



antmicro

Rowhammer tester

Antmicro

2024-01-25

CONTENTS

1 General	1
1.1 Architecture	1
1.2 Installing dependencies	2
1.3 Packaging the bitstream	3
1.4 Local documentation build	3
1.5 Tests	3
2 User guide	4
2.1 Network USB adapter setup	4
2.2 Controlling the board	6
2.3 Hammering	6
2.4 Utilities	11
2.5 Simulation	16
3 Visualization	17
3.1 Plot bitflips - logs2plot.py	17
3.2 Plot per DQ pad - logs2dq.py	19
3.3 Use F4PGA Visualizer - logs2vis.py	19
4 Playbook	21
4.1 Payload	21
4.2 Row mapping	21
4.3 Row Generator class	22
4.4 Payload generator class	22
4.5 Configurations	26
5 DRAM modules	28
5.1 Adding new modules	28
5.2 SPD EEPROM	29
6 Arty-A7 board	30
6.1 Board configuration	30
7 ZCU104 board	32
7.1 Board configuration	32
7.2 Preparing SD card	32
7.3 Loading bitstream	33
7.4 Verifying if bitstream is loaded	34
7.5 ZCU104 microUSB	35
7.6 Network setup	35

7.7	SSH access	36
7.8	Controlling the board	36
7.9	ZCU104 SD card image	37
8	LPDDR4 Test Board	41
8.1	Board configuration	42
9	Data Center DRAM Tester	43
9.1	Board configuration	44
10	DDR5 Tester	45
10.1	Rowhammer Tester Target Configuration	45
10.2	Linux Target Configuration	46
10.3	Simulation	51
11	DDR5 Test Board	52
11.1	Board configuration	53
12	Documentation for Row Hammer Tester Arty-A7	54
12.1	Modules	54
12.2	Register Groups	54
13	Documentation for Row Hammer Tester ZCU104	91
13.1	Modules	91
13.2	Register Groups	91
14	Documentation for Row Hammer Tester Data Center DRAM Tester	138
14.1	Modules	138
14.2	Register Groups	138
15	Documentation for Row Hammer Tester LPDDR4 Test Board	184
15.1	Modules	184
15.2	Register Groups	184
16	Documentation for Row Hammer Tester DDR5 Test Board	229
16.1	Modules	229
16.2	Register Groups	229
17	Documentation for Row Hammer Tester DDR5 Tester	280
17.1	Modules	280
17.2	Register Groups	281

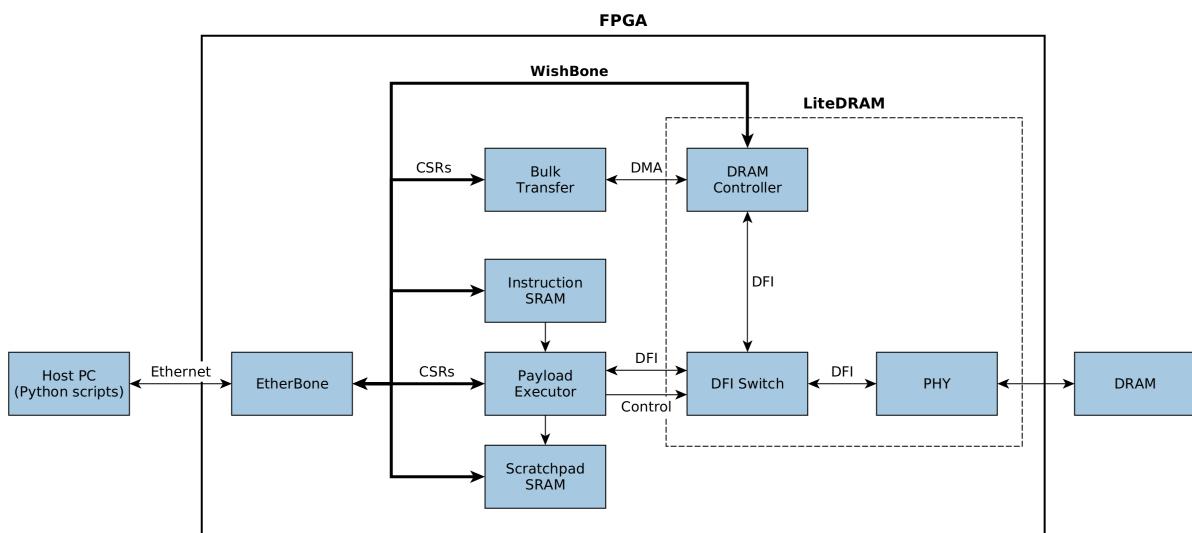
CHAPTER ONE

GENERAL

The aim of this project is to provide a platform for testing DRAM vulnerability to rowhammer attacks.

1.1 Architecture

The setup consists of FPGA gateware and application side software. The following diagram illustrates the general system architecture.



The DRAM is connected to LiteDRAM, which provides swappable PHYs and a DRAM controller implementation.

In the default bulk transfer mode the LiteDRAM controller is connected to PHY and ensures correct DRAM traffic. Bulk transfers can be controlled using dedicated Control & Status Registers (CSRs) and use LiteDRAM DMA to ensure fast operation.

The Payload Executor allows executing a user-provided sequence of commands. It temporarily disconnects the DRAM controller from PHY, executes the instructions stored in the SRAM memory, translating them into DFI commands and finally reconnects the DRAM controller.

The application side consists of a set of Python scripts communicating with the FPGA using the LiteX EtherBone bridge.

1.2 Installing dependencies

Make sure you have Python 3 installed with the `venv` module, and the dependencies required to build `verilator`, `openFPGALoader` and `OpenOCD`. To install the dependencies on Ubuntu 18.04 LTS, run:

```
apt install git build-essential autoconf cmake flex bison libftdi-dev libjson-c-dev libevent-dev libtinfo-dev uml-utilities python3 python3-venv python3-wheel-proto-compiler libcairo2 libftdi1-2 libftdi1-dev libhidapi-hidraw0 libhidapi-dev libudev-dev pkg-config tree zlib1g-dev zip unzip help2man curl ethtool
```

Note: On some Debian-based systems there's a problem with a broken dependency:

```
libc6-dev : Breaks: libgcc-9-dev (< 9.3.0-5~) but 9.2.1-19 is to be installed
```

`gcc-9-base` package installation solves the problem.

On Ubuntu 22.04 LTS the following dependencies may also be required:

```
apt install libtool libusb-1.0-0-dev pkg-config
```

1.2.1 Rowhammer tester

Now clone the `rowhammer-tester` repository and install the rest of the required dependencies:

```
git clone --recursive https://github.com/antmicro/rowhammer-tester.git  
cd rowhammer-tester  
make deps
```

The last command will download and build all the dependencies (including a RISC-V GCC toolchain) and will set up a `Python virtual environment` under the `./venv` directory with all the required packages installed.

The virtual environment allows you to use Python without installing the packages system-wide. To enter the environment, you have to run `source venv/bin/activate` in each new shell. You can also use the provided `make env` target, which will start a new Bash shell with the `virtualenv` already sourced. You can install packages inside the virtual environment by entering the environment and then using `pip`.

To build the bitstream you will also need to have Vivado (version 2020.2 or newer) installed and the `vivado` command available in your PATH. To configure Vivado in the current shell, you need to source `/PATH/T0/Vivado/VERSION/settings64.sh`. This can be put in your `.bashrc` or other shell init script.

To make the process automatic without hard-coding these things in shell init script, tools like `direnv` can be used. A sample `.envrc` file would then look like this:

```
source venv/bin/activate  
source /PATH/T0/Vivado/VERSION/settings64.sh
```

All other commands assume that you run Python from the virtual environment with vivado in your PATH.

1.3 Packaging the bitstream

If you want to save the bitstream and use it later or share it with someone, there is an utility target `make pack`. It packs files necessary to load the bitstream and run rowhammer scripts on it. Those files are:

- `build/$TARGET/gateware/$TOP.bit`
- `build/$TARGET/csr.csv`
- `build/$TARGET/defs.csv`
- `build/$TARGET/sdram_init.py`
- `build/$TARGET/litedram_settings.json`

After running `make pack`, you should have a zip file named like `$TARGET-$BRANCH-$COMMIT.zip`.

Next time you want to use a bitstream packaged in such way, all you need to do is to run `unzip your-bitstream-file.zip` and you are all set.

1.4 Local documentation build

The gateware part of the documentation is auto-generated from source files. Other files are static and are located in the `doc/` directory. To build the documentation, enter:

```
source venv/bin/activate
pip install -r requirements.txt
python -m sphinx -b html doc/build/documentation/html
```

The documentation will be located in `build/documentation/index.html`.

Note: For easier development one can use `sphinx-autobuild` using command `sphinx-autobuild -b html doc/build/documentation/html --re-ignore 'doc/build/.*'`. The documentation can be then viewed in a browser at `http://127.0.0.1:8000`.

1.5 Tests

To run project tests use:

```
make test
```

CHAPTER
TWO

USER GUIDE

This tool can be run on real hardware (FPGAs) or in a simulation mode. As the rowhammer attack exploits physical properties of cells in DRAM (draining charges), no bit flips can be observed in simulation mode. However, the simulation mode is useful to test command sequences during the development.

The Makefile can be configured using environmental variables to modify the network configuration used and to select the target. Currently, 4 boards are supported, each targeting different memory type:

Board	Memory type	TARGET
Arty A7	DDR3	arty
ZCU104	DDR4 (SO-DIMM)	zcu104
DDR Datacenter DRAM Tester	DDR4 (RDIMM)	ddr4_datacenter_test_board
LPDDR4 Test Board	LPDDR4 (SO-DIMM)	lpddr4_test_board
DDR5 Tester	DDR5 (RDIMM)	ddr5_tester
DDR5 Test Board	DDR5 (SO-DIMM)	ddr5_test_board

Note: Although you choose a target board for the simulation, it doesn't require having a physical board. Simulation is done entirely on your computer.

For board-specific instructions refer to [Arty A7](#), [ZCU104](#), [DDR4 Datacenter DRAM Tester](#), [LPDDR4 Test Board](#), [DDR5 Tester](#) and [DDR5 Test Board](#) chapters. The rest of this chapter describes operations that are common for all supported boards.

2.1 Network USB adapter setup

In order to control the Rowhammer platform an Ethernet connection is necessary. In case you want to use an USB Ethernet adapter for this purpose read the instructions below.

1. Make sure you use a 1GbE USB network adapter
2. Figure out the MAC address for the USB network adapter:
 - Run `sudo lshw -class network -short` to get the list of all network interfaces
 - Check which of the devices uses the `r8152` driver by running `sudo ethtool -i <device>`

- Display the link information for the device running `sudo ip link show <device>` and look for the mac address next to the link/ether field
3. Configure the USB network adapter to appear as network device `fpga0` using `systemd`
- Create `/etc/systemd/network/10-fpga0.link` with the following contents:

```
[Match]
# Set this to the MAC address of the USB network adapter
MACAddress=XX:XX:XX:XX:XX

[Link]
Name=fpga0
```

4. Configure the `fpga0` network device with a static IP address, always up (even when disconnected) and ignored by the network manager.

- Run the following command, assuming your system uses `NetworkManager`

```
nmcli con add type ethernet con-name 'Rowhammer Tester' ifname fpga0
  ↳ ipv4.method manual ipv4.addresses 192.168.100.100/24
```

- Alternatively, if your system supports legacy interfaces configuration file:

1. Make sure your `/etc/network/interfaces` file has the following line:

```
source /etc/network/interfaces.d/*
```

2. Create `/etc/network/interfaces.d/fpga0` with the following contents:

```
auto fpga0
allow-hotplug fpga0
iface fpga0 inet static
  address 192.168.100.100/24
```

3. Check that `nmcli device` says the state is connected (externally) otherwise run `sudo systemctl restart NetworkManager`

- Run `ifup fpga0`

5. Run `sudo udevadm control --reload` and then unplug the USB ethernet device and plug it back in
6. Check you have an `fpga0` interface and it has the correct IP address by running `networkctl status`

Note: In case you see `libusb_open()` failed with `LIBUSB_ERROR_ACCESS` when trying to use the rowhammer tester scripts with the USB ethernet adapter then it means that you have a permissions issue and need to allow access to the FTDI USB to serial port chip. Check the group listed for the `tty`'s when running `ls -l /dev/ttyUSB*` and add the current user to this group by running `sudo adduser <username> <group>`.

2.2 Controlling the board

Board control is the same for both simulation and hardware runs. In order to communicate with the board via EtherBone, the `litex_server` needs to be started with the following command:

```
export IP_ADDRESS=192.168.100.50 # optional, should match the one used during  
↪build  
make srv
```

Warning: If you want to run the simulation and the rowhammer scripts on a physical board at the same time, you have to change the `IP_ADDRESS` variable, otherwise the simulation can conflict with the communication with your board.

