ZeBu®
EEPROM Memory Model
User Guide

Version V-2024.03-SP1, February 2025

Copyright Notice and Proprietary Information

Contents

About This Manual

Overview

This guide describes how to use the ZeBu EEPROM memory model with your design being emulated in ZeBu.

Related Documentation

For more information about the ZeBu supported features and limitations, see the ZeBu Release Notes in the ZeBu documentation package corresponding to your software version.

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 Introduction

The ZeBu EEPROM synthesizable memory models are based on ZeBu memory primitives and are implemented in the ZeBu FPGAs.

The ZeBu EEPROM memory models offer a multi-MHz operating frequency on all ZeBu platforms. Each ZeBu EEPROM memory model uses a BFM wrapper on top of ZeBu memories to model the EEPROM memory device implemented in ZeBu. Since, it relies on ZeBu memories, the ZeBu EEPROM memory model provides the usual ZeBu hardware debugging features such as:

- ❖ Runtime memory load and dump.
- ❖ Memory cell Read and Write.

1.1 ZEEPROM Library

The ZeBu EEPROM memory models are available for 1Kb, 2Kb, 256Kb, 512 Kb, and 1024 Kb memory capacity components on any ZeBu systems.

Memory Capacity	EEPROM Memory Device	ZeBu EEPROM Model
1Kb	Atmel AT93C46 Atmel AT24C01C	zEEPROM_AT93C_1Kb zEEPROM_AT24C_1Kb
2Kb	Atmel AT24C02C	zEEPROM_AT24C_2Kb
256Kb	Atmel AT24C256	zEEPROM_AT24C_256Kb
512Kb	Atmel AT24C512	zEEPROM_AT24C_512Kb
1024Kb	Atmel AT24C1024	zEEPROM_AT24C_1024Kb

1.2 Features

The ZeBu EEPROM memory models have the following features:

- ❖ Supports byte write, page write and acknowledge polling operations.
- ❖ Supports current address read, random read and sequential read operations.
- ❖ Supports erase operations.

2 Installation

2.1 Installing the ZeBu EEPROM Package

install the ZeBu EEPROM package, proceed as follows:

1. Make sure the ZEBU_IP_ROOT environment variable in your shell points to your IP installation directory. Set it accordingly otherwise.
2. Make sure you have WRITE permissions on the IP directory and on the current directory.
3. Launch the installation script as follows:
   - `./ZEEPROM.<VERSION>.sh install`

   Note: If the `ZEBU_IP_ROOT` environment variable is not set, you may launch the installation script as follows:

   - `./ZEEPROM.<VERSION>.sh` install `<ZEBU_IP_ROOT>`
   - The installation script executes the following operations:
     - It creates the `$ZEBU_IP_ROOT/HW_IP/ZEEPROM.<VERSION>` directory.
     - It extracts all files of the package into this directory.
     - It creates a `$ZEBU_IP_ROOT/HW_IP/ZEEPROM` symbolic link to target the `$ZEBU_IP_ROOT/HW_IP/ZEEPROM.<VERSION>` directory.
     - It unzips the `.gz` simulation library files of the package to the `$ZEBU_IP_ROOT/ HW_IP/lib` directory, and symbolic links are created to target the most recent libraries.

2.2 Package Description

After the ZeBu EEPROM memory models have been correctly installed, they come with the following elements:

`libIpsimu` directory	Compressed .so libraries (`.gz`) of the memory models for simulation.
`component` directory	Component blackbox for design synthesis
`simu` directory.	Verilog RTL simulation model
`doc` directory	This manual

2.3 Uninstalling the ZeBu EEPROM Package

To uninstall the ZeBu EEPROM package, launch the automatic uninstallation script as follows:

```
source ${ZEBU_IP_ROOT}/HW_IP/ZEEPROM.<VERSION>/uninstall
```

The uninstallation script executes the following operations:

- ❖ It removes the `$ZEBU_IP_ROOT/HW_IP/ZEEPROM.<VERSION>` directory.
- ❖ It removes the `$ZEBU_IP_ROOT/HW_IP/ZEEPROM` symbolic link for this package version.

