# ZeBu® Sparse Memory Application Note

Version V-2024.03-1, July 2024

Feedback

# Contents

About This Book . . . . . 4

Contents of This Book . . . . . 4

Related Documentation . . . . . 4

Synopsys Statement on Inclusivity and Diversity . . . . . 5

**1. Sparse Memory Overview** . . . . . 7

**2. Memory Profiler** . . . . . 9

**3. Compiling for Sparse Memory** . . . . . 12

Enabling Sparse Memory . . . . . 12

Use Model for Implementing Sparse Memory . . . . . 13

Reporting . . . . . 15

**4. Runtime** . . . . . 17

Performance Consideration . . . . . 18

**5. Limitations** . . . . . 19

Feedback

# Preface

This chapter consists of the following topics:

- About This Book
- Contents of This Book
- Related Documentation
- Synopsys Statement on Inclusivity and Diversity

## About This Book

The *ZeBu® Sparse Memory Application Note* provides details of implementing a memory as Sparse.

## Contents of This Book

The *ZeBu® Sparse Memory Application Note* has the following chapters:

| **Chapter** | **Describes...** |
|---|---|
| Sparse Memory Overview | Sparse memory in emulation |
| Compiling for Sparse Memory | Enabling and analyzing results for sparse memory |
| Runtime | Using sparse memory at runtime and understand impact |
| Memory Profiler | Using memory profiler |
| Limitations | Limitations of sparse memory implementation |

## Related Documentation

| **Document Name** | **Description** |
|---|---|
| *ZeBu User Guide* | Provides detailed information on using ZeBu. |
| *ZeBu Debug Guide* | Provides information on tools you can use for debugging. |

| Document Name | Description |
|---|---|
| *ZeBu Debug Methodology Guide* | Provides debug methodologies that you can use for debugging. |
| *ZeBu Unified Command-Line User Guide* | Provides the usage of Unified Command-Line Interface (UCLI) for debugging your design. |
| *ZeBu UTF Reference Guide* | Describes Unified Tcl Format (UTF) commands used with ZeBu. |
| *ZeBu Power Aware Verification User Guide* | Describes how to use Power Aware verification in ZeBu environment, from the source files to runtime. |
| *ZeBu Functional Coverage User Guide* | Describes collecting functional coverage in emulation. |
| *Simulation Acceleration User Guide* | Provides information on how to use Simulation Acceleration to enable cosimulating SystemVerilog testbenches with the DUT |
| *ZeBu Verdi Integration Guide* | Provides Verdi features that you can use with ZeBu. This document is available in the Verdi documentation set. |
| *ZeBu Runtime Performance Analysis With zTune User Guide* | Provides information about runtime emulation performance analysis with zTune. |
| *ZeBu Custom DPI Based Transactors User Guide* | Describes ZEMI-3 that enables writing transactors for functional testing of a design. |
| *ZeBu LCA Features Guide* | Provides a list of Limited Customer Availability (LCA) features available with ZeBu. |
| *ZeBu Synthesis Verification User Guide* | Provides a description of zFmCheck. |
| *ZeBu Transactors Compilation Application Note* | Provides detailed steps to instantiate and compile a ZeBu transactor. |
| *ZeBu zManualPartitioner Application Note* | Describes the zManualPartitioner feature for ZeBu. It is a graphical interface to manually partition a design. |
| *ZeBu Hybrid Emulation Application Note* | Provides an overview of the hybrid emulation solution and its components. |

## Synopsys Statement on Inclusivity and Diversity

Synopsys is committed to creating an inclusive environment where every employee, customer, and partner feels welcomed. We are reviewing and removing exclusionary language from our products and supporting customer-facing collateral. Our effort also includes internal initiatives to remove biased language from our engineering and working environment, including terms that are embedded in our software and IPs. At the same time, we are working to ensure that our web content and software applications are usable to people of varying abilities. You may still find examples of non-inclusive language in our software or documentation as our IPs implement industry-standard specifications that are currently under review to remove exclusionary language.