The build files (CSRs address list) must be up to date. It can be re-generated with `make` without arguments.

Then, in another terminal, you can use the Python scripts provided. *Remember to enter the Python virtual environment before running the scripts!* Also, the `TARGET` variable should be set to load configuration for the given target. For example, to use the `leds.py` script, run the following:

```
source ./venv/bin/activate  
export TARGET=arty # (or zcu104) required to load target configuration  
cd rowhammer_tester/scripts/  
python leds.py # stop with Ctrl-C
```

2.3 Hammering

Rowhammer attacks can be run against a DRAM module. It can be then used for measuring cell retention. For the complete list of scripts' modifiers, see `--help`.

There are two versions of a rowhammer script:

- `rowhammer.py` - this one uses regular memory access via EtherBone to fill/check the memory (slower)
- `hw_rowhammer.py` - BIST blocks will be used to fill/check the memory (much faster, but with some limitations regarding fill pattern)

BIST blocks are faster and are the intended way of running Rowhammer tester.

Hammering of a row is done by reading it. There are two ways to specify a number of reads:

- `--read_count N` - one pass of N reads
- `--read_count_range K M N` - multiple passes of reads, as generated by `range(K, M, N)`

Regardless of which one is used, the number of reads in one pass is divided equally between hammered rows. If a user specifies `--read_count 1000`, then each row will be hammered 500 times.

Normally hammering is being performed via DMA, but there is also an alternative way with `--payload-executor`. It bypasses the DMA and directly talks with the PHY. That allows the user to issue specific activation, refresh and precharge commands.

2.3.1 Attack modes

Different attack and row selection modes can be used, but only one of them can be specified at the same time.

- `--hammer-only`

Only hammers rows, without doing any error checks or reports. When run with `rowhammer.py` the attack is limited to one row pair. `hw_rowhammer.py` can attack up to 32 rows. With `--payload-executor` enabled the row limit is dictated by the payload memory size.

For example following command will hammer rows 4 and 6 1000 times total (so 500 times each):

```
(venv) $ python hw_rowhammer.py --hammer-only 4 6 --read_count 1000
```

- `--all-rows`

Row pairs generated from `range(start-row, nrows - row-pair-distance, row-jump)` expression will be hammered.

Generated pairs are of form `(i, i + row-pair-distance)`. Default values for used arguments are:

argument	default
<code>--start-row</code>	0
<code>--row-jump</code>	1
<code>--row-pair-distance</code>	2

So you can run following command to hammer rows `(0, 2), (1, 3), (2, 4)`:

```
(venv) $ python hw_rowhammer.py --all-rows --nrows 5
```

And in case of:

```
(venv) $ python hw_rowhammer.py --all-rows --start-row 10 --nrows 16 --row-
jump 2 --row-distance 3
```

hammered pairs would be: `(10, 13), (12, 15)`.

In a special case, where `--row-pair-distance` is 0, you can check how hammering a single row affects other rows. Normally activations and deactivations are achieved with row reads using the DMA, but in this case it is not possible. Because the same row is being read all the time, no deactivation command would be sent by the DMA. In this case, `--payload-executor` is required as it bypasses the DMA and sends deactivation commands on its own.

```
(venv) $ python hw_rowhammer.py --all-rows --nrows 5 --row-pair-distance 0 --
    ↪payload-executor
```

- `--row-pairs sequential`

Hammers pairs of (start-row, start-row + n), where n is from 0 to nrows.

```
(venv) $ python hw_rowhammer.py --row-pairs sequential --start-row 4 --nrows_
    ↪10
```

Command above, would hammer following set of row pairs:

```
(4, 4 + 0)
(4, 4 + 1)
...
(4, 4 + 9)
(4, 4 + 10)
```

- `--row-pairs const`

Two rows specified with the `const-rows-pair` parameter will be hammered:

```
(venv) $ python hw_rowhammer.py --row-pairs const --const-rows-pair 4 6
```

- `--row-pairs random`

nrows pairs of random rows will be hammered. Row numbers will be between start-row and start-row + nrows.

```
(venv) $ python hw_rowhammer.py --row-pairs random --start-row 4 --nrows 10
```

- `--no-attack-time <time>`

Instead of performing a rowhammer attack, the script will load the RAM with selected pattern and sleep for time nanoseconds. After this time, it will check for any bitflips that could have happened. This option does not imply `--no-refresh`!

```
(venv) $ python hw_rowhammer.py --no-attack-time 100e9 --no-refresh
```

2.3.2 Patterns

User can choose a pattern that memory will be initially filled with:

- `all_0` - all bits set to 0
- `all_1` - all bits set to 1
- `01_in_row` - alternating 0's and 1's in a row (0xaaaaaaaa in hex)
- `01_per_row` - all 0's in odd-numbered rows, all 1's in even rows
- `rand_per_row` - random values for all rows

2.3.3 Example output

```
(venv) $ python hw_rowhammer.py --nrows 512 --read_count 10e6 --pattern 01_in_row_
↪--row-pairs const --const-rows-pair 54 133 --no-refresh
Preparing ...
WARNING: only single word patterns supported, using: 0aaaaaaaaa
Filling memory with data ...
Progress: [=====] 16777216 / 16777216
Verifying written memory ...
Progress: [=====] 16777216 / 16777216 (Errors: ↪0)
OK
Disabling refresh ...
Running Rowhammer attacks ...
read_count: 10000000
Iter 0 / 1 Rows = (54, 133), Count = 10.00M / 10.00M
Reenabling refresh ...
Verifying attacked memory ...
Progress: [=====] 16777216 / 16777216 (Errors: ↪30)
Bit-flips for row 53: 5
Bit-flips for row 55: 11
Bit-flips for row 132: 12
Bit-flips for row 134: 3
```

2.3.4 Row selection examples

Warning: Attacks are performed on a single bank. By default it is bank 0. To change the bank that is being attacked use the `--bank` flag.

- Select row pairs from row 3 (`--start-row`) to row 59 (`--nrows`) where the next pair is 5 rows away (`--row-jump`) from the previous one:

```
(venv) $ python hw_rowhammer.py --pattern 01_in_row --all-rows --start-row 3_
↪--nrows 60 --row-jump 5 --no-refresh --read_count 10e4
```

- Select row pairs from row 3 to to row 59 without a distance between subsequent pairs (`no --row-jump`), which means that rows pairs are incremented by 1:

```
(venv) $ python hw_rowhammer.py --pattern 01_in_row --all-rows --start-row 3_
↪--nrows 60 --no-refresh --read_count 10e4
```

- Select all row pairs (from 0 to nrows - 1):

```
(venv) $ python hw_rowhammer.py --pattern 01_in_row --all-rows --nrows 512 --
↪no-refresh --read_count 10e4
```

- Select all row pairs (from 0 to nrows - 1) and save the error summary in JSON format to the test directory:

```
(venv) $ python hw_rowhammer.py --pattern 01_in_row --all-rows --nrows 512 --
    ↵no-refresh --read_count 10e4 --log-dir ./test
```

- Select only one row (42 in this case) and save the error summary in JSON format to the test directory:

```
(venv) $ python hw_rowhammer.py --pattern all_1 --row-pairs const --const-
    ↵rows-pair 42 42 --no-refresh --read_count 10e4 --log-dir ./test
```

- Select all rows (from 0 to nrows - 1) and hammer them one by one 1M times each.

```
(venv) $ python hw_rowhammer.py --all-rows --nrows 100 --row-pair-distance 0 --
    ↵--payload-executor --no-refresh --read_count 1e6
```

Note: Since for a single ended attack row activation needs to be triggered the --payload-executor switch is required. The size of the payload memory is set by default to 1024 bytes and can be changed using the --payload-size switch.

2.3.5 Cell retention measurement examples

- Select all row pairs (from 0 to nrows - 1) and perform a set of tests for different read count values, starting from 10e4 and ending at 10e5 with a step of 20e4 (--read_count_range [start stop step]):

```
(venv) $ python hw_rowhammer.py --pattern 01_in_row --all-rows --nrows 512 --
    ↵no-refresh --read_count_range 10e4 10e5 20e4
```

- Perform set of tests for different read count values in a given range for one row pair (50, 100):

```
(venv) $ python hw_rowhammer.py --pattern 01_in_row --row-pairs const --
    ↵const-rows-pair 50 100 --no-refresh --read_count_range 10e4 10e5 20e4
```

- Perform set of tests for different read count values in a given range for one row pair (50, 100) and stop the test execution as soon as a bitflip is found:

```
(venv) $ python hw_rowhammer.py --pattern 01_in_row --row-pairs const --
    ↵const-rows-pair 50 100 --no-refresh --read_count_range 10e4 10e5 20e4 --
    ↵exit-on-bit-flip
```

- Perform set of tests for different read count values in a given range for one row pair (50, 100) and save the error summary in JSON format to the test directory:

```
(venv) $ python hw_rowhammer.py --pattern 01_in_row --row-pairs const --
    ↵const-rows-pair 50 100 --no-refresh --read_count_range 10e4 10e5 20e4 --
    ↵log-dir ./test
```

- Perform set of tests for different read count values in a given range for a sequence of attacks for different pairs, where the first row of a pair is 40 and the second one is a row of a number from range (40, nrows - 1):

```
(venv) $ python hw_rowhammer.py --pattern 01_in_row --row-pairs sequential --
→start-row 40 --nrows 512 --no-refresh --read_count_range 10e4 10e5 20e4
```

2.4 Utilities

Some of the scripts are simple and do not take command line arguments, others will provide help via `<script_name>.py --help` or `<script_name>.py -h`.

Few of the scripts accept a `--srv` option. With this option enabled, a program will start its own instance of `liteX_server` (the user doesn't need to run `make srv` to *control the board*)

2.4.1 Run LEDs demo - `leds.py`

Displays a simple “bouncing” animation using the LEDs on Arty-A7 board, with the light moving from side to side.

`-t TIME_MS` or `--time-ms TIME_MS` option can be used to adjust LED switching interval.

2.4.2 Check version - `version.py`

Prints the data stored in the LiteX identification memory:

- hardware platform identifier
- source code git hash
- build date

Example output:

```
(venv) python version.py
Rowhammer tester SoC on xc7k160tffg676-1, git:_
→e7854fdd16d5f958e616bbb4976a97962ee9197d 2022-07-24 15:46:52
```

2.4.3 Check CSRs - `dump_regs.py`

Dumps values of all CSRs. Example output of `dump_regs.py`:

```
0x82000000: 0x00000000 ctrl_reset
0x82000004: 0x12345678 ctrl_scratch
0x82000008: 0x00000000 ctrl_bus_errors
0x82002000: 0x00000000 uart_rxtx
0x82002004: 0x00000001 uart_txfull
0x82002008: 0x00000001 uart_rxempty
0x8200200c: 0x00000003 uart_ev_status
0x82002010: 0x00000000 uart_ev_pending
...
```

Note: Note that `ctrl_scratch` value is 0x12345678. This is the reset value of this register. If you are getting a different, this may indicate a problem.

2.4.4 Initialize memory - `mem.py`

Before the DRAM memory can be used, the initialization and leveling must be performed. The `mem.py` script serves this purpose.

Expected output:

```
(venv) $ python mem.py
(LiteX output)
=====
Initialization =====
Initializing SDRAM @0x40000000...
Switching SDRAM to software control.
Read leveling:
m0, b0: |11111111111100000000000000000000| delays: 06+-06
m0, b1: |00000000000000011111111111111000| delays: 21+-08
m0, b2: |00000000000000000000000000000011| delays: 31+-01
m0, b3: |00000000000000000000000000000000| delays: -
m0, b4: |00000000000000000000000000000000| delays: -
m0, b5: |00000000000000000000000000000000| delays: -
m0, b6: |00000000000000000000000000000000| delays: -
m0, b7: |00000000000000000000000000000000| delays: -
best: m0, b01 delays: 21+-07
m1, b0: |11111111111100000000000000000000| delays: 07+-07
m1, b1: |00000000000000011111111111111000| delays: 22+-08
m1, b2: |00000000000000000000000000000001| delays: 31+-00
m1, b3: |00000000000000000000000000000000| delays: -
m1, b4: |00000000000000000000000000000000| delays: -
m1, b5: |00000000000000000000000000000000| delays: -
m1, b6: |00000000000000000000000000000000| delays: -
m1, b7: |00000000000000000000000000000000| delays: -
best: m1, b01 delays: 22+-08
Switching SDRAM to hardware control.
Memtest at 0x40000000 (2MiB)...
  Write: 0x40000000-0x40200000 2MiB
  Read: 0x40000000-0x40200000 2MiB
Memtest OK
Memspeed at 0x40000000 (2MiB)...
  Write speed: 12MiB/s
  === Initialization succeeded. ===
Proceeding ...

Memtest (basic)
OK

Memtest (random)
OK
```

2.4.5 Enter BIOS - bios_console.py

Sometimes it may happen that memory initialization fails when running the `mem.py` script. This is most likely due to using boards that allow to swap memory modules, such as ZCU104.