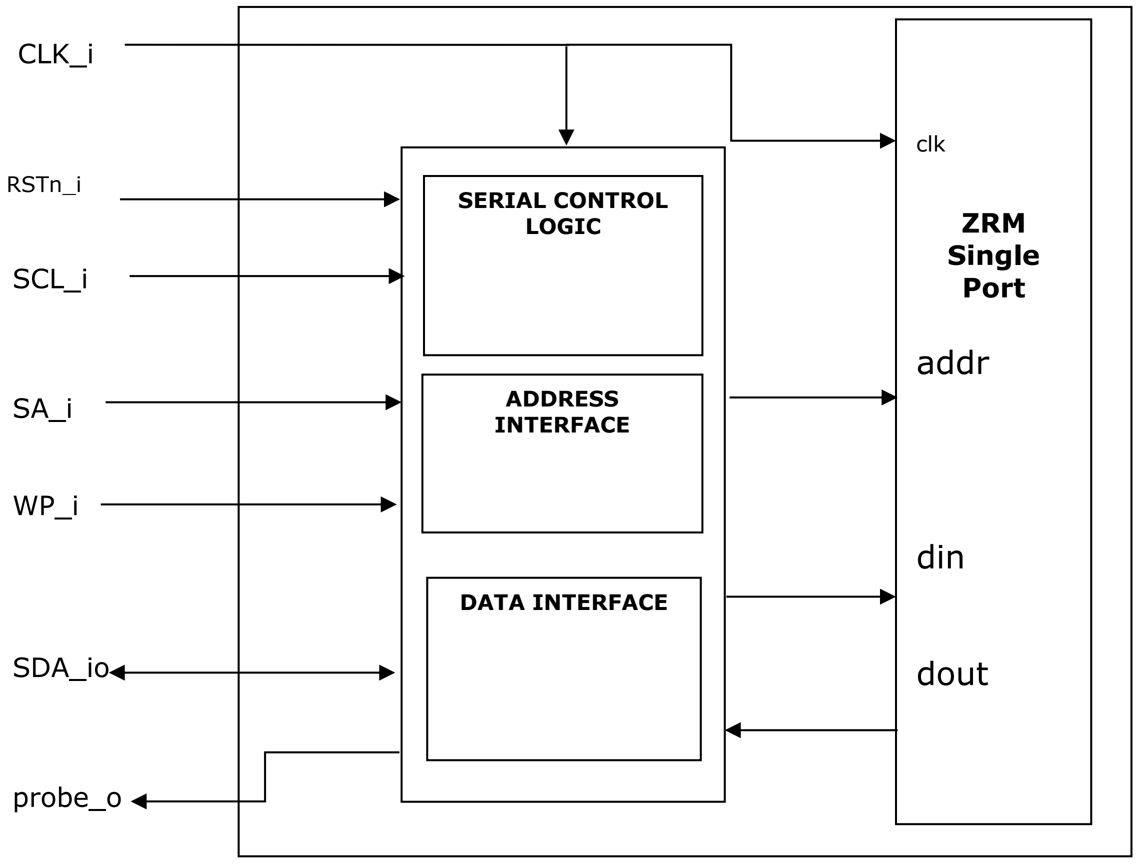
3 Memory Model Description

Table 3-1: ZeBu EEPROM Memory Models

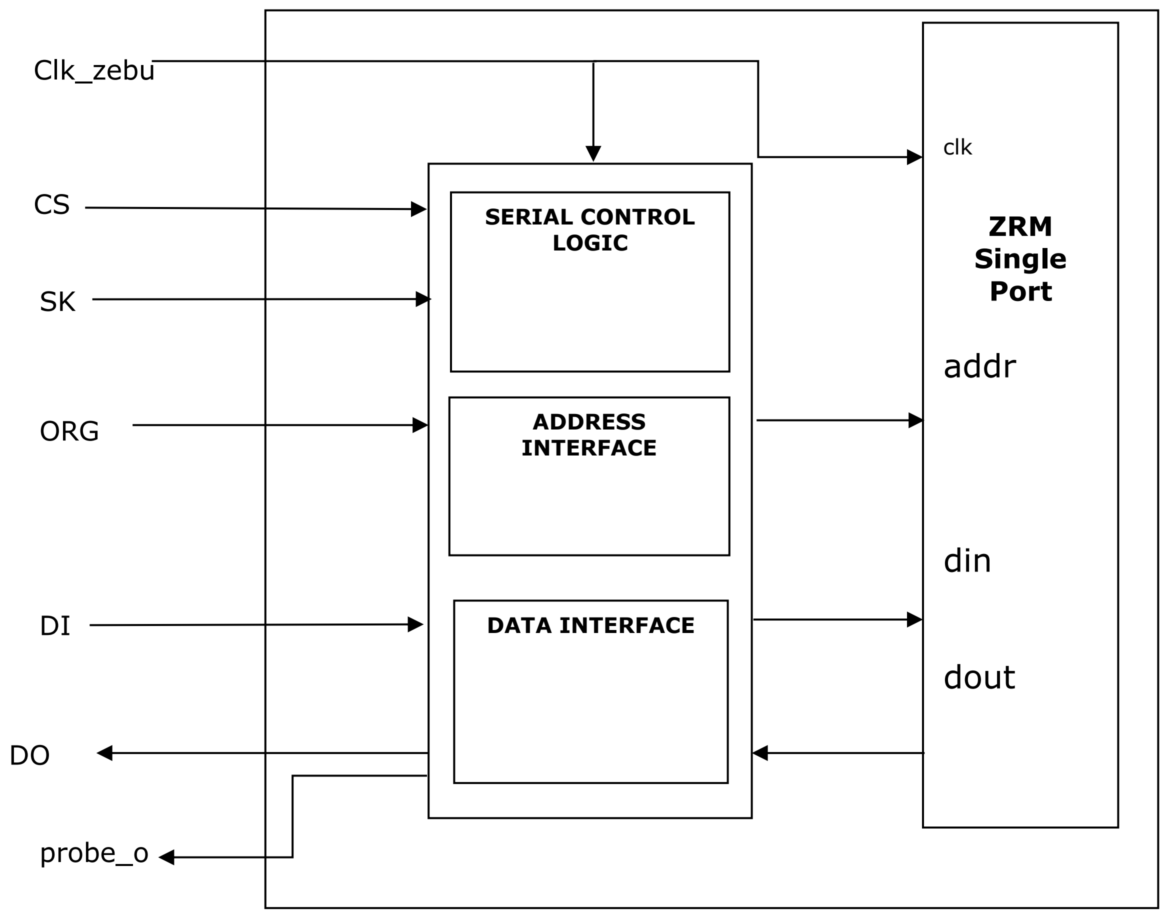
ZeBu EEPROM Model	Memory Capacity	Organization	EEPROM Device P/N
`zEEPROM_AT93C_1Kb`	1Kb	128-words x 8-bits OR, 64-words x 16-bits	Atmel AT93C46
`zEEPROM_AT24C_1Kb`	1Kb	128-words x 8-bits	Atmel AT24C01C
`zEEPROM_AT24C_2Kb`	2Kb	256-words x 8-bits	Atmel AT24C02C
`zEEPROM_AT24C_256Kb`	256Kb	32768-words x 8-bits	Atmel AT24C256
`zEEPROM_AT24C_512Kb`	512Kb	65536-words x 8-bits	Atmel AT24C512
`zEEPROM_AT24C_1024Kb`	1024Kb	131072-words x 8-bits	Atmel AT24C1024

3.1 Functional Block Diagram

The ZEEPROM synthesizable ZeBu memory models for Part numbers **Atmel AT24C01C, Atmel AT24C02C, Atmel AT24C256, Atmel AT24C512,** and **Atmel AT24C1024** are architected as follows:



The ZEEPROM synthesizable ZeBu memory models for Part number **Atmel AT93C46** is architected as follows:



3.2 Interface Description

Signal Name	Type	Description
`CLK_i`	Input	Additional EEPROM memory clock input. It is ZeBu EEPROM specific signal and doesn't exist in the standard interface. *See Note
`RSTn_i`	Input	Hardware reset pin input. It is ZeBu EEPROM specific signal and doesn't exist in the standard interface.
`SA_i`	Input	Address inputs.
`SCL_i`	Input	Serial clock input.
`SDA_io`	Inout	Serial data input/output.
`WP_i`	Input	Write protect input.