# 1

# Sparse Memory Overview

The sparse memory is defined as follows:

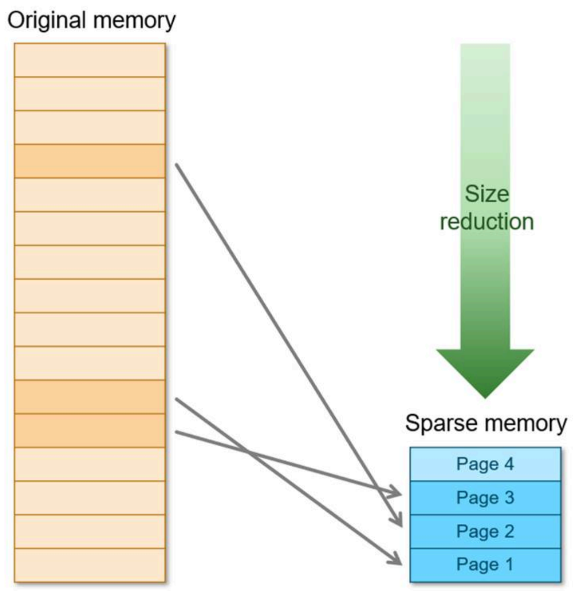
- An instance or module and MDA signal
- A number of pages (power of 2 number, if not they are rounded to the next power of 2)
- A size for the page (expressed in number of words, or bits; these are power of 2 numbers; otherwise, they are rounded to the next power of 2)

Implementing a memory as sparse can help to:

- Reduce the physical resources required to map the memory
- Improve performance by reducing the hardware size and allow to map on faster primitives (that is, ZRM to BRAM/URAM)

The following figure depicts an illustration of converting the original memory into a sparse memory:

*Figure 1 Sparse memory implementation*



To select the parameters, a profiler is available as described in the following:

- A word represents the data for one address location. The word size is the width of the memory.
- If multiple memories are implemented as sparse, the pages are not shared and unused resources from one memory cannot be used by another one.
- If the parameters provided create a memory of the same size as the original one, the command is ignored.
- Since page size and number of pages are power of 2, the largest size for a sparse memory is 50% of the original. Next sizes are 25%, 12.5%, and so on.

At runtime, the memory can be transparently accessed, both from hardware DUT or software testbench. When a memory is loaded with the backdoor, data is dumped into files, which will be used to initialize the pages when the design accesses them.

Feedback

# 2 Memory Profiler

For identifying the parameters for the sparse memory (for example, number of pages and size), a profiler is available to extract these parameters (only supports profiling of ZRM-memory accesses).

The profiler assumes page-size of 1024 words and captures which pages are used. To compile a design, enable the profiler and then run test cases to capture the memory usage.

The following advanced `zTopBuild` command enables the profiler:

- For one memory:

  ```
  ztopbuild -advanced_command { sparse_memory -instance <path to memory>
   -profile}
  ```

- For all memories:

  ```
  ztopbuild -advanced_command { sparse_memory -under_instance <top name>
   -profile}
  ```

The reporting in `zTopBuild.log` is shown as follows:

```
#   step SPARSE MEMORY : Starting
#   step USER TYPE SELECTION : Type 'zrm' has been chosen by user for
 memory 'mem_ZMEM_array_sparseMem'.
#   step SPARSE MEMORY : Profiled info for memory=top.u0_dut.u0_mem.array
 will be written at runtime in file with prefix "memory_profiler_pmem_0"
#   step USER TYPE SELECTION : Type 'zrm' has been chosen by user for
 memory 'mem_0000_ZMEM_array_sparseMem'.
#   step SPARSE MEMORY : Profiled info for memory=top.u1_dut.u0_mem.array
 will be written at runtime in file with prefix "memory_profiler_pmem_1"
#   step SPARSE MEMORY : Done in    elapsed:0.10 s   user:0.12 s
 system:0.1 s     %cpu:126.83       load:0.25       fm:253254 m
 vm:1159 m       vm:+0 m      um:735 m       um:+1 m
```

At the end of the run, the profiler generates a CSV file per-memory per test-vector, using the prefix mentioned in the log file and concatenating the date, for example, `memory_profiler_pmem0_31_08_2022_16_49_04.csv`.

The file starts with the memory information, such as full path, word size, and depth.

After that, each line gives a possible combination for the sparse memory parameters. The last part lists the pages hit during profiling.

The columns are:

- `words_per_page`: word per page (with word width = memory width)
- `number_of_pages`: number of pages
- `effective memory depth`: depth of the page memory: word_per_page x number_of_pages

```
memory instance path,top.u0_dut.u0_mem.array
word width=256,depth=536870912
words_per_page,number_of_pages,effective memory depth,
1024,3,3072
2048,3,6144
4096,3,12288
8192,3,24576
16384,3,49152
32768,3,98304
65536,3,196608
131072,3,393216
262144,3,786432
524288,3,1572864
1048576,3,3145728
2097152,2,4194304
4194304,2,8388608
8388608,2,16777216
16777216,2,33554432
33554432,1,33554432

67108864,1,67108864
134217728,1,134217728
268435456,1,268435456
536870912,1,536870912
Detail of pages index used for words_per_page=1024 settings
0
1024
16384
```

The preceding file shows that only 3 pages of size 1024 words are accessed. Hence, it requires a sparse-memory with 3072 words.