Memory initialization procedure is performed by the CPU instantiated inside the FPGA fabric. The CPU runs the LiteX BIOS. In case of memory training failure it may be helpful to access the LiteX BIOS console.

If the script cannot find a serial terminal emulator program on the host system, it will fall back to `lite_term` which is shipped with LiteX. It is however advised to install `picocom/minicom` as `lite_term` has worse performance.

In the BIOS console use the `help` command to get information about other available commands. To re-run memory initialization and training type `reboot`.

Note: To close `picocom/minicom` enter `CTRL+A+X` key combination.

Example:

```
(venv) $ python bios_console.py
LiteX Crossover UART created: /dev/pts/4
Using serial backend: auto
picocom v3.1

port is      : /dev/pts/4
flowcontrol  : none
baudrate is   : 1000000
parity is     : none
databits are  : 8
stopbits are  : 1
escape is     : C-a
local echo is : no
noinit is     : no
noreset is    : no
hangup is     : no
nolock is     : no
send_cmd is   : sz -vv
receive_cmd is: rz -vv -E
imap is       :
omap is       :
emap is       : crcrlf,delbs,
logfile is    : none
initstring    : none
exit_after is : not set
exit is       : no

Type [C-a] [C-h] to see available commands
Terminal ready
ad speed: 9MiB/s
```

(continues on next page)

(continued from previous page)

```
===== Boot =====
Booting from serial...
Press Q or ESC to abort boot completely.
sL5DdSMmkekro
      Timeout
No boot medium found

===== Console =====
litex>
```

Perform memory tests from the BIOS

After entering the BIOS, you may want to perform a memory test using utilities built into the BIOS itself. There are a couple ways to do such:

- `mem_test` - performs a series of writes and reads to check if values read back are the same as those previously written. It is limited by a 32-bit address bus, so only 4 GiB of address space can be tested. You can get origin of the RAM space using `mem_list` command.
- `sram_test` - basically `mem_test`, but predefined for first 1/32 of defined RAM region size.
- `sram_hw_test` - similar to `mem_test`, but accesses the SDRAM directly using DMAs, so it is not limited to 4 GiB. It requires passing 2 arguments (`origin` and `size`) with a 3rd optional argument being `burst_length`. When using `sram_hw_test` you don't have to offset the `origin` like in the case of `mem_test`. `size` is a number of bytes to test and `burst_length` is a number of full transfer writes to the SDRAM, before reading and checking written content. The default value for `burst_length` is 1, which means that after every write, a check is performed. Generally, bigger `burst_length` means faster operation.

2.4.6 Test with BIST - `mem_bist.py`

A script written to test BIST block functionality. Two tests are available:

- `test-modules` - memory is initialized and then a series of errors is introduced (on purpose). Then BIST is used to check the content of the memory. If the number of errors detected is equal to the number of errors introduced, the test is passed.
- `test-memory` - simple test that writes a pattern in the memory, reads it, and checks if the content is correct. Both write and read operations are done via BIST.

2.4.7 Run benchmarks - benchmark.py

Benchmarks memory access performance. There are two subcommands available:

- etherbone - measure performance of the EtherBone bridge
- bist - measure performance of DMA DRAM access using the BIST modules

Example output:

```
(venv) $ python benchmark.py etherbone read 0x10000 --burst 255
Using generated target files in: build/lpddr4_test_board
Running measurement ...
Elapsed = 4.189 sec
Size    = 256.000 KiB
Speed   = 61.114 KiBps

(venv) $ python benchmark.py bist read
Using generated target files in: build/lpddr4_test_board
Filling memory before reading measurements ...
Progress: [=====] 16777216 / 16777216
Running measurement ...
Progress: [=====] 16777216 / 16777216 (Errors: ↵0)
Elapsed = 1.591 sec
Size    = 512.000 MiB
Speed   = 321.797 MiBps
```

2.4.8 Use logic analyzer - analyzer.py

This script utilizes the Litescope functionality to gather debug information about signals in the LiteX system. In-depth Litescope documentation [is here](#).

As you can see in Litescope documentation, Litescope analyzer needs to be instantiated in your design. Example design with analyzer added was provided as arty_litescope TARGET. As the name implies it can be run using Arty board. You can use rowhammer_tester/targets/arty_litescope.py as a reference for your own Litescope-enabled targets.

To build arty_litescope example and upload it to device, follow instructions below:

1. In root directory run:

```
export TARGET=arty_litescope
make build
make upload
```

analyzer.csv file will be created in the root directory.

2. We need to copy it to target's build dir before using analyzer.py.

```
cp analyzer.csv build/arty_litescope/
```

3. Then start litex-server with:

```
make srv
```

4. And execute analyzer script in a separate shell:

```
export TARGET=arty_litescope  
python rowhammer_tester/scripts/analyser.py
```

Results will be stored in dump.vcd file and can be viewed with gtkwave:

```
gtkwave dump.vcd
```

2.5 Simulation

Select TARGET, generate intermediate files & run simulation:

```
export TARGET=arty # (or zcu104)  
make sim
```

This command will generate intermediate files & simulate them with Verilator. After simulation has finished, a signals dump can be investigated using [gtkwave](#):

```
gtkwave build/$TARGET/gateware/sim.fst
```

Warning: The repository contains a wrapper script around sudo which disallows LiteX to interfere with the host network configuration. This forces the user to manually configure a TUN interface for valid communication with the simulated device.

1. Create the TUN interface:

```
tunctl -u $USER -t litex-sim
```

2. Configure the IP address of the interface:

```
ifconfig litex-sim 192.168.100.1/24 up
```

3. Optionally allow network traffic on this interface:

```
iptables -A INPUT -i litex-sim -j ACCEPT  
iptables -A OUTPUT -o litex-sim -j ACCEPT
```

Note: Typing `make ARGS="--sim"` will cause LiteX to generate only intermediate files and stop right after that.

VISUALIZATION

If you have already executed some attacks on your board you can use the results to draw a plot or visualize it in the [F4PGA Database Visualizer](#).

3.1 Plot bitflips - logs2plot.py

This script can plot graphs out of generated logs. It can generate two different types of graphs:

1. Distribution of bitflips across rows and columns. For example, you can generate some graphs by calling:

```
(venv) $ python logs2plot.py your_error_summary.json
```

One graph will be generated for every attack. So if you attacked two row pairs (A, B), (C, D) with two different read counts each (X, Y), for a total of 4 attacks, you will get 4 plots:

- read count: X and pair: (A, B)
- read count: X and pair: (C, D)
- read count: Y and pair: (A, B)
- read count: Y and pair: (C, D)

You can control the number of displayed columns with `--plot-columns`. For example if your module has 1024 columns and you provide `--plot-columns 16`, then the DRAM columns will be displayed in groups of 64.

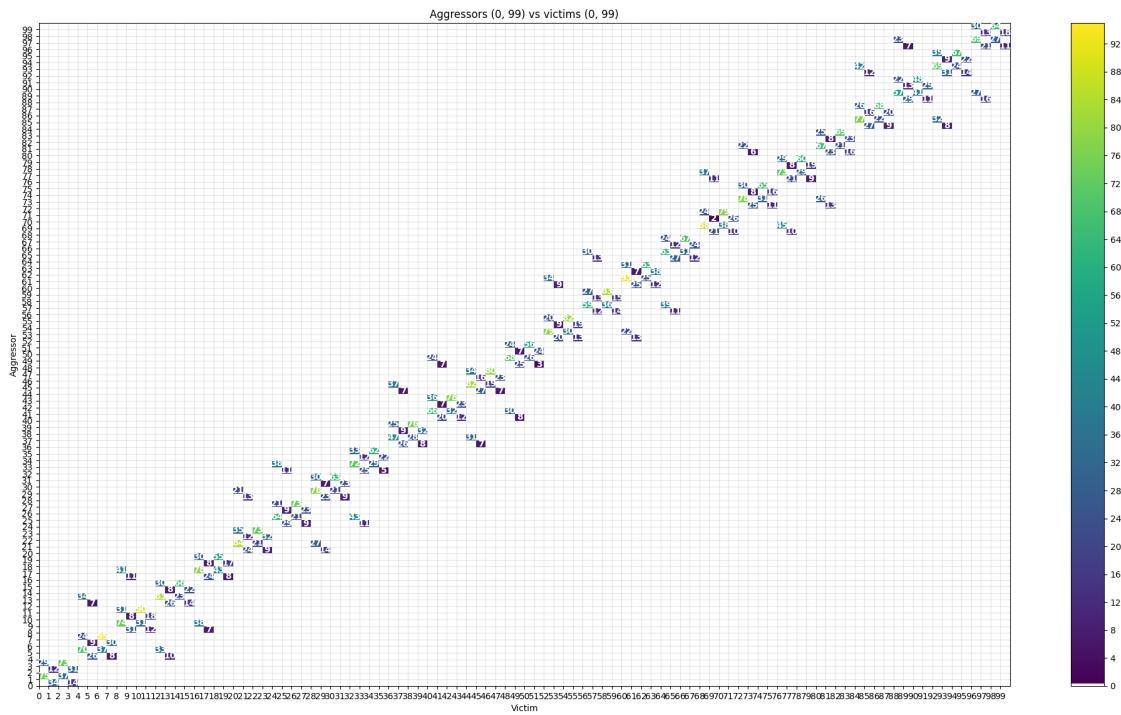
2. Distribution of rows affected by bitflips due to targeting single rows. For example one can generate a graph by calling:

```
(venv) $ python logs2plot.py --aggressors-vs-victims your_error_summary.  
→json
```

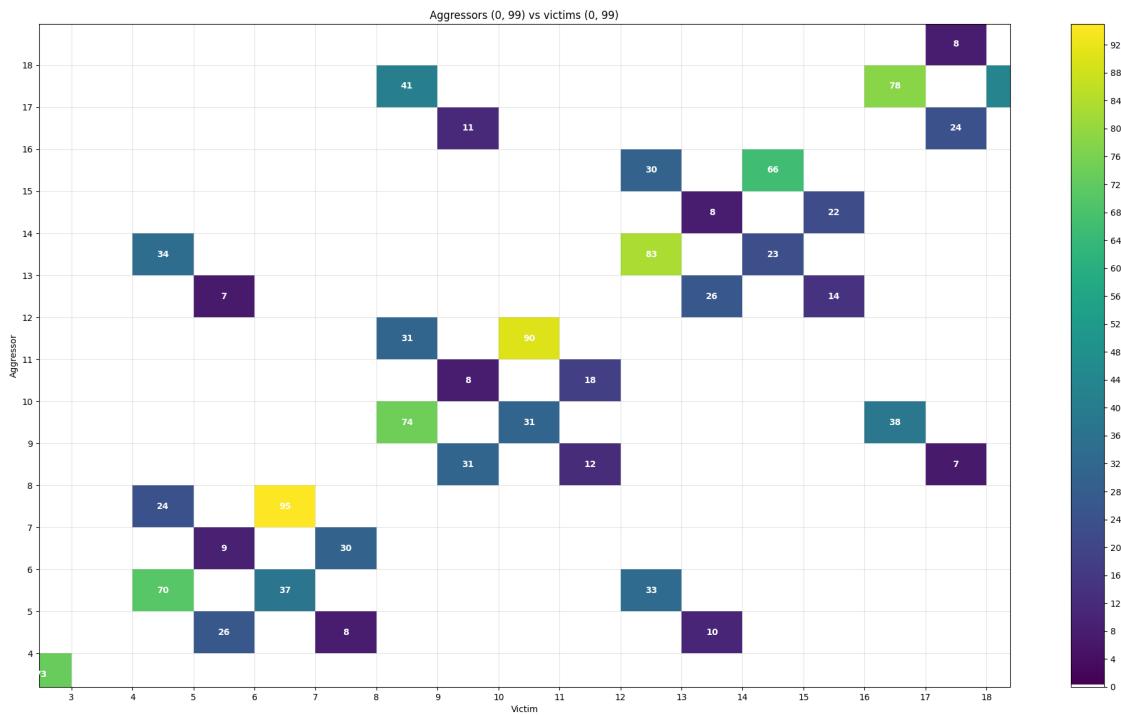
One graph will be generated with victims on the X axis and aggressors on the Y axis. The colors of the tiles indicate how many bitflips occurred on each victim.

You can enable additional annotation with `--annotate_bitflips` so that the number of occurred bitflips will be explicitly labeled on top of each victim tile.

