my_core*

The user could also choose to exclude a certain memory library type while generating memory cores. This can be done conveniently using the –el switch option. In the example above, to generate a memory core excluding all dram physical memory types (while allowing the use of all other types) the user types the command as follows:

$ ***memogen*** ***–a*** *1R1W* ***-w*** *4096* ***-b*** *32* ***-f*** *500* ***-el*** *dram* ***-name*** *my_core*

The user can include more than one lib as the value for –el (or –ul) switches. For instance, to generate a memory core excluding sram and dram libs, -el sram,dram can be specified on the command line.

3.2. Cost Function

The arguments of the '–c' switch option act as directives to optimize the generated memory core based on one of three performance parameters.

3.2.1. Total Area (-c a=1; Default option)

This option allows the user to generate a memory core which is lowest in total area. Total area of the memory core includes area of the base memories plus the area of overhead logic.

Due to logic overhead inherent in an Algorithmic Memory, for very small user specified core sizes the better solution may consist simply of stitched physical memories – without any RTL algorithms built around – from the user's target library. MemoGen can detect this and always presents the most optimal solution to the user.

For instance, if a user specifies a small size 1R1W memory core then MemoGen may determine that the best area-optimized solution is a stitched two-port physical memory in the target library. If a two-port memory was not available it will consider a dual-port physical memory (assuming that is available) and provide that as a solution if it provided a lower area than a pure 1R1W algorithmic memory.

3.2.2. Static-power (-c sp=1)

This option allows the user to generate a memory core which has the lowest static power. With this option the tool gives preference to physical memories with lower static power when generating cuts. In general, lowest area (a=1) also produces lowest static power memory but there are exceptions to this rule depending on the target physical memory library and the size of the memory core being generated. Hence the provision of 'sp=1' as a separate option.

3.2.3. Total power (-c p=1)

This option allows the user to generate a memory core which has the lowest total power. Total power includes static power plus dynamic power. Power optimized memory cores take a bit longer to run as compared to area optimized ones. To generate a memory core which is lowest in total power, the user can specify the following:

$ ***memogen*** *–**a** 1R1W* ***-w*** *4096* ***-b*** *32* ***-f*** *500* ***-c*** *p=1*

The user is also urged to read the special appendix (Appendix I) at the end of this document explaining the optimization process in more detail.

3.3. Latency Control & Flip-flop Insertion Options

The ***–tl*** switch allows the user to specify a target latency value for the generated memory cut. Memogen algorithms can trade off latency to achieve better optimization targets (Area or Power) for the generated core. Hence if the user can sustain higher latency in the host chip architecture then that value should be specified in the command line. Memogen will make the best effort to achieve that (or lower) value of latency in the generated core. It should be noted that latency of the generated core is the sum of:

a) Any latency introduced by Memogen's algorithm

b) The inherent latency of the embedded physical memories (0 for flow-thru, 1 for clk-ed, 2 for pipelined)

c) The latencies due to flop switch settings (described next)

A solution will be generated in any case with the actual value of latency reflected on the output screen and in the core-specific datasheet.

Specifying ***-flop flopin=1:flopcmd=1:flopmem=1:flopout=1:flopecc=1*** in the command line will result in all flop options being enabled. Various flop options can be selectively disabled by setting the corresponding flags to '0'. For example, input flops, output flops, and command flops can be enabled using ***-flop flopin=1:flopcmd=1:flopmem=0:flopout=1:flopecc=0*** or simply ***-flop flopin=1:flopcmd=1:flopout=1.***

Specifying ***–flop flopin=1*** in the command line results in the insertion of flip-flops on all the inputs (except Clock input) of the generated memory core. The user should consider setting this option for timing closure on the input path to the address bits of the embedded memories. Setting this option increases the latency value by 1.

Specifying ***–flop flopout=1*** in the command line results in the insertion of flip-flops on all the data outputs of the generated memory core. The user should consider setting this option for timing closure on the output path(s) from the embedded memories outputs to the outputs of the generated core. Setting this option increases the latency value by 1.