Signal Name	Type	Description
probe_o	Output	Debug probe. It is ZEEPROM specific signal and doesn't exist in the standard interface. *See Chapter-6 for more details.

Note: The frequency of CLK_i should be at least 2 times of that of SCL_i.

The table below describes the ZeBu EEPROM interface for Part number – Atmel AT93C46

Table 3-2: ZeBu EEPROM Memory Model Interface for Part number – Atmel AT93C46

Signal Name	Type	Description
CS	Input	Chip select input.
Clk_zebu	Input	Additional EEPROM memory clock input. It is ZEEPROM specific signal and doesn't exist in the standard interface. See Note *
DI	Input	Serial data input.
DO	Output	Serial data output.
ORG	Input	Internal organization input.
SK	Input	Serial data clock
probe_o	Output	Debug probe. It is ZEEPROM specific signal and doesn't exist in the standard interface. *See Chapter-6 for more details.

Note: The frequency of Clk_zebu should be at least 2 times of that of SK.

4 Integrating Memory Model With DUT

This section describes how to integrate the memory model with the DUT using the encrypted Verilog model.

4.1 Syntax Variables

Please note the following for syntax variables throughout this chapter:

- _`<model_name>` = Any of the memory models listed in ZEEPROM Library.
- _ `<hw_ip_path>` stands for `$ZEBU_IP_ROOT/HW_IP`
- _ `<ip_path>` stands for `<hw_ip_path>/ZEEPROM.<version>`
- _ `<model_path>` stands for `<ip_path>/<model_name>/<size>`

4.2 Synthesizing and Compiling for ZeBu

Use the following commands to add the RTL model to an existing zCui compilation UTF project.

```
//In VCS compilation command, add following option and files://

+define+ZEBU_SYNTH $ZEBU_IP_ROOT/HW_IP/vlog/vcs/<model_name>.vp
```

Alternatively, you can use `-y` VCS option as follows:

```
+define+ZEBU_SYNTH +libext+.v+.vp+.sv -y $ZEBU_IP_ROOT/HW_IP/vlog/vcs
```

4.3 Simulation

The ZEEPROM package is supplied with:

- A `libIpSimu_<32/64>.so` library in the `<hw_ip_path>/lib` directory for simulation purposes.
- An RTL-level Verilog simulation model in an encrypted format with encryption depending on the target HDL simulator. It is available in the `<model_path>/simu/rtl/vcs` directory.

This section explains Simulation with RTL-Level Model and Result of ZeBu IP License Checking.

4.4 Simulation with RTL-Level Model

Specify the following information for the memory model at the end of the script, as follows:

```
vcs <my_options> <file_list> -sverilog
$ZEBU_IP_ROOT/HW_IP/vlog/vcs/<model_name>.vp

$ZEBU_IP_ROOT/HW_IP/lib/libIpSimu_<32/64>.so
```

If you need to use several different ZEEPROM models, you should compile each one separately as VCS does not support multiple compilations of the same sub-modules in the same command line.

Example

```
vlogan -sverilog <model_path>/simu/rtl/vcs/<model_name>.vp

…

…

vcs <my_options> <my_top_level_name>
<hw_ip_path>/lib/libIpSimu_<32/64>.so
```

Else, the following error message is displayed:

```
Error-[MPD] Module previously declared
```

5 Accessing the ZEEPROM Memory (zrm Load & Dump)

To access the content of the memory at runtime, you can use the standard way to read from or write to ZeBu memories with its appropriate hierarchical path. This path to the memory is like:

```
<path_to_ZEEPROM_mem> = <path_to_ZEEPROM_inst>.EEPROM_mem.mem
```

5.1 Example

You want to initialize memory in a design using an EEPROM model with the '`memory.init`' content file. If the path to the ZEEPROM instance is '`top_dut.my_ZEEPROM_inst`', then the full path to the ZRM memory would be:

```
<path_to_ZEEPROM_mem> = top_dut.my_ZEEPROM_inst.EEPROM_mem.mem
```