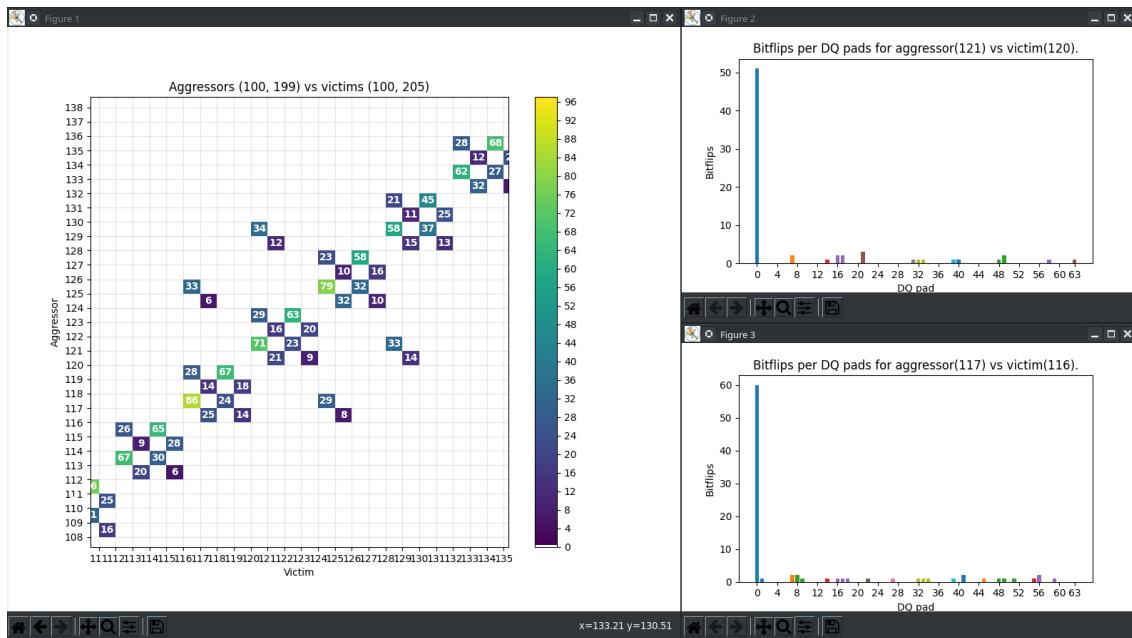
Example plot generated with annotation enabled:



You can zoom-in on interesting parts by using a matplotlib's zoom tool (in the bottom left corner):



This type of plot has built-in DQ per pad statistics for each attack. After clicking a specific tile you will see a new pop-up window with a plot:



3.2 Plot per DQ pad - logs2dq.py

This script allows you to visualize bitflips and group them per DQ pad. Pads themselves are grouped using colors to differentiate modules. Using this script you can visualize and check which module is failing the most.

By default it shows you mean bitflips across all attacks with standard deviation.

First run `rowhammer.py` or `hw_rowhammer.py` with the `--log-dir log_directory` flag.

Then run:

```
python3 logs2dq.py log_directory/your_error_summary.json
```

You can also pass optional arguments:

- `--dq DQ` - how many pads are connected to one module
- `--per-attack` - allows you to also view DQ groupings for each attacked pair of rows

3.3 Use F4PGA Visualizer - logs2vis.py

Similarly to `logs2plot.py`, you can generate visualizations using the [F4PGA Database Visualizer](#).

To view results using the visualizer you need to:

1. Clone and build the visualizer with:

```
git clone https://github.com/chipsalliance/f4pga-database-visualizer
cd f4pga-database-visualizer
npm run build
```

2. Run `rowhammer.py` or `hw_rowhammer.py` with `--log-dir log_directory`

Generate JSON files for the visualizer:

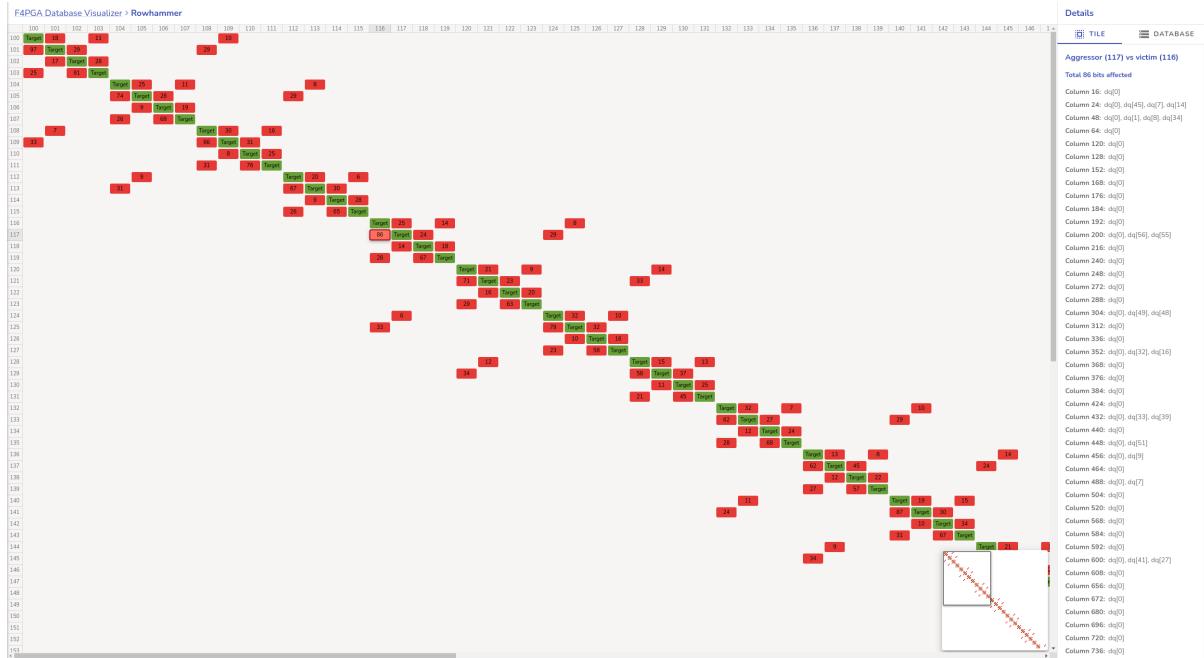
```
python3 logs2vis.py log_directory/your_error_summary.json vis_directory
```

3. Copy generated JSON files from `vis_directory` to `/path/to/f4pga-database-visualizer/dist/production/`

4. Start a simple HTTP server inside the production directory:

```
python -m http.server 8080
```

An example output generated with the `--aggressors-vs-victims` flag looks like this:



CHAPTER FOUR

PLAYBOOK

The [Playbook directory](#) contains a group of Python classes and scripts designed to simplify the process of writing various rowhammer-related tests. These tests can be executed against a hardware platform.

4.1 Payload

Tests are generated as payload data. After generation, this data is transferred to a memory area in the device reserved for this purpose called payload memory. The payload contains an instruction list that can be interpreted by the payload executor module in hardware. The payload executor translates these instructions into DRAM commands. The payload executor connects directly to the DRAM PHY, bypassing the DRAM controller, as explained in [Architecture](#).

4.1.1 Changing payload memory size

Payload memory size can be changed. Of course it can't exceed the memory available on the hardware platform used. Currently, the payload memory size is defined in [common.py](#), as an argument to LiteX:

```
add_argument("--payload-size", default="1024", help="Payload memory size in bytes  
→")
```

The examples shown in this chapter don't require any changes. When writing your own [Configurations](#), you may need to change the default value.

4.2 Row mapping

In the case of DRAM modules, the physical layout of the memory rows in hardware can be different from the logical numbers assigned to them. The nature of rowhammer attack is that only the physically adjacent rows are affected by the aggressor row. To deal with the problem of disparity between physical location and logical enumeration, we various mapping strategies can be implemented:

- TrivialRowMapping - logical address is the same as physical one
- TypeARowMapping - more complex mapping method, reverse-engineered as a part of [this research](#)

-
- TypeBRowMapping - logical address is the physical one multiplied by 2, taken from [this paper](#)

4.3 Row Generator class

Row Generator class is responsible for creating a list of row numbers used in a rowhammer attack. Currently, only one instance of this class is available.

4.3.1 EvenRowGenerator

Generates a list of even numbered rows. Uses the row mapping specified by the [Payload generator class](#) used. Two configuration parameters are needed for EvenRowGenerator:

- nr_rows - number of rows to be generated
- max_row - maximal number to be used. The chosen numbers will be *modulo max_row*

4.3.2 HalfDoubleRowGenerator

Generates a list of rows for a [Half-Double](#) attack. The list will repeat rows to create weight difference between attacks at different distances. Used by HalfDoubleAnalysisPayloadGenerator.

- nr_rows - number of rows in the attack pattern.
- distance_one - does the attack have a distance one component?
- double_sided - is the attack double-sided?
- distance_two - does the attack have a distance two component?
- attack_rows_start - the index of the first attack row.
- max_attack_row_idx - the position of the last attack row relative to attack_rows_start.
- decoy_rows_start - the index of the first decoy row. There are 3 decoy rows that are used as placebos in various situations: to hammer away from the victim but to keep the number of hammers and timing constant.

4.4 Payload generator class

The purpose of the payload generator is to prepare a payload and process the test outcome. It is a class that can be reused in different tests [Configurations](#). Payload generators are located in [payload_generators directory](#)

4.4.1 Available payload generators

Row mapping and row generator settings are combined into a payload generator class. Currently, only two payload generators are available.

RowListPayloadGenerator

RowListPayloadGenerator is a simple payload generator that can use a RowGenerator class to generate rows, and then generate a payload that hammers that list of rows (hammering is a term used to describe multiple read operations on the same row). It can also issue refresh commands to the DRAM module. Here are the configs that can be used in *payload_generator_config* for this payload generator:

- `row_mapping` - this is the *Row mapping* used
- `row_generator` - this is the *Row Generator class* used to generate the rows
- `row_generator_config` - parameters for the row generator
- `verbose` - should verbose output be generated (true or false)
- `fill_local` - when enabled, permits shrinking the filled memory area to just the aggressors and the victim
- `read_count` - number of hammers (reads) per row
- `refresh` - should refresh be enabled (true or false)

Example *Configurations* for this test were provided as `configs/example_row_list_*.cfg` files. Some of them require a significant amount of memory declared as `payload memory`. To execute a minimalistic example from within rowhammer-tester repo, enter:

```
source venv/bin/activate
export TARGET=arty # change accordingly
cd rowhammer_tester/scripts/playbook/
python playbook.py configs/example_row_list_minimal.cfg
```

Expected output:

```
Progress: [=====] 65536 / 65536
Row sequence:
[0, 2, 4, 6, 14, 12, 10, 8, 16, 18]
Generating payload:
  tRAS = 5
  tRP = 3
  tREFI = 782
  tRFC = 32
  Repeatable unit: 930
  Repetitions: 93
  Payload size = 0.10KB / 1.00KB
  Payload per-row toggle count = 0.010K x10 rows
  Payload refreshes (if enabled) = 10 (disabled)
  Expected execution time = 1903 cycles = 0.019 ms
```

(continues on next page)

(continued from previous page)

Transferring the payload ...

Executing ...

Time taken: 0.738 ms

Progress: [==

] 3338 / 65536 (Errors: 1287)

...

HammerTolerancePayloadGenerator

HammerTolerancePayloadGenerator is a payload generator for measuring and characterizing rowhammer tolerance. It can provide information about how many rows and bits are susceptible to the rowhammer attack. It can also provide information about where the susceptible bits are located.

A series of double-sided hammers against the available group of victim rows is performed. The double-sided hammers increase in intensity based on `read_count_step` parameter. Here are the parameters that can be specified in `payload_generator_config` for this payload generator:

- `row_mapping` - this is the *Row mapping* used
- `row_generator` - this is the *Row Generator class* used to generate the rows
- `row_generator_config` - parameters for the row generator
- `verbose` - should verbose output be generated (true or false)
- `fill_local` - when enabled, permits shrinking the filled memory area to just the aggressors and the victim
- `nr_rows` - number of rows to conduct the experiment over. This is the number of aggressor rows. Victim rows will be 2 times fewer than this number. For example, to perform hammering for 32 victim rows, use 34 as the parameter value
- `read_count_step` - this is how much to increment the hammer count between multiple tests for the same row. This is the number of hammers on single side (total number of hammers on both sides is 2x this value)
- `initial_read_count` - hammer count for the first test for a given row. Defaults to `read_count_step` if unspecified.
- `distance` - distance between aggressors and victim. Defaults to 1.
- `baseline` - when enabled, a retention effect baseline is collected by hammering distant rows for the same amount of time that the aggressors would be hammered.
- `first_dummy_row` - location of the first of two dummy rows used for baselining.
- `iters_per_row` - number of times the hammer count is incremented for each row

The results are a series of histograms with appropriate labeling.