Specifying ***–flop flopcmd=1*** in the command line results in the insertion of flip-flops on all the inputs of the embedded memories except clock input. The user should consider setting this option for timing

closure on the input path(s) from primary inputs of the generated core to the inputs of embedded memories.

Specifying ***–flop flopecc=1*** in the command line results in the insertion of flops at the output of ECC checking logic and at the input of ECC computation logic. Algorithemic memories that support ECC support this feature. Setting this option increases latency by 1.

Specifying ***–flop flopmem=1*** in the command line results in the insertion of flip-flops on the data outputs of the embedded memories. Like in the case of –flopout option, the user should consider setting this option for timing closure on the output path(s) from the embedded memories outputs to the outputs of the generated core. Setting this option increases the latency value by 1.

Multiple –flop options can be specified by concatenating them. For example, input flops and output flops can be enabled together using ***–flop flopin=1:flopout=1***.

Note: The user will get a good handle on setting of the flop options only after performing a synthesis run on the generated memory core. The flop settings gives the users options to meet any timing closure issues in the various paths into and out of the generated core as shown in Figure 3-2.

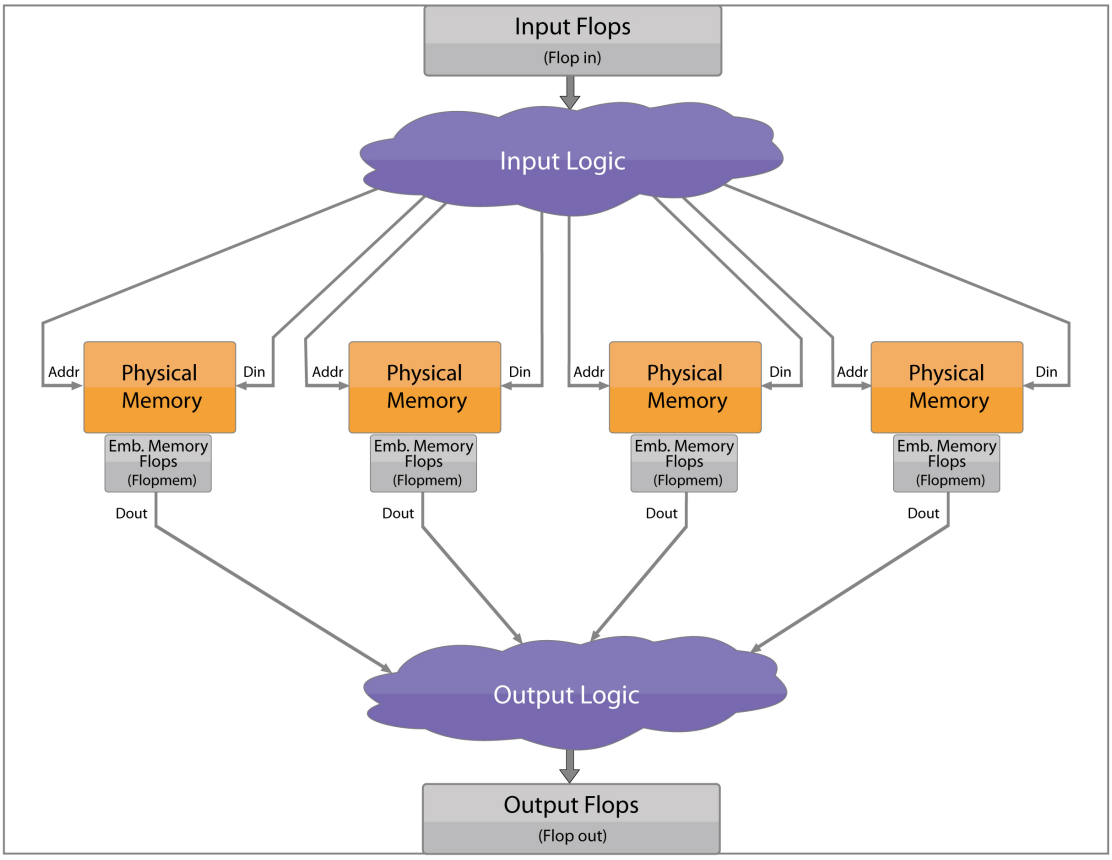

Figure 3-2: Memory Core Topology

3.4. File-In Option

The ***-in*** switch allows the user to generate multiple memory cores with a single command as shown below. The user specifies each core via an input file (*csv* format) as shown in Figure 3-3. A minimum of four columns, with headings in line-1 as shown, are required in this file. The individual memory core specifications are described in lines 2 through 6.

a	w	b	f
1R1W	131072	64	333
2RW	131072	32	500
2R1W	1048576	128	500
1R3W	32768	66	750
2Ror1W	262144	64	500

Figure 3-3: Input File for File-In Option

The command to invoke Memogen with the file input option is given below:

$ ***memogen –in*** *<input-file name>* ***-name*** *<append-string>* ***-dir*** *<output directory>*

The usage of *–name* option here is a bit different as compared to its usage in commands to generate a single instance. The argument-string provided with *–name* is appended to the ids assigned automatically by Memogen to each core. These ids are of the format 'C<n>' where <n> is the sequence number of the memory core in the .csv file. The resulting string "<name>-C<n>" is the full assigned name of the generated memory core. With the file input option all the generated files are persisted in the same directory specified with the ***–dir*** switch on the command line. The –dir switch is mandatory when using the file input option.