Choosing a bigger page-size would lead to less pages, but the required physical-memory can increase and a significant amount would be unused.

There is a utility script in the release to generate the UTF command with the selected page size based on this report.

```
$ZEBU_ROOT/etc/scripts/analyze_profiler_result.sh <words_per_page> <all
 profiler files to analyze>
```

```
% $ZEBU_ROOT/etc/scripts/analyze_profiler_result.sh 1024 `find <path>
 -name memory_profiler_pmem*`
#--------- auto generated sparse memory command -----------
sparse_memory -words_per_page 1024 -num_pages 3 -instance
 top.u0_dut.u0_mem.array
```

Feedback

# 3

# Compiling for Sparse Memory

The sparse memory is defined during the `zTopBuild` step with the advanced `sparse_memory` command.

The section describes the following sub-sections:

- Enabling Sparse Memory
- Use Model for Implementing Sparse Memory
- Reporting

## Enabling Sparse Memory

To enable sparse memory, use the following commands: `sparse_memory`

*Table 1 Sparse memory options*

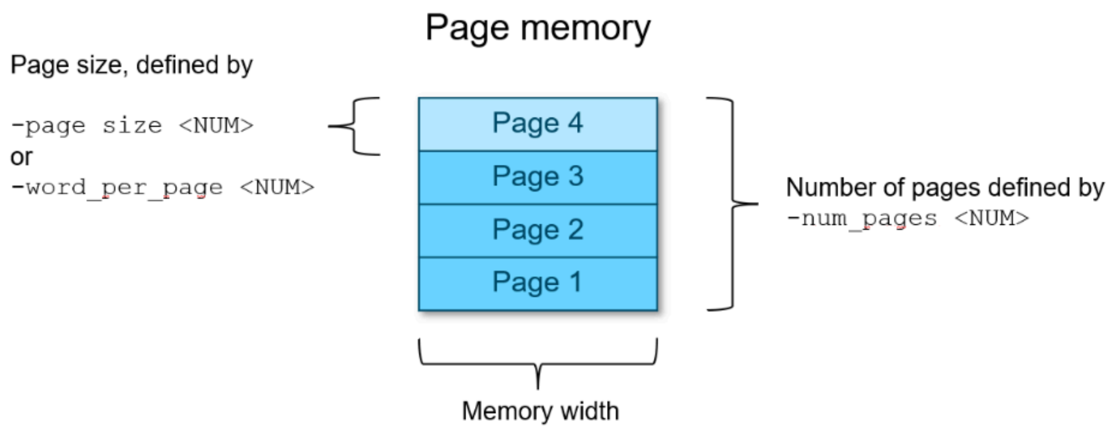
| Command Name | Description |
|---|---|
| `[-instance <hierarchical path to array]` | Converts array instance to sparse implementation.Mutually exclusive with *-module*. |
| `[-module <module name>]` | Specifies module if all instances of this module need to be converted to sparse.Mutually exclusive with *-instance*. |
| `[-signal <name of mda>]` | Specifies the name of the memory signal in the module specified by the *-module* option. |
| `[-under_instance <instance path>]` | Useful to target memories inside encrypted IP. |
| `-num_pages <integer>` | Creates number of page-memories. |
| `[-words_per_page <integer>]` | Specifies the depth of each page (internally rounded to 2^n)Mutually exclusive with *page_size* |
| `[-page_size <integer>]` | Specifies (*depth* * *width*) of page-memory, expressed in bits.Mutually exclusive with *words_per_page*. |
| `[-verbose]` | Enables to dump more information |

*Table 1 Sparse memory options (Continued)*

| Command Name | Description |
|---|---|
| `[-size <integer>]` | Implements the memory as sparse with `<integer>`% of the total capacity of the original memory. |
| `[-zrm_promotion <on_chip/auto/disable>]` | The ZRM promotion options are:*on_chip*: Enables the tool to enforce the conversion of ZRM to BRAM at any cost.*auto*: Disables resource increase, but ZRM promotion to on-chip memories is applied if it fits in existing resources. This is the default option.*disable*: Disables the tool not to convert ZRM to on-chip memories. |