Example *Configurations* for this test were provided as `configs/example_hammer_*.cfg` files. Some of them require a significant amount of memory declared as `payload` memory. To execute a minimalistic example from within `rowhammer-tester` repo, enter:

```
source venv/bin/activate
export TARGET=arty # change accordingly
cd rowhammer_tester/scripts/playbook/
python playbook.py configs/example_hammer_minimal.cfg
```

Expected output:

```
Progress: [=====] 3072 / 3072
Generating payload:
tRAS = 5
tRP = 3
tREFI = 782
tRFC = 32
Repeatable unit: 186
Repetitions: 93
Payload size = 0.04KB / 1.00KB
Payload per-row toggle count = 0.010K x2 rows
Payload refreshes (if enabled) = 10 (disabled)
Expected execution time = 1263 cycles = 0.013 ms

Transferring the payload ...

Executing ...
Time taken: 0.647 ms

Progress: [=====] 323 / 1024 (Errors: 320)
...
```

HalfDoubleAnalysisPayloadGenerator

Half-Double is a Rowhammer phenomenon where accesses to both distance-one and distance-two neighbours of a victim row are used to generate bit flips. This payload generator allows us to characterize the Half-Double effect on a memory part.

For each candidate victim row, the analysis starts out with the maximum number of hammers and minimum dilution level. Then, we proceed as follows:

1. Dilution is increased until pure distance-one attacks stop working.
2. Verify that pure distance-two attack doesn't work.
3. Increase dilution level and record the number of bit flips in the victim until either the bit flips stop or maximum dilution level is reached.
4. Once maximum dilution level is reached or bit flips stop, reduce hammer count by step and reset dilution to initial level and retry step 3. Repeat until the lowest hammer count is reached.

Note: the hammer count changes on a linear scale and dilution changes on an exponential scale.

Results are presented as a table of values with columns representing hammer count and the rows representing dilution levels. See Tables 2 and 3 in the Half-Double white paper as examples.

- `max_total_read_count` - maximum number of hammers issued to any given row during an iteration.
- `read_count_steps` - the amount to decrement the number of hammers for each iteration of the outer loop.
- `initial_dilution` - initial value for dilution. Dilution resets to this value at the beginning of the inner loop.
- `dilution_multiplier` - dilution is multiplied by this value for each iteration of the inner loop.
- `verbose` - generates more output.
- `row_mapping` - specifies the style in which the rows are mapped on the chip.
- `attack_rows_start` - starting row number for rows actually used to attack the victim.
- `max_attack_row_idx` - index measured from `attack_rows_start` for the last attack row.
- `decoy_rows_start` - the position of the first decoy row. There are three decoy rows. They are used as placebos during pure distance one portions of the experiment to make the number of hammers and their timing comparable.
- `max_dilution` - maximum value for dilution.
- `fill_local` - only reinitialize affected rows between experiments, as an optimization.

4.5 Configurations

Test configuration files are represented as JSON files. An example:

```
{
    "inversion_divisor" : 2,
    "inversion_mask" : "0b10",
    "payload_generator" : "RowListPayloadGenerator",
    "payload_generator_config" : {
        "row_mapping" : "TypeARowMapping",
        "row_generator" : "EvenRowGenerator",
        "read_count" : 27,
        "max_iteration" : 10,
        "verbose" : true,
        "refresh" : false,
        "fill_local" : true,
        "row_generator_config" : {
            "nr_rows" : 10,
            "max_row" : 64
        }
    }
}
```

Different parameters are supported:

- `payload_generator` - name of the *Payload generator class* to use
- `row_pattern` - pattern that will be stored in rows

- `inversion_divisor` and `inversion_mask` - controls which rows get the inverted pattern described in [Inversion](#)
- `payload_generator_config` - these parameters are specific for [Payload generator class](#) used

4.5.1 Inversion

If needed, the data pattern used for some of the tested rows can be bitwise-inverted.

Two parameters are used to specify which rows are to be inverted:

- `inversion_divisor`
- `inversion_mask`

An example. `inversion_divisor = 8`, `inversion_mask = 0b10010010` (bits 1, 4 and 7 are “on”). We iterate through all row numbers $0, 1, 2, 3, 4, \dots, 8, 9, 10, \dots$. First, a modulo `inversion_divisor` operation is performed on a row number. In our case it’s mod 8. Next, we check if the bit in `inversion_mask` in the position corresponding to our row number (after modulo) is “on” or “off”. If it’s “on”, this whole row will be inverted. The results for our example are shown in a table below.

Table 4.1: Inversion example

Row num- ber	Row num- ber	Value
mod- ulo divisor (8)		
0	0	pattern
1	1	inverted pattern
2	2	pattern
3	3	pattern
4	4	inverted pattern
5	5	pattern
6	6	pattern
7	7	inverted pattern
8	0	pattern
9	1	inverted pattern
10	2	pattern
11	3	pattern
12	4	inverted pattern

CHAPTER
FIVE

DRAM MODULES

When building one of the targets in [rowhammer_tester/targets](#), a custom DRAM module can be specified using the `--module` argument. To find the default modules for each target, check the output of `--help`.

Note: Specifying different DRAM module makes most sense on boards that allow to easily replace the DRAM module, such as on ZCU104. On other boards it would be necessary to desolder the DRAM chip and solder a new one.

5.1 Adding new modules

[LiteDRAM](#) controller provides out-of-the-box support for many DRAM modules. Supported modules can be found in [litedram/modules.py](#). If a module is not listed there, you can add a new definition.

To make development more convenient, modules can be added in the rowhammer-tester repository directly in file [rowhammer_tester/targets/modules.py](#). These definitions will be used before definitions in LiteDRAM.

Note: After ensuring that the module works correctly, a Pull Request to LiteDRAM should be created to add support for the module.

To add a new module definition, use the existing ones as a reference. New module class should derive from `SDRAMModule` (or the helper classes, e.g. `DDR4Module`). Timing/geometry values for a module have to be obtained from the relevant DRAM module's datasheet. The timings in classes deriving from `SDRAMModule` are specified in nanoseconds. The timing value can also be specified as a 2-element tuple `(ck, ns)`, in which case `ck` is the number of clock cycles and `ns` is the number of nanoseconds (and can be `None`). The highest of the resulting timing values will be used.

5.2 SPD EEPROM

On boards that use DIMM/SO-DIMM modules (e.g. ZCU104) it is possible to read the contents of the DRAM module's **SPD EEPROM memory**. SPD contains several essential module parameters that the memory controller needs in order to use the DRAM module. SPD EEPROM can be read over I2C bus.

5.2.1 Reading SPD EEPROM

To read the SPD memory use the script `rowhammer_tester/scripts/spd_eeprom.py`. First prepare the environment as described in [Controlling the board](#). Then use the following command to read the contents of SPD EEPROM and save it to a file, for example:

```
python rowhammer_tester/scripts/spd_eeprom.py read MTA4ATF51264HZ-3G2J1.bin
```

The contents of the file can then be used to get DRAM module parameters. Use the following command to examine the parameters:

```
python rowhammer_tester/scripts/spd_eeprom.py show MTA4ATF51264HZ-3G2J1.bin 125e6
```

Note that system clock frequency must be passed as an argument to determine timing values in controller clock cycles.

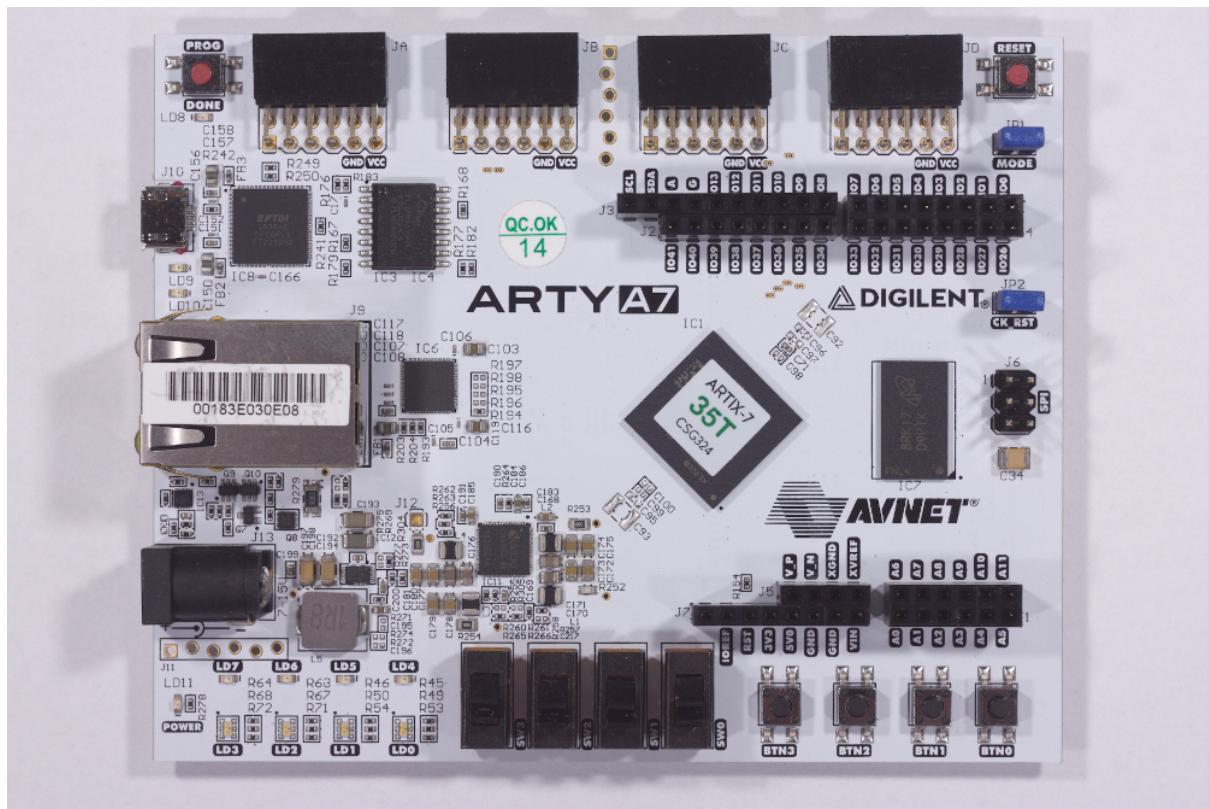
5.2.2 Using SPD data

The memory controller is able to set the timings read from an SPD EEPROM during system boot. The only requirement here is that the SoC is built with I2C controller, and I2C pins are routed to the (R)DIMM module. There is no additional action required from system user. The timings will be set automatically.

CHAPTER SIX

ARTY-A7 BOARD

The Arty-A7 board allows testing its on-board DDR3 module. The board is designed around the Artix-7™ Field Programmable Gate Array (FPGA) from Xilinx.



The following instructions explain how to set up the board.

6.1 Board configuration

Connect the board USB and Ethernet cables to your computer and configure the network. The board's IP address will be 192.168.100.50 (so you could e.g. use 192.168.100.2/24). The IP_ADDRESS environment variable can be used to modify the board's address. Next, generate the FPGA bitstream:

```
export TARGET=arty
make build
```

Note: This will by default target Arty A7 with the XC7A35TICSG324-1L FPGA. To build for XC7A100TCG324-1, use `make build TARGET_ARGS="--variant a7-100"`

The results will be located in: `build/arty/gateware/digilent_arty.bit`. To upload it, use:

```
export TARGET=arty  
make upload
```

Note: By typing `make` (without `build`) LiteX will generate build files without invoking Vivado.

To save bitstream in flash memory, use:

```
export TARGET=arty  
make flash
```

Bitstream will be loaded from flash memory upon device power-on or after a PROG button press.

ZCU104 BOARD

The **ZCU104 board** enables testing DDR4 SO-DIMM modules. It features a Zynq UltraScale+ MPSoC device consisting of PS (Processing System with quad-core ARM Cortex-A53) and PL (programmable logic).

On the ZCU104 board the Ethernet PHY is connected to PS instead of PL. For this reason it is necessary to route the Ethernet/EtherBone traffic through PC <-> PS <-> PL. To do this, a simple EtherBone server has been implemented (the source code can be found in the `firmware/zcu104/etherbone/` directory).

The following instructions show how to set up the board for the first time.