One other switch option (-nortl) must also be specified on the command line if the user intends to not generate any files. This option cannot be specified as a column within the .csv file.

All the options outlined in Table 3-1 can be specified in the first line of input file. If not specified the default values are assumed by Memogen. An example of a more elaborate .csv file using more than minimum required switch options is shown in Figure 3-4.

Some things to note:

a) Two (or more) input values for a column can be included by separating them with a comma (e.g. "dram,sram" in col -ul).

b) For options with binary (True or False) values, the user states 'TRUE' or 'FALSE' in the respective columns. E.g Values for the –flopout column in the figure below.

a	w	b	f	c	flopout	ul	el
1R1W	131072	64	333	a=1	TRUE	sram,dram,rf	
2RW	131072	32	500	a=1	FALSE	rf,sram	dram
2R1W	1048576	128	500	p=1	TRUE	sram,dram,rf	
1R3W	32768	66	750	ca=1	TRUE	sram,rf	dram
2Ror1W	262144	64	500	a=1	FALSE	dram,rf	sram

Figure 3-4: Example of Elaborate Input File

A special output file (in MS Excel format) is also generated. This file contains the solution summary for each memory core providing the user a quick view of the generated results.

4. MemoGen Output

MemoGen generates the RTL-model along with EDA support files for every memory core specified by the user. Table 4-1 describes the various files generated.

Table 4-1: MemoGen EDA files

EDA Function	Directory	Support File Name
RTL Model	Rtl	<name>.v; <name>_beh.v
Simulation	run_sample	run; run_beh
Datasheet	Doc	datasheet.txt
Physical Design	scripts	synthesis.tcl equivalence.dofile timing.tcl

4.1. RTL-IP Model

The Verilog model for the memory core (*<name>.v*) is generated in the *rtl* directory. **The user should use this file for integration in the host-chip**. It references the vendor-provided models for the physical memory instantiations (The path to this library is set by the user via the lib environment variable at installation time). The RTL model also references an encrypted parameterized logic file which implements the RTL-IP of the memory core. These files, specific to each memory core type, are present under the *'ip'* sub-directoory of the installation. The user has to use Cadence RTL compiler for synthesis of this file since encryption is done using Cadence tools.

A behavioral model (*<name>_beh.v*) which uses generic models for the physical memories (created by Memoir) is also made available to enable faster simulation.