Therefore, add the following line in a `designFeatures` file (default mode for synthesizable testbenches):

```
$memoryInitDB = "init_mem"
```

Where, `init_mem` is a file consisting of a collection of lines, with each line listing a memory and the corresponding content file name. In this example, the content of the `init_mem` file is:

```
<path_to_ZEEPROM_mem> memory.init
```

Note: The '`designFeatures`' file is a way to initialize the ZRM memories in an emulation-based environment similar to '`$readmemh`' Verilog system task used in simulation.

5.2 Initializing Memories for Simulation

Verilog system tasks, `$readmemh` and `$writememh`, are used to initialize and dump memory data respectively.

- `$readmemh` system task reads address and corresponding data (in hexadecimal format) from a file and loads it into the memory.

```
$readmemh("load_data.hex", "<path_to_ZEEPROM_inst>.EEPROM_mem.mem");
```

- `$writememh` system task dumps address and corresponding data (in hexadecimal format) to a file from the memory.

```
$writememh("dump_data.hex","<path_to_ZEEPROM_inst>.EEPROM_mem.mem");
```

6 Debug Information

For debugging purposes, a list of signals is available at runtime to trace the internal behavior of ZEEPROM models.

❖ Probe signals for Part numbers: Atmel AT24C01C, Atmel AT24C02C, Atmel AT24C256, Atmel AT24C512, Atmel AT24C1024

Signal	Description
`probe_o[26:19]`	Reserved.
`probe_o[18:17]`	Memory Access FSM states: **2'b00:** `Idle` **2'b01:** `Read data` **2'b10:** `Write address` **2'b11:** `Write data`
`probe_o[16]`	ZRM write enable.
`probe_o[15]`	ZRM read enable.
`probe_o[14]`	Reserved.
`probe_o[13:10]`	**4'b0000** to **4'b1111**: Clock (`CLK_i`) posedge counter.
`probe_o[9:6]`	**4'b0000** to **4'b1111**: Clock (`CLK_i`) negedge counter.
`probe_o[5]`	SDA output enable, when high, the ZEEPROM is driving the bidirectional SDA
`probe_o[4:2]`	Command FSM states: **3'b000:** `Idle` **3'b001:** `Header` **3'b010:** `Header_ACK` **3'b011:** `Receive` **3'b100:** `Receive_ACK` **3'b101:** `Transmit`

Signal	Description
	3'b110: Transmit_ACK
probe_o[1]	**1'b1**: Stop condition (A low-to-high transition of SDA with SCL high).
probe_o[0]	**1'b1**: Start condition (A high-to-low transition of SDA with SCL high).

❖ Probe signals for Part number: **Atmel AT93C46**

Signal	Description
probe_o[14]	The ZEEPROM is driving the data output on DO pin
probe_o[13]	Erase Write enable (EWEN).
probe_o[12:4]	Input address during address state (4'b0001).
probe_o[3:0]	FSM states: **4'b0000:** Idle **4'b0001:** Address **4'b0010:** Din **4'b0011:** Dout **4'b0100:** Write **4'b0101:** Busy

7 Examples

The ZEEPROM package provides a VCS simulation example, which describes how to use the ZEEPROM Memory Models.

7.1 Prerequisite

Before running the tutorial/example, ensure that the following environment variables are correctly set:

- `ZEBU_ROOT`: Specify a valid ZeBu installation.
- `ZEBU_IP_ROOT`: Specify the path to the package installation directory.

7.2 Running the Tutorial-

Synopsys VCS simulator performs the HDL simulation for the RTL level model. Perform the following steps to compile and run the example:

1. Go to the `example/<model>` directory.
2. Launch the compilation flow with the Makefile:
   ```
   make compil
   ```
   Note: `<model>` is AT24C or AT93C.
3. Launch the emulation flow with the Makefile:
   ```
   make run
   ```
4. Optionally, clean the compilation and run directory (the working directory automatically created at compilation and run is removed):
   ```
   make clean
   ```