7.1 Board configuration

To make the ZCU104 boot from the SD card it is necessary to ensure proper switch configuration. The mode switch (SW6) consisting of 4 switches is located near the FMC LPC Connector (J5) (the same side of the board as USB, HDMI, Ethernet). For details, see “ZCU104 Evaluation Board User Guide (UG1267)”. To use an SD card, configure the switches as follows:

1. ON
2. OFF
3. OFF
4. OFF

7.2 Preparing SD card

For easiest setup, get the pre-built SD card image `zcu104.img` from [github releases](#). It has to be loaded to the microSD card. To load it to the SD card, insert the card into your PC card slot and find the device name. `lsblk` can be used to check available devices. Example output can look like:

```
$ lsblk
NAME      MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda        8:0     0 931.5G  0 disk
└─sda1     8:1     0 931.5G  0 part /data
sdb        8:16    1 14.8G  0 disk
└─sdb1     8:17    1   128M  0 part /run/media/BOOT
```

(continues on next page)

(continued from previous page)

```

└─sdb2      8:18    1   128M  0 part /run/media/rootfs
nvme0n1    259:0   0 476.9G  0 disk
└─nvme0n1p1 259:1   0   512M  0 part /boot
└─nvme0n1p2 259:2   0 476.4G  0 part /

```

In the example output the SD card is sdb with two partitions sdb1 and sdb2.

Warning: Make sure that you have selected the proper device name or you may damage the hard drive on your system! Check the device SIZE to match the size of your microSD card and compare the outputs of the lsblk command with and without the SD card inserted.

Make sure to unmount all partitions on the card before loading the image. For example, assuming the SD card is /dev/sdb (device is without a number), use sudo umount /dev/sdb1 /dev/sdb2 to unmount its partitions.

To load the image, use the following command, replacing <DEVICE> according to the output of lsblk (in the example above it would be /dev/sdb):

```
sudo dd status=progress oflag=sync bs=4M if=zcu104.img of=<DEVICE>
```

Now, the microSD card should be ready to use. If it has been loaded successfully, you should be able to mount the two partitions (BOOT and rootfs) on your PC and browse the files. First, check if your system hasn't auto-mounted the partitions. If not, then you can for example use:

```

sudo mkdir -p /mnt/boot /mnt/rootfs
sudo mount /dev/sdb1 /mnt/boot
sudo mount /dev/sdb2 /mnt/rootfs

```

7.3 Loading bitstream

Instead of loading the bitstream through the JTAG interface, it must be copied to the microSD card BOOT partition (FAT32). The bitstream will be then loaded by the bootloader during system startup.

The prebuilt card image comes with a sample bitstream, but in order to use the provided rowhammer Python scripts, a fresh bitstream must be created. Build the bitstream as you would do for other boards:

```

export TARGET=zcu104
export IP_ADDRESS=... # optional
make build

```

Note: You only need to export IP_ADDRESS if you want to use a different address than the default one (see *Network setup*).

The resulting bitstream file will be located in build/zcu104/gateware/zcu104.bit. Copy it to the BOOT partition (FAT32) of the microSD card. Make sure it is named zcu104.bit.

When the SD card is ready, insert it into the microSD card slot on your ZCU104 board and power it on.

7.4 Verifying if bitstream is loaded

The first indication that the bitstream has been loaded successfully are the onboard LEDs. When the board is powered up, the LED will be red, which then turns green if the bitstream is loaded successfully. The ZCU104 bitstream will also make the four LEDs near the user buttons turn on and off in a circular pattern.



Fig. 7.1: The board without a bitstream.



Fig. 7.2: The state when the bitstream has been loaded successfully.

The serial console over USB can be used to further check if the board is working correctly.

7.5 ZCU104 microUSB

ZCU104 has a microUSB port connected to the FTDI chip. It provides 4 channels, which are connected as follows:

- Channel A is configured to support the JTAG chain.
- Channel B implements UART0 MIO18/19 connections.
- Channel C implements UART1 MIO20/21 connections.
- Channel D implements UART2 PL-side bank 28 4-wire interface.

In general they should show up as subsequent /dev/ttyUSBx devices (0-3 if no others were present). Channel B is connected to the console in the PS Linux system.

To login to the board connect the microUSB cable to the PC and find Channel B among the ttyUSB devices in your system. If there are only ttyUSB0 to ttyUSB3 then Channel B will be ttyUSB1.

Then use e.g. `picocom` or `minicom` to open the serial console. Install one of them if not already installed on your system. With `picocom` use the following command (may require sudo):

```
picocom -b 115200 /dev/ttyUSB1
```

Then press enter, and when you see the following prompt,

```
Welcome to Buildroot  
buildroot login:
```

use login root and empty password. You can set up a password if needed.

7.6 Network setup

Connect the ZCU104 board to your local network (or directly to a PC) using an Ethernet cable.

The board uses a static IP address. By default it will be 192.168.100.50. If it does not conflict with your local network configuration you can skip this section. (the default configuration can be found [here](#)).

To verify connectivity, use `ping 192.168.100.50`. You should see data being transmitted, e.g.

```
$ ping 192.168.100.50  
PING 192.168.100.50 (192.168.100.50) 56(84) bytes of data.  
64 bytes from 192.168.100.50: icmp_seq=1 ttl=64 time=0.332 ms  
64 bytes from 192.168.100.50: icmp_seq=2 ttl=64 time=0.072 ms  
64 bytes from 192.168.100.50: icmp_seq=3 ttl=64 time=0.081 ms
```

7.6.1 Modifying network configuration

If you need to modify the configuration, edit file `/etc/network/interfaces`. The Linux rootfs on the SD card is fairly minimal, so there is only the `vi` editor available. You could also just mount the card on your PC and edit the file.

After changing the configuration, reboot the board (type `reboot` in the serial console) and test if you can ping it with `ping <NEW_IP_ADDRESS>`.

7.7 SSH access

These instructions are optional but can be useful for more convenient updates of the bitstream (no need to remove the SD card from the slot on ZCU104).

Note: SSH on the board is configured to allow passwordless root access for simplicity but if that poses a security risk, modify `/etc/ssh/sshd_config` according to your needs and add a password for root.

You can login over SSH using (replace the IP address if you modified board network configuration):

```
ssh root@192.168.100.50
```

To access the boot partition, first mount it with:

```
mount /dev/mmcblk0p1 /boot
```

This can be automated by adding the following entry in `/etc/fstab`:

```
/dev/mmcblk0p1 /boot vfat rw 0 2
```

When the boot partition is mounted, you can use `scp` to load the new bitstream, e.g.

```
scp build/zcu104/gateware/zcu104.bit root@192.168.100.50:/boot/zcu104.bit
```

Then use the `reboot` command to restart the board.

7.8 Controlling the board

When the setup has been finished the board can be controlled as any other board. Just make sure to use `export TARGET=zcu104` before using the scripts (and `export IP_ADDRESS=...` if you modified the network configuration).

7.9 ZCU104 SD card image

The easiest way is to use the prebuilt SD card image, as described in [Preparing SD card](#) section. However, it is possible to build the image from source, if needed.

The SD card image consists of boot partition and rootfs. Currently, only rootfs is built using buildroot. The boot partition content has to be built manually.

In the future, the buildroot configuration should be revised to build all the required software. Initial configuration has been included but it is still a work in progress and does not boot.

7.9.1 Bootloaders & kernel

Currently, we are using Xilinx FSBL, but it should be possible to use U-Boot SPL ([link1](#), [link2](#)).

FSBL and PMU firmware can be built with following the steps:

- <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842462/Build+PMU+Firmware>
- <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841798/Build+FSBL>

Create a project from the Vivado example project “Base Zynq UltraScale+ MPSoC” for ZCU104 eval board. Open the PS IP configurator and add the following: * PS-PL Interfaces -> AXI HPM0 FPD (32-bit), AXI HPM1 FPD (32-bit) * disable Carrier Detect in Memory Interfaces -> SD -> SD 0

The following script can be used to generate FSBL, PMU firmware and Device Tree:

```
#!/usr/bin/tclsh

set hwdesign [open_hw_design PATH/T0/Base_Zynq_MPSoC_wrapper.hdf]

generate_app -hw $hwdesign -os standalone -proc psu_cortexa53_0 -app zynqmp_fsbl -  
-compile -sw fsbl -dir ./fsbl/  
generate_app -hw $hwdesign -os standalone -proc psu_pmu_0 -app zynqmp_pmufw -  
-compile -sw pmufw -dir ./pmufw

set_repo_path PATH/T0/device-tree-xlnx
create_sw_design device-tree -os device_tree -proc psu_cortexa53_0
generate_target -dir dts

close_hw_design [current_hw_design]
```

The Device Tree generated by Vivado is missing the ethernet-phy node. Modify `pcw.dtsi` as follows:

```
&gem3 {  
    phy-mode = "rgmii-id";  
    status = "okay";  
    xlnx,ptp-enet-clock = <0x0>;  
    phy0: phy@c {
```

(continues on next page)

(continued from previous page)

```

reg = <0xc>;
ti,rx-internal-delay = <0x8>;
ti,tx-internal-delay = <0xa>;
ti,fifo-depth = <0x1>;
ti,rxctrl-strap-worka;
};

};

```

Then generate the Device Tree Blob in the dts directory:

```

gcc -I include -I . -E -nostdinc -undef -D__DTS__ -x assembler-with-cpp -o system.
↳dts system-top.dts
dtc -I dts -O dtb -o system.dtb system.dts

```

Build the rest of the required components:

- ARM trusted firmware: (<https://github.com/Xilinx/arm-trusted-firmware.git>) e6eea88b14aaaf456c49f9c7e6747584224648cb9 (tag: xlnx_rebase_v2.2)
- U-Boot: (<https://github.com/Xilinx/u-boot-xlnx.git>) d8fc4b3b70bccf1577dab69f6ddfd4ada9a93bac (tag: xilinx-v2018.3)
- Linux kernel: (<https://github.com/Xilinx/linux-xlnx.git>) 22b71b41620dac13c69267d2b7898ebfb14c954e (tag: xlnx_rebase_v5.4_2020.1)

Note: It may be necessary to apply the patches from `firmware/zcu104/buildroot/board/zynqmp/patches` when building U-Boot/Linux.

When building U-Boot make sure to update its configuration (`u-boot-xlnx/.config`) with the following options:

```

CONFIG_USE_BOOTARGS=y
CONFIG_BOOTARGS="earlycon clk_ignore_unused console=ttyPS0,115200 root=/dev/
↳mmcblk0p2 rootwait rw earlyprintk rootfstype=ext4"
CONFIG_USE_BOOTCOMMAND=y
CONFIG_BOOTCOMMAND="load mmc 0:1 0x2000000 zcu104.bit; fpga load 0 0x2000000
↳$filesize; load mmc 0:1 0x2000000 system.dtb; load mmc 0:1 0x3000000 Image;_
↳booti 0x3000000 - 0x2000000"

```

These configure U-Boot to load the bitstream from the SD card and then start the system. Unfolding CONFIG_BOOTCOMMAND we can see:

```

load mmc 0:1 0x2000000 zcu104.bit
fpga load 0 0x2000000 $filesize
load mmc 0:1 0x2000000 system.dtb
load mmc 0:1 0x3000000 Image
booti 0x3000000 - 0x2000000

```

Example of building ARM Trusted firmware:

```
make distclean
make -j`nproc` PLAT=zynqmp RESET_TO_BL31=1
```

Example of building U-Boot:

```
make -j`nproc` distclean
make xilinx_zynqmp_zcu104_revC_defconfig
# now modify .config directly or using `make menuconfig` as described earlier
make -j`nproc`
```

Example of building Linux:

```
make -j`nproc` ARCH=arm64 distclean
make ARCH=arm64 xilinx_zynqmp_defconfig
# optional `make menuconfig` 
make -j`nproc` ARCH=arm64 dtbs
make -j`nproc` ARCH=arm64
```

Then download [zynq-mkbootimage](#) and prepare the following boot.bif file:

```
image:
{
    [fsbl_config] a53_x64
    [bootloader] fsbl.elf
    [pmufw_image] pmufw.elf
    [, destination_cpu=a53-0, exception_level=el-2] bl31.elf
    [, destination_cpu=a53-0, exception_level=el-2] u-boot.elf
}
```

Open a terminal and make sure that the filepaths specified in boot.bif are correct. Then use ``mkbootimage -zynqmp boot.bif boot.bin`` to create the boot.bin file.

7.9.2 Root filesystem

Download buildroot

```
git clone git://git.buildroot.net/buildroot
git checkout 2020.08.2
```

Note: As of time of writing `git checkout f45925a951318e9e53bead80b363e004301adc6f` was required to avoid fakeroot errors when building.

Then prepare configuration using external sources and build everything:

```
make BR2_EXTERNAL=/PATH/TO/REPO/rowhammer-tester/firmware/zcu104/buildroot zynqmp_
↳zcu104_defconfig
make -j`nproc`
```

7.9.3 Flashing SD card

One can use `fdisk` to directly partition the SD card `/dev/xxx` or use the provided `genimage` configuration to create an SD card image that can be then directly copied to the SD card. The second method is usually more convenient.