4.2. Simulation

MemoGen also provides a sample test simulation environment for the generated memory core. The testbench (under the *run_sample*) directory has a stimulus vector generator which performs random writes/reads to the memory core model. Assuming the user has access to the '**nc-verilog**' Verilog simulator, a sanity-check simulation can be run by doing the following:

$ ***cd run_sample***

$ ***./run*** *<name>* : To simulate the Verilog memory core model

OR

$ ***./run_beh*** *<name>* : To simulate the behavioral Verilog memory core model

A successful test completion is indicated by '**Simulation PASSED'** output message.

4.3. Datasheet

A memory core-specific datasheet is generated by MemoGen in the doc directory. This document states the area, static and the maximum read and write dynamic power numbers for the memory core. In addition it lists the components of the physical memory library used to generate the memory core.

The formula to calculate maximum dynamic power is:

*Max. Dynamic Power = [(Max.read power) * AFr] + [(Max.write power) * AFw]*

Where AFr and AFw are read and write activity factors. They represent the fraction of the time the respective port is being used. For instance, for a 1R1W core, if reads happens only 50% of the time and writes happen 80% of the time then AFr = 0.5 and AFw = 0.8

For a multi-read (and/or multi-write) port memory core, the activity factor is combined across all the read (or write) ports. E.g. for a 2R1W memory core, if the activity on first read port is 0.5 while it is 0.8 on the other, the AFr for this core is 1.3. The maximum AFr for this core is 2.

Other memory core-specific timing parameters like Reset-to-Ready delay and Refresh scheme parameters (applicable only for eDRAM based memory cores) are also listed in this document. (These parameters are described in the generic Algorithmic Memory datasheet mentioned earlier).

4.4. Physical Design Reference Scripts

For every memory core, sample reference files are provided in the *scripts* directory to support synthesis, timing and formal verification processes.

The ***synthesis.tcl*** file is a sample synthesis script for synthesizing the generated RTL using Cadence

The ***equivalence.dofile*** file is a sample do-file used for matching cell-names between the RTL and user-synthesized gate-level netlist.

The ***timing.tcl*** file is a sample file to be used for post-layout static timing analysis of the generated memory core.

The user is also urged to read the special appendix explaining the typical IP-integration process included at the end of this document (Appendix II: Renaissance Memory Core On-chip Integration). The contents of the script files are examined in more detail in this appendix.

Appendix I: Optimization of Memory Cores using MemoGen

The MemoGen software generates Algorithmic Memory™ cores by wrapping RTL-logic around physical memories in the specified target library. The memory cores are generated as RTL-IP along with the EDA-support files required for integration in the host chip. This application note gives an overview of the optimization process used by MemoGen in generating the specified memory core.

The user can specify an optimization parameter (-c cost-function) when invoking MemoGen. This guides the tool to generate a solution wherein this parameter is maximized. The choices for this parameter are:

a) **Area** (command line switch option ' –c a=1') (This is the default option)

b) **Static Power** (command line switch option '-c sp=1')

c) **Power** (command line switch option '-c p=1')

MemoGen's working for each of these options is described next

Area

With this option MemoGen attempts to minimize the total area of the generated core. Architectural optimization is the main technique used to realize area savings. There are a few other techniques which the tool uses to minimize the overall area even when algorithmic memory is not involved. Some of these are described next:

Sub-packing: This technique involves figuring out the optimal memory size which packs multiple memories of lower bit-width into a single memory of higher bit-width. For example, packing 8 4-bit memories into a single 8x32-bit memory may lead to lower overall area. This technique is mainly used in the course of Architectural optimization where a user specified core is typically built out of multiple smaller memories for implementation of algorithms (described later).