The following figure depicts an illustration of page memory:

*Figure 2 Page memory*



## Use Model for Implementing Sparse Memory

The following example use models provide the details of implementing a memory as sparse:

### Example 1

Converts the memory-array `top.A.B` to sparse implementation, with 4 pages of 8192-bit size, that is, sparse memory size = 4*8192 = 32.768 bit. If `top.A.B` is a ZRM memory, ZRM promotion will be attempted if fits in current number of boards.

```
sparse_memory -instance top.A.B -num_pages 4 -page_size 8192
```

## Example 2

Converts the memory-instance *top.A.B* to sparse with 4 pages of 8192-bit size, that is, sparse memory size = 4*8192 = 32.768 bit. ZRM promotion will be attempted if no board-count increase is required. `sparse_memory -instance top.A.B -num_pages 4 -page_size 8192 -zrm_promotion auto`

## Example 3

Converts the memory-instance `top.A.B` to sparse with 4 pages of 1024 depth (10-bit address) each, that is, sparse memory is represented by 12-bit address. ZRM promotion will be attempted which may increase board requirement.

```
sparse_memory -instance top.A.B -num_pages 4 -words_per_page 1024
 -zrm_promotion on_chip
```

## Example 4

Converts instance belonging to the `mem_model` module specified by signal `mem` to sparse with 4 pages of 1024 depth (10-bit address) each, that is, sparse memory is represented by 12-bit address that dumps more messages.

```
sparse_memory -module mem_model -signal mem -num_pages 4 -words_per_page
 1024 -verbose
```

## Example 5

Implements the memory-instance *top.A.B* as sparse, such that if the original memory size 16 Gbits, the sparse memory takes up only 50% of its total capacity, which is 8 Gbits.

```
sparse_memory -instance top.A.B -size 50
```

With the `-size <>` option, `-num_page` and `-words_per_page` are automatically computed internally. User has the option to control the granularity by providing either `-num_page` or `-words_per_page`, as given in Example 6 and 7.

## Example 6

Implements the memory-instance *top.A.B* as sparse, such that if the original memory size 16 Gbits, the sparse memory takes up only 50% of its total capacity, which is 8 Gbits, with 4 pages.

```
sparse_memory -instance top.A.B -size 50 -num_page 4
```

### Example 7

Implements the memory-instance *top.A.B* as sparse, such that if the original memory size 16 Gbits, the sparse memory takes up only 50% of its total capacity, which is 8 Gbits, with pages of 1024 depth (10-bit address) each.

```
sparse_memory -instance top.A.B -size 50 -words_per_page 1024
```

**Note:**

`-num_pages` and `-words_per_page` are rounded to (2)^n.

## Reporting

The basic reporting in `zTopBuild.log` shows as follows:

- Status of the memories requested to be implemented as sparse
- Original size
- Sparse size and instrumentation size

The sparse memory reporting is shown as follows:

```
#   step SPARSE MEMORY : Starting
#   step USER TYPE SELECTION : Type 'zrm' has been chosen by user for
 memory 'mem_ZMEM_array_sparseMem'.
#   step USER TYPE SELECTION : Type 'zrm' has been chosen by user for
 memory 'mem_0000_ZMEM_array_sparseMem'.
#   step TABLE: Summary of Sparse Memory conversion  :
# +----+-----------+-----------------------+---------+------------+
# |S.No|Inst/Module|Name|Converted|Original Memory|Sparse
 Memory|Instrumentation Memory|Wall Time|
# +----+-----------+-----------------------+---------+------------------+
# |    |           |                       | (YES/NO)|Size, Type, #
 Ports|Size, Type, # Ports|   Size, Type, # Ports|    (sec)|
# +----+-----------+-----------------------+---------+------------------+
# |   1|       Inst|top.u1_dut.u0_mem.array|      YES|       4 MB, zrm,
 3|     16 KB, bram, 3|     3072 B, ramlut, 4|     4.0s|
# |   2|       Inst|top.u0_dut.u0_mem.array|      YES|       2 GB, zrm,
 3|     64 KB, bram, 3|        48 KB, bram, 4|     3.2s|
# +----+-----------+-----------------------+---------+------------------+
# step SPARSE MEMORY : Done in  elapsed:7 s   user:0.34 s   system:0.15
 s       %cpu:6.81      load:16.80       fm:198685 m     vm:1171 m
 vm:+12 m      um:756 m      um:+14 m
# step SPARSE MEMORY : Detailed report printed in zTopBuild_report.log
```