Formatting SD card manually

Use `fdisk` or other tool to partition the SD card. The recommended partitioning scheme is as follows:

- Partition 1, FAT32, 128M
- Partition 2, ext4, 128M

Then create the filesystems:

```
sudo mkfs.fat -F 32 -n BOOT /dev/OUR_SD_CARD_PARTITION_1
sudo mkfs.ext4 -L rootfs /dev/OUR_SD_CARD_PARTITION_2
```

Write the rootfs:

```
sudo dd status=progress oflag=sync bs=4M if=/PATH/T0/BUILDROOT/output/images/
↪rootfs.ext4 of=/dev/OUR_SD_CARD_PARTITION_2
```

Mount the boot partition and copy the boot files and kernel image created earlier and the ZCU104 bitstream:

```
cp boot.bin /MOUNT/POINT/BOOT/
cp /PATH/T0/rowhammer-tester/build/zcu104/gateware/zcu104.bit /MOUNT/POINT/BOOT/
cp /PATH/T0/linux-xlnx/arch/arm64/boot/Image /MOUNT/POINT/BOOT/
cp /PATH/T0/linux-xlnx/arch/arm64/boot/dts/xilinx/zynqmp-zcu104-revA.dtb /MOUNT/
↪POINT/BOOT/system.dtb
```

Note: make sure to name the device tree blob `system.dtb` for the U-Boot to be able to find it (as shown in above commands).

Using genimage

By default ZCU104 buildroot configuration will also build the `genimage` tool for the host system. Image configuration is described in the `firmware/zcu104/image.cfg` file. There is also a script `firmware/zcu104/genimage.sh` for convenience. Run it without arguments to get help. Then run it providing correct paths to all the required files to generate the `zcu104.img` file.

The image can be then copied to the SD card device (not partition! so e.g. `/dev/sdb`, not `/dev/sdb1`) using `dd`:

```
sudo dd status=progress oflag=sync bs=4M if=/PATH/T0/zcu104.img of=/dev/OUR_SD_
↪CARD
```

CHAPTER
EIGHT

LPDDR4 TEST BOARD



LPDDR4 Test Board is a platform developed by Antmicro for testing LPDDR4 memory. It uses Xilinx Kintex-7 FPGA (XC7K70T-FBG484) and by default includes a custom SO-DIMM module with Micron's MT53E256M16D1 LPDDR4 DRAM.

The hardware is open and can be found on GitHub:

- Test board: <https://github.com/antmicro/lpddr4-test-board>
- Testbed: <https://github.com/antmicro/lpddr4-testbed>

8.1 Board configuration

First insert the LPDDR4 DRAM module into the socket and make sure that jumpers are set in correct positions:

- VDDQ (J10) should be set in position 1V1
- MODE2 should be set in position FLASH

Then connect the board USB and Ethernet cables to your computer and configure the network. The board's IP address will be 192.168.100.50 (so you could e.g. use 192.168.100.2/24). The IP_ADDRESS environment variable can be used to modify the board's address. Next, generate the FPGA bitstream:

```
export TARGET=lpddr4_test_board  
make build
```

The results will be located in: build/lpddr4_test_board/gateware/antmicro_lpddr4_test_board.bit. To upload it, use:

```
export TARGET=lpddr4_test_board  
make upload
```

Note: By typing make (without build) LiteX will generate build files without invoking Vivado.

To save bitstream in flash memory, use:

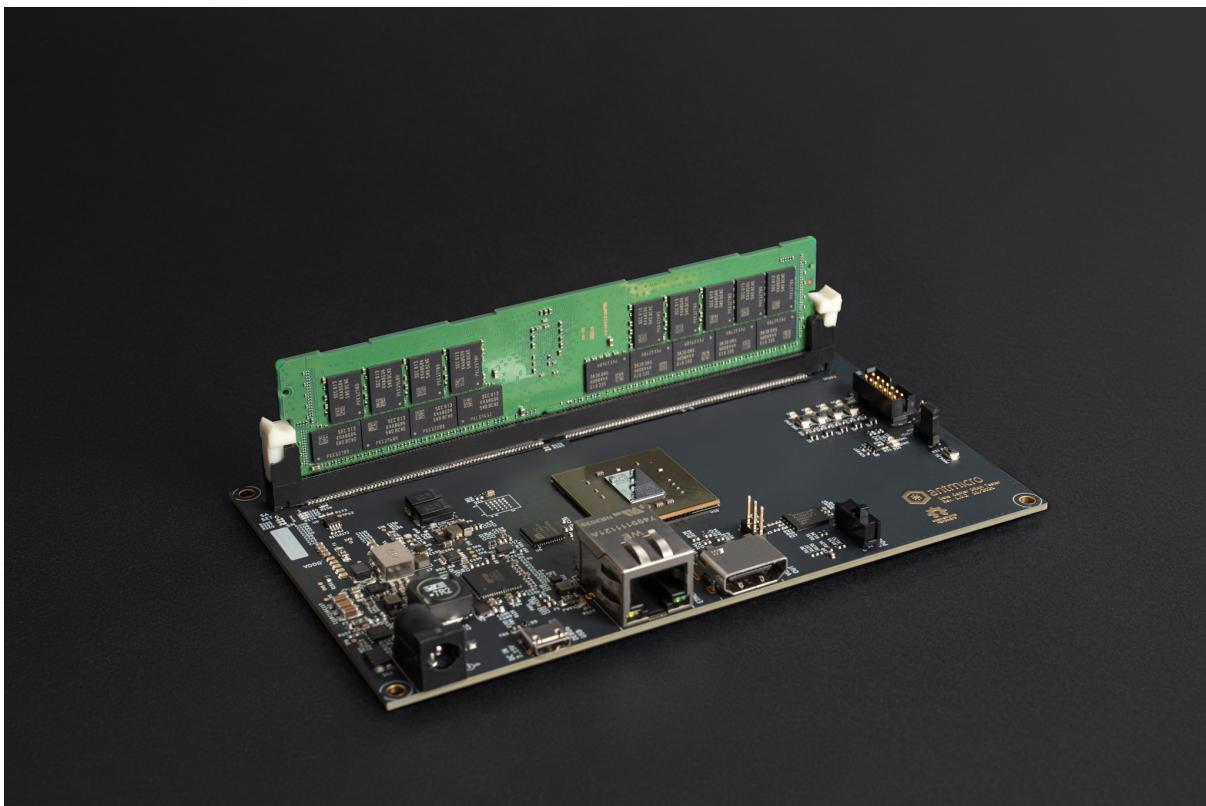
```
export TARGET=lpddr4_test_board  
make flash
```

Warning: There is a JTAG/FLASH jumper named MODE2 on the right side of the board. Unless it's set to the FLASH setting, the FPGA will load the bitstream received via JTAG.

Bitstream will be loaded from flash memory upon device power-on or after a PROG button press.

CHAPTER
NINE

DATA CENTER DRAM TESTER



The data center DRAM tester is an open source hardware test platform that enables testing and experimenting with various DDR4 RDIMMs (Registered Dual In-Line Memory Module).

The hardware is open and can be found on GitHub: [https://github.com/antmicro/
data-center-dram-tester/](https://github.com/antmicro/data-center-dram-tester/)

The following instructions explain how to set up the board.

9.1 Board configuration

First connect the board USB and Ethernet cables to your computer, plug the board to the socket and turn it on using power switch. Then configure the network. The board's IP address will be 192.168.100.50 (so you could e.g. use 192.168.100.2/24). The IP_ADDRESS environment variable can be used to modify the board's address. Next, generate the FPGA bitstream:

```
export TARGET=ddr4_datacenter_test_board  
make build
```

Note: By typing make (without build) LiteX will generate build files without invoking Vivado.

The results will be located in: build/ddr4_datacenter_test_board/gateware/antmicro_datacenter_ddr4_test_board.bit. To upload it, use:

```
export TARGET=ddr4_datacenter_test_board  
make upload
```

To save bitstream in flash memory, use:

```
export TARGET=ddr4_datacenter_test_board  
make flash
```

Warning: There is a JTAG/SPI jumper named MODE2 on the right side of the board. Unless it's set to the SPI setting, the FPGA will load the bitstream received via JTAG.

Bitstream will be loaded from flash memory upon device power-on or after a PROG button press.

CHAPTER
TEN

DDR5 TESTER



The DDR5 tester is an open source hardware test platform that enables testing and experimenting with various DDR5 RDIMMs (Registered Dual In-Line Memory Module).

The hardware is open and can be found on GitHub: <https://github.com/antmicro/ddr5-tester/>
The following instructions explain how to set up the board.

10.1 Rowhammer Tester Target Configuration

First connect the board USB and Ethernet cables to your computer, plug the board to the socket and turn it on using power switch. Then configure the network. The board's IP address will be 192.168.100.50 (so you could e.g. use 192.168.100.2/24). The `IP_ADDRESS` environment variable can be used to modify the board's address. Next, generate the FPGA bitstream:

```
export TARGET=ddr5_tester
make build TARGET_ARGS="--l2-size 256 --build --iodelay-clk-freq 400e6 --bios-lto_
↪--rw-bios --module MTC10F1084S1RC --no-sdram-hw-test"
```

Note: `--l2-size 256` sets L2 cache size to 256 bytes

`--no-sdram-hw-test` disables hw accelerated memory test

Note: By typing `make` (without `build`) LiteX will generate build files without invoking Vivado.

The results will be located in: `build/ddr5_tester/gateware/antmicro_ddr5_tester.bit`. To upload it, use:

```
export TARGET=ddr5_tester
make upload
```

To save bitstream in flash memory, use:

```
export TARGET=ddr5_tester
make flash
```

Warning: There is a JTAG/SPI jumper named `MODE` on the right side of the board. Unless it's set to the SPI setting, the FPGA will load the bitstream received via JTAG.

Bitstream will be loaded from flash memory upon device power-on or after a PROG button press.

10.2 Linux Target Configuration

There's another DDR5 Tester target provided. The `ddr5_tester_linux` is a linux capable target (**with no rowhammer tester**) that allows you to define custom DDR5 tester module utilizing linux-based software.

10.2.1 Base DDR5 Tester Linux Options

The `ddr5_tester_linux` is configured via specifying `TARGET_ARGS` variable. The required arguments for the `ddr5_tester_linux` are:

Option	Documentation
--build	When specified will invoke synthesis and hardware analysis tool (Vivado by default). Will produce programmable bitstream.
--l2-size	Specifies the L2 cache size.
--iodelay-clk-freq	IODELAY clock frequency.
--module	The DDR5 module to be used.
--with-wishbone-memory	VexRiscV SMP specific option. Disables native LiteDRAM interface.
--wishbone-force-32b	VexRiscV SMP specific option. Forces the wishbone bus to be 32 bits wide.

Additionally, **etherbone** or **ethernet** can be set up with either:

10.2.2 Ethernet Options

Option	Documentation
--with-ethernet	Sets up ethernet for DDR5 Tester board.
--remote-ip-address	The IP address of the remote machine connected to DDR5 Tester.
--local-ip-address	Local (DDR5 Tester's) IP address.

10.2.3 Etherbone Options

Option	Documentation
--with-etherbone	Sets up ethernet for DDR5 Tester board.
--ip-address	IP address to be used for the etherbone.
--mac-address	MAC address to be used for the etherbone.

10.2.4 Building DDR5 Tester Linux Target

After having configured the DDR5 Tester Linux, the target can be build with `make build` Makefile target. Use example of DDR5 Tester Linux Target with ethernet configured:

```
make build TARGET=ddr5_tester_linux TARGET_ARGS="--build --l2-size 256 --iodelay-freq 400e6 --module MTC10F1084S1RC --with-wishbone-memory --wishbone-force-32b --with-ethernet --remote-ip-address 192.168.100.100 --local-ip-address 192.168.100.50"
```

10.2.5 Interacting with DDR5 Tester Linux Target

First, load the bitstream onto the DDR5 Tester with the help of the OpenFPGALoader:

```
openFPGALoader --board antmicro_ddr5_tester build/ddr5_tester_linux/gateware/
↪antmicro_ddr5_tester.bit --freq 3e6
```

In order to connect to the board, assign IP Address – 192.168.100.100 – to the ethernet interface that is plugged to the DDR5 Tester board and set up the device if needed. For example by:

```
ip addr add 192.168.100.100/24 dev $ETH
ip link set dev $ETH up
```

Where **ETH** is the name of your ethernet interface.

When the ethernet interface has been set up correctly, you may access the BIOS console on the DDR5 Tester with:

```
picocom -b 115200 /dev/ttyUSB2
```

10.2.6 Setting up TFTP Server

From this point, several linux boot methods can be invoked but booting via ethernet is recommended. In order to enable netboot, the TFTP server needs to be set up first.