The technique can also be used to optimize single memory instances of size W (rows) x B (bits) (where W > B). For instance it may turn out that a memory tile of size B (rows) x W (bits) is actually lower in area. So the solution is better built with a BxW memory, with some external bits-side muxing logic to render B bits out of W. e.g. Generate a user requested 16Kx8 memory with a 8Kx16 tile using the MSB address input bit to control a bit-side mux to render 8 bits out of a 16-bit memory output.

Super-packing: This technique is used when 'stitching' multiple smaller memories together to generate a single bigger memory which the user has requested. It enables Memogen to choose the right 'divisor ratio' to optimize the overall area of the solution. For instance, when a user requests a large single-port memory, the tool 'stitches' together 'n' physical memories by judiciously choosing 'n' (the 'divisor ratio') to optimize the overall area.

The technique involves packing along both vertical (rows) and horizontal (bits) dimensions. In case of vertical packing, Memogen has the ability to choose 'non power-of-2' (word-sized) memories when generating the solution. Normally, a user constructing a solution manually, would tend to choose only power-of-2 memories. For instance, use one 8Kx32 and one 2Kx32 memory to realize a 10Kx32 memory. But depending on the underlying physical library, a more area efficient solution may consist of using two 5Kx32 memories (non-powers of two).

An example of horizontal packing is when Memogen determines if a 16Kx128 size memory core is best generated using 8 instances of 16Kx16 tiles or from 16 instances of 16Kx8 tiles or any other combination like that.

Combining sub-packing and super-packing techniques allows Memogen to generate the most optimal memory cores possible from the given physical memory library.

Architectural optimization: This is the main technique used by Memogen to generate the user specified memory core. It typically involves breaking down the specified core into smaller sized banks. Some overhead memories along with logic are then added to implement caching, meta-data store and other algorithms embodied within Memogen. The exact algorithm to implement depends on port-capability (mRnW) and latency specified by the user.

The total area consists of areas of the physical memories (or 'tiles) used to make up the user space plus areas of any overhead (or buffer) memories used in the implementation of the core's IP. To minimize logic complexity the size of buffer tiles is typically matched with those of physical memories chosen to implement the user memory space. To minimize the area penalty due to overhead memories MemoGen tends to choose the smallest buffer sizes. However smaller memory-tile sizes tend to increase the overall area of the solution due to lower densities associated with smaller memories. In the end, an optimal point is achieved where the increase in area is balanced out by the reduction in overhead.

Also, lower internal buffer size implies higher physical memory instance counts. This may lead to higher dynamic power (depends on the core type). Hence power performance may not be optimal with this parameter option.

Users can improve the execution time of the tool by selecting (or excluding) the best density physical libraries using the '-ul <lib-type>' (or –el <lib-type>) switch options. For instance a best area solution could be achieved faster by stating '-ul dram' and '-el sram' in the command line. The tool finds the best area solution even without this option. But the option helps speed up the search/execution time of the tool by ignoring the sram physical library altogether.

Static Power

With this option MemoGen generates a core which has the lowest possible static power for the specified size. Usually lowest area solutions will also lead to lowest static power solutions. But there are exceptions to this rule depending on the characteristics of the underlying physical memories and user-specified core size. For instance, a sram memory may have lower static power while having higher area as compared to a similar sized register-file memory. This typically happens at the size-transition point between the two physical libraries (high end range of register-files and low end range of srams, where their capacities are roughly equal). The user has the option to improve the execution time of the tool by including (or excluding) libraries which have the best static power characteristics.

Users wishing to achieve the lowest standby power should use this option. It does not necessarily lead to the lowest total power which is the topic of the next section.

Power

Users wishing to achieve lowest total power (Static + Dynamic) for their memory cores should use this option. Since the tool tries to minimize both static and dynamic power metrics the execution time is usually higher for this option.

To minimize static power, MemoGen chooses physical memories with the lowest static power. But these memories may not necessarily have the lowest area. It then chooses internal buffer sizes such that the total instance count is minimized (this is opposite of what it does for the case of Area optimization). It does so since lower instance count typically leads to lower dynamic power.