The `zTopBuild_report.log` file contains a complete report on memory conversions.

```
32.Sparse Memory Conversion.
============================
########### CONVERSION SPECIFIED BY MODULE BEGIN ###########
No Memory module for sparse conversion.
########### CONVERSION SPECIFIED BY INSTANCE BEGIN ###########
Number of Memory instances for sparse conversion : 2
1) top.u1_dut.u0_mem.array
        Original Memory : array
-------------------------------------------------------------------
|Type |Size |Depth |Width |Num Port |
-----------------------------------------------------------
|zrm |4 MB |1048576 |32 |3 |
-----------------------------------------------------------
Sparsed Memory (Page Memory) :
-----------------------------------------------------------
|Type |Size |Depth | Width |Num Port |
-----------------------------------------------------------
|bram |16 KB |4096 |32 |3|
-----------------------------------------------------------
Instrumentation Memory (Page Table) :
-----------------------------------------------------------
|Type |Size |Depth |Width |Num Port |
-----------------------------------------------------------
|ramlut | 3072 B |1024 | 3 | 4 |
-----------------------------------------------------------

Resource used after sparse memory conversion
+-----------------+----+----+------+----+----+
|Resource used by| LUT| REG|LUTRAM|BRAM|URAM|
+-----------------+----+----+------+----+----+
|      Controller|2850|1079|     0|   0|   0|
+-----------------+----+----+------+----+----+
|      Page Table| 163|  76|    48|   0|   0|
+-----------------+----+----+------+----+----+
|     Page Memory| 101| 152|     0|   4|   0|
+-----------------+----+----+------+----+----+
```

Feedback

# 4 Runtime

This section describes the usage of sparse memory at runtime.

A memory implemented as sparse can be used like any memory at runtime such as, load, store, clear, pattern initialization, and so on. When a sparse memory is initialized through the backdoor, the content is stored in a file in a new directory inside the current working directory: `sparse_memory_<date_time>`

```
sparse_memory_20_03_2023_10_51_54/
+-- mem_0
+-- mem_1
```

This file size will be identical to the memory size. These files are cleaned automatically at the end of the run. The creation of these files can be configured at runtime using `set_app_var`:

- Change the base directory to write the memory content:

  ```
  set_app_var sparse_base_dir <string>
  ```

- Configure if the files are to be deleted at the end of the run:

  ```
  set_app_var sparse_clean_files [true|false]
  ```

If the runtime accesses more pages than specified at compile-time, a page overflow happens and runtime is interrupted.

The following error message shows the path of the memory-array and its characteristics.

```
-- ZeBu : zRci : ###### SPARSE MEMORY SETTING #############
-- ZeBu : zRci : Instance name = top.u0_dut.u0_mem.array
-- ZeBu : zRci : Num of Pages  = 3
-- ZeBu : zRci : WordsPerPage  = 1024
-- ZeBu : zRci : AddrWidth     = 27
-- ZeBu : zRci : AddrOffset    = 10
-- ZeBu : zRci : Sparse Memory: Num of pages accessed=4 (limit=3)
-- ZeBu : zRci : Sparse Memory: address at port(smp2_r2) is
 11111111111111111**********
-- ZeBu : zRci : Sparse Memory: address at port(smp0_w0) is
 11111111111111011**********
-- ZeBu : zRci : Sparse Memory: End message
-- ZeBu : zRci : WARNING : "ZEBU_DEBUG_EXCESS_ERROR_LIMIT=100" has been
 evaluated.
```

```
-- ZeBu : zRci : ERROR : ZPRIV2609E : Stopping Emulation because
 of following error(s) Error due to Page OverFlow for memory
 top.u0_dut.u0_mem.array
```

## Performance Consideration

When the memory is initialized with some content, it must be uploaded into the hardware on every page miss. The consequence is a short clock stoppage on every page miss to upload the new used page with the proper data.

If the page miss is frequent, it effects the runtime throughput. Therefore, it is not recommended to create a sparse model with very small pages that might create frequent interruptions.

When a large ZRM memory is remapped into FPGA memory, backdoor access might be slower.

# 5
# Limitations

The following is the sparse memory limitation:

- Sparse memory is supported for ZeBu Server 4 , ZeBu Server 5 and ZeBu EP1.