As with earlier options, the user can help speed up execution time by including (or excluding) physical memories with best static power / dynamic power characteristics.

Appendix II: Renaissance Memory Core On-chip Integration

The MemoGen software generates Algorithmic MemoryTM cores by wrapping RTL-logic around physical memories in the specified target library. The memory cores are generated as RTL-IP along with EDA script files to help with the physical design integration. The following sections of this application note describe the typical flow used for physical design and the help offered by script files to ease this process. It should be noted that these files are provided for 'reference-only'. Users have to customize them to their environments.

Synthesis

The very first step for the user is to synthesize the generated memory core's RTL-model. The HDL (Verilog) representing the memory core is converted to a gate-level netlist in this process. These models (*name.v*)[5] are made available in the *rtl* sub-directory of the user's working directory. Since the core-logic of this model is encrypted with Cadence tools the user must use Cadence's RTL Compiler for the synthesis process.

The synthesis help file (***synthesis.tcl*** – made available in the scripts sub-directory) is organized into **four sections**. The **first section** deals with the setup of global attributes required by the synthesis tool. The user must customize this section by providing choices for the various attributes. Examples of attributes include synthesis and mapping effort levels, wireload_mode, search-pathnames for synthesizer input files like the main design file, associated hdl and lib files, pathnames for synthesizer output, report and log files, etc. Commands to create the output directories, in case they don't already exist are also included.

The **second section** reads in the library and HDL-design files. The user can also read in other library files like *lef* and *capacitance_table* files. These are needed if a 'layout-aware' synthesis is to be performed (directives to read these are currently commented out – the user is responsible to provide these files). Commands to 'elaborate' the design and report any unresolved hierarchies, are also included. The last part of this section deals with the reading in of the synthesis constraints file. This file includes settings for design 'constraints' like clock frequency, input/output delays, input drive strengths, output loads, etc. Since these constraints are specific to the user's environment, this file has to be provided by the user.

The commands to begin synthesis are given in **section three**. Synthesis is typically done as a two (or three) step process. The input design is first converted into a generic intermediate form. This is then 'mapped' to a gate-level netlist in step two. By splitting the synthesis command in two steps, the user can control the 'effort-level' of the synthesizer for each step, thereby controlling the overall execution time. Also this allows for optimization of the design in terms of area and power. Step two (mapping to gates) is typically done with a lower power/area gate libraries. Any violating timing paths here are then fixed with an 'incremental' synthesis command which is done as step three. For this step the user specifies additional libraries (typically high speed, low Vt libs) to allow the synthesizer to fix the broken timing paths. This way bulk of the design is realized with lower power/area gates with only a sprinkling of higher power gates to meet any critical timing paths.

Once the design is synthesized, the last task to do is to generate all the output files. This is done in **section four**. Commands to write the main design database, gate-level Verilog netlist, timing constraints file to guide the layout tool, logic-equivalence script-file any other reports are present here.

Logic Equivalence Check (LEC)

LEC process consists of checking the HDL design ('golden' design) versus its gate-level netlist ('revised' design). It is done not so much as so to check the integrity of the synthesizer tool but to guard against any omissions in the generated gate-level netlist. One way such omissions can occur is if some section

[5] User's should NOT use name_beh.v model for synthesis. This model is made available strictly for faster simulation times.

of the HDL file was 'black-boxed' for any reason and the user later forgot to unblock it before synthesis process. Another reason for doing LEC is after implementing minor ECOs. These ECOs are typically implemented with manual fixes to the gate-level netlist/layout of the design. The HDL file is also changed to reflect this change however no re-synthesis is done, the danger being a full re-synthesis may completely change the gate-level netlist. To confirm that the two changes are equivalent a LEC is necessary.

MemoGen generates the ***equivalence.dofile*** script to help with the LEC process after the user has synthesized the generated core. The LEC process involves three essential steps:

a) Reading in 'golden' (HDL) design file and the 'revised' file (synthesized gate-level netlist).

b) Providing naming/re-naming rules to the LEC tool to help it identify equivalent points (also called 'key points') between the two files. This step is necessary since after synthesis the register names specified in the HDL file may be renamed differently by the synthesis tool. The set of key points typically include all the inputs, outputs and the flip-flops (or registers) internal to the design.

c) Issuing the 'compare' command and generating output reports.

The provided dofile is organized into sections to implement the above steps. Once LEC check is done the design is ready for layout. This topic is discussed next.

Block placement and Layout

In some physical design flows, placement of physical memories could be done as the very first step. The extracted placement information is then used to drive a 'placement aware' synthesis process. The wire-delays used by the synthesis tool are more accurate in such a flow, leading to lesser timing violations post-layout.

For best timing margins, it is recommended that all the physical memory blocks be placed close to each other with the core logic laid out in between. One suggested way is shown in Figure 0-1. Layout of memory core gates should be done once placement of blocks is fixed.

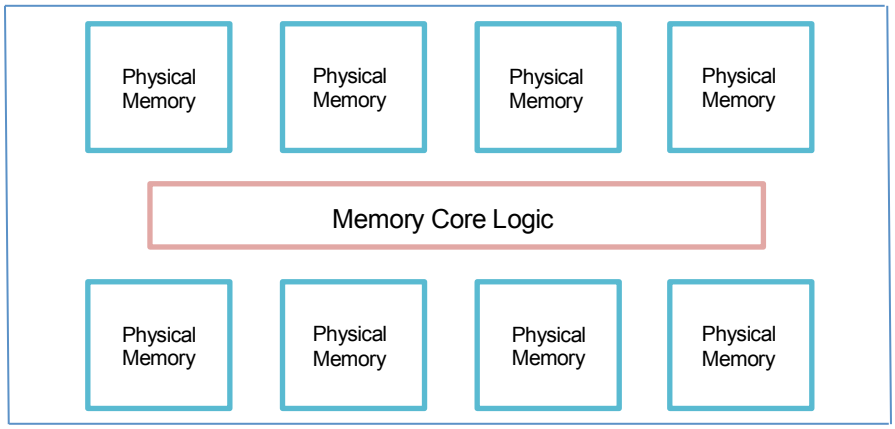

Figure 0-1: Physical Design Placement Suggestion

Post Layout Timing Closure

The last step in physical design process is post-layout static timing analysis (STA). The script to help with this process is provided in the file name ***'timing.tcl'*** in the scripts output directory. The file provided is quite elaborate and provides commands for doing SI (Signal Integrity) simulation besides ST analysis. The initial sections deal with setting attributes and variables. The reading of design files and setting conditions and modes for SI/STA runs are done next. Later sections deal with generation of reports and

output files, including the SDF file (post-layout annotated timing delay file) required for gate level simulations.

5. Errata and Revision Change Log

Table 5-1: Errata

Version	Description	Workaround
V1.0	-	
V1.1		
-nofile option	The –nofile switch option for Memogen command line has been removed	User should use –nortl option instead
Updated information in core-specific datasheet	The following parameters and information have been updated from the generated core-specific datasheet: a) Reset-to-Ready delay value b) Refresh window (Tref) value c) Read-Error usage, d) Physical mapping information (mapping between physical address space and reference-designators of the memory tiles used to generate core	Contact Memoir Sales for the missing information related to cores generated using Memogen

Table 5-2: Revision Change Log

Version	Date of Release	Notes
V1.0	Mar 1st 2012	Initial Release.
V1.1	Oct 1st 2012	Ren4X Functional Generator, Latency additions
V1.2	Nov 15th 2012	Minor updates to text files

For any additional information or support questions related to MemoGen software, feel free to contact Memoir Systems at the address / email below:

Memoir Systems Inc.
2350 Mission College Blvd. # 1275
Santa Clara, CA 95054

Phone:1-408-550-2382

Email: support@memoir-systems.com