Note: Running TFTP server varies between distributions, this can mean different TFTP implementation name and a different location of the configuration file.

As an example, here's a quick guide on how to configure TFTP server for Arch Linux:

Firstly, if not equipped already, get an implementation of a TFTP server, for example:

```
pacman -S tftp-hpa
```

The TFTP server is configured via a `/etc/conf.d/tftpd` file, here's a suggested configuration for the DDR5 Tester Linux boot process:

```
TFTP_USERNAME="tftp"
TFTPD_OPTIONS="--secure"
TFTP_DIRECTORY="/srv/tftp"
TFTP_ADDRESS="192.168.100.100:69"
```

The **TFTP_ADDRESS** is the specified `--remote-ip-address` whilst building the target and the port is the default one for the TFTP server. The **TFTP_DIRECTORY** is the TFTP's root directory.

Then, the TFTP service needs to be started:

```
systemctl start tftpd
```

To check if the TFTP sever has been set up properly, one may run:

```
cd /srv/tftp/ && echo "TEST TFTP SERVER" > test
cd ~/ && tftp 192.168.100.100 -c get test
```

The **test** file should appear in the home directory with “TEST TFTP SERVER” contents.

10.2.7 Booting Linux on DDR5 Tester Linux Target

You will need the following binaries:

- Linux kernel Image
- Compiled devicetree
- Opensbi’s fw_jump.bin
- rootfs.cpio

All of those can be obtained with the use of provided buildroot_ext buildroot external configuration. To build binaries with buildroot run:

```
git clone --single-branch -b 2023.05.x https://github.com/buildroot/buildroot.git
pushd buildroot
make BR2_EXTERNAL="$(pwd)/../buildroot_ext" ddr5_vexriscv_defconfig
```

Then, transfer the binaries to the TFTP root directory:

```
mv buildroot/output/images/* /srv/tftp/
mv /srv/tftp/fw_jump.bin /srv/tftp/opensbi.bin
```

The address map of the binaries alongside boot args can be contained within the **boot.json** file, for example:

```
{
  "/srv/tftp/Image":      "0x40000000",
  "/srv/tftp/rv32.dtb":   "0x40ef0000",
  "/srv/tftp/rootfs.cpio": "0x42000000",
  "/srv/tftp/opensbi.bin": "0x40f00000",
  "bootargs": {
    "r1":    "0x00000000",
    "r2":    "0x40ef0000",
    "r3":    "0x00000000",
    "addr":  "0x40f00000"
  }
}
```

Having placed linux boot binaries in the TFTP’s root directory with **boot.json**, the netboot can be invoked from BIOS console with:

```
netboot /srv/tftp/boot.json
```

Upon successful execution a similar log will be printed:

```

litex> netboot /srv/tftp/boot.json
Booting from network...
Local IP: 192.168.100.50
Remote IP: 192.168.100.100
Booting from /srv/tftp/boot.json (JSON)...
Copying /srv/tftp/Image to 0x40000000... (7395804 bytes)
Copying /srv/tftp/rv32.dtb to 0x40ef0000... (2463 bytes)
Copying /srv/tftp/rootfs.cpio to 0x42000000... (22128128 bytes)
Copying /srv/tftp/opensbi.bin to 0x40f00000... (1007056 bytes)
Executing booted program at 0x40f00000

===== Liftoff! =====

```

Then, the Opensbi and Linux boot log should follow:

```

OpenSBI v1.3-24-g84c6dc1

      _____
     /  _ \   |  / _ \  |  / _ \  |  / _ \  |
    | | | | | - - |  _ - - |  _ - - |  _ - - | | | | |
    | | | | | ' \ / _ \ ' \ / _ \ ' \ / _ \ |
    | |--| | |_) |  _/ | | |_) |  _/ | |_) |
\_____| | .-/ \_____| | | |_____| |_____| |_____| |
      | |
      | |

Platform Name          : LiteX / VexRiscv-SMP
Platform Features       : medeleg
Platform HART Count    : 8
Platform IPI Device     : aclint-mswi
Platform Timer Device   : aclint-mtimer @ 100000000Hz
Platform Console Device : litex_uart
(...)
[    0.000000] Linux version 5.11.0 (riscv32-buildroot-linux-gnu-gcc.br_real_
↳(Buildroot 2023.05.2-154-g787a633711) 11.4.0, GNU ld (GNU Binutils) 2.38) #2_
↳SMP Mon Sep 25 10:52:22 CEST 2023
[    0.000000] earlycon: sbi0 at I/O port 0x0 (options '')
[    0.000000] printk: bootconsole [sbi0] enabled
(...)
```

At the very end you should be greeted with:

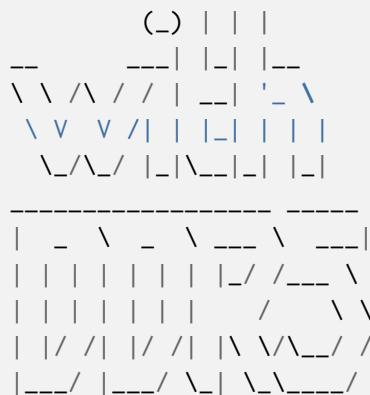
```

Welcome to Buildroot
buildroot login: root

      _____
     /  _ \   |  / _ \  |  / _ \  |  / _ \  |
    | | | | | - - |  _ - - |  _ - - |  _ - - | | | | |
    | | | | | ' \ / _ \ ' \ / _ \ ' \ / _ \ |
    | |--| | |_) |  _/ | | |_) |  _/ | |_) |
\_____| | .-/ \_____| | | |_____| |_____| |_____| |
      | |
      | |
```

(continues on next page)

(continued from previous page)



32-bit RISC-V Linux running on DDR5 Tester.

login[65]: root login on 'console'

10.3 Simulation

The simulation is based on a DDR5 DRAM model (`sdram_simulation_model.py`) and a DDR5 PHY simulation model (`simphy.py`). These two models are used by the SoC simulation model (`simsoc.py`).

The simulation can be started with:

```
python3 third_party/litedram/litedram/phy/ddr5/simsoc.py --no-masked-write --with-
↪sub-channels --dq-dqs-ratio 4 --modules-in-rank 1 --log-level error --skip-csca_
↪--skip-reset-seq --skip-mrs-seq --with-prompt --l2-size 256 --uart-name serial
```

CHAPTER
ELEVEN

DDR5 TEST BOARD



The DDR5 test board is an open source hardware test platform that enables testing and experimenting with various x4/x8 DDR5 modules embedded on DDR5 testbed.

The hardware is open and can be found on GitHub:

- Main board <https://github.com/antmicro/lpddr4-test-board/>
- Testbed <https://github.com/antmicro/ddr5-testbed/>

The following instructions explain how to set up the board.

11.1 Board configuration

First connect the board USB and Ethernet cables to your computer, plug the board to the socket and turn it on using power switch. Then configure the network. The board's IP address will be 192.168.100.50 (so you could e.g. use 192.168.100.2/24). The IP_ADDRESS environment variable can be used to modify the board's address. Next, generate the FPGA bitstream:

```
export TARGET=ddr5_test_board  
make build TARGET_ARGS="--l2-size 256 --build --iodelay-clk-freq 400e6 --bios-lto  
--rw-bios --no-sdram-hw-test"
```

Note: `--l2-size 256` sets L2 cache size to 256 bytes

`--no-sdram-hw-test` disables hw accelerated memory test

Note: By typing `make` (without `build`) LiteX will generate build files without invoking Vivado.

The results will be located in: `build/ddr5_test_board/gateware/antmicro_ddr5_test_board.bit`. To upload it, use:

```
export TARGET=ddr5_test_board  
make upload
```

To save bitstream in flash memory, use:

```
export TARGET=ddr5_test_board  
make flash
```

Warning: There is a JTAG/SPI jumper named MODE2 on the right side of the board. Unless it's set to the SPI setting, the FPGA will load the bitstream received via JTAG.

Bitstream will be loaded from flash memory upon device power-on or after a PROG button press.

CHAPTER
TWELVE

DOCUMENTATION FOR ROW HAMMER TESTER ARTY-A7

12.1 Modules

12.1.1 Interrupt Controller

This device has an EventManager-based interrupt system. Individual modules generate *events* which are wired into a central interrupt controller.

When an interrupt occurs, you should look the interrupt number up in the CPU- specific interrupt table and then call the relevant module.

Assigned Interrupts

The following interrupts are assigned on this system:

Interrupt	Module
1	<i>TIMER0</i>
0	<i>UART</i>

12.2 Register Groups

12.2.1 LEDs

Register Listing for LEDs

Register	Address
<i>LEDS_OUT</i>	<i>0xf0000000</i>

LEDS_OUT

Address: $0xf0000000 + 0x0 = 0xf0000000$

Led Output(s) Control.

12.2.2 DDRPHY

Register Listing for DDRPHY

Register	Address
<i>DDRPHY_RST</i>	<i>0xf0000800</i>
<i>DDRPHY_DLY_SEL</i>	<i>0xf0000804</i>
<i>DDRPHY_HALF_SYS8X_TAPS</i>	<i>0xf0000808</i>
<i>DDRPHY_WLEVEL_EN</i>	<i>0xf000080c</i>
<i>DDRPHY_WLEVEL_STROBE</i>	<i>0xf0000810</i>
<i>DDRPHY_RDLY_DQ_RST</i>	<i>0xf0000814</i>
<i>DDRPHY_RDLY_DQ_INC</i>	<i>0xf0000818</i>
<i>DDRPHY_RDLY_DQ_BITSLIP_RST</i>	<i>0xf000081c</i>
<i>DDRPHY_RDLY_DQ_BITSLIP</i>	<i>0xf0000820</i>
<i>DDRPHY_WDLY_DQ_BITSLIP_RST</i>	<i>0xf0000824</i>
<i>DDRPHY_WDLY_DQ_BITSLIP</i>	<i>0xf0000828</i>
<i>DDRPHY_RDPHASE</i>	<i>0xf000082c</i>
<i>DDRPHY_WRPHASE</i>	<i>0xf0000830</i>

DDRPHY_RST

Address: $0xf0000800 + 0x0 = 0xf0000800$

DDRPHY_DLY_SEL

Address: $0xf0000800 + 0x4 = 0xf0000804$

DDRPHY_HALF_SYS8X_TAPS

Address: $0xf0000800 + 0x8 = 0xf0000808$

DDRPHY_WLEVEL_EN

Address: $0xf0000800 + 0xc = 0xf000080c$

DDRPHY_WLEVEL_STROBE

Address: $0xf0000800 + 0x10 = 0xf0000810$

DDRPHY_RDLY_DQ_RST

Address: $0xf0000800 + 0x14 = 0xf0000814$

DDRPHY_RDLY_DQ_INC

Address: $0xf0000800 + 0x18 = 0xf0000818$

DDRPHY_RDLY_DQ_BITSLIP_RST

Address: $0xf0000800 + 0x1c = 0xf000081c$

DDRPHY_RDLY_DQ_BITSLIP

Address: $0xf0000800 + 0x20 = 0xf0000820$

DDRPHY_WDLY_DQ_BITSLIP_RST

Address: $0xf0000800 + 0x24 = 0xf0000824$

DDRPHY_WDLY_DQ_BITSLIP

Address: $0xf0000800 + 0x28 = 0xf0000828$

DDRPHY_RDPHASE

Address: $0xf0000800 + 0x2c = 0xf000082c$

DDRPHY_WRPHASE

Address: $0xf0000800 + 0x30 = 0xf0000830$

12.2.3 CONTROLLER_SETTINGS

Allows to change LiteDRAMController behaviour at runtime

Register Listing for CONTROLLER_SETTINGS

Register	Address
<i>CONTROLLER_SETTINGS_REFRESH</i>	<i>0xf0001000</i>

CONTROLLER_SETTINGS_REFRESH

Address: $0xf0001000 + 0x0 = 0xf0001000$

Enable/disable Refresh commands sending

12.2.4 DDRCTRL

Register Listing for DDRCTRL

Register	Address
<i>DDRCTRL_INIT_DONE</i>	<i>0xf0001800</i>
<i>DDRCTRL_INIT_ERROR</i>	<i>0xf0001804</i>

DDRCTRL_INIT_DONE

Address: $0xf0001800 + 0x0 = 0xf0001800$

DDRCTRL_INIT_ERROR

Address: $0xf0001800 + 0x4 = 0xf0001804$

12.2.5 ETHPHY

Register Listing for ETHPHY

Register	Address
<i>ETHPHY_CRG_RESET</i>	<i>0xf0002000</i>
<i>ETHPHY_MDIO_W</i>	<i>0xf0002004</i>
<i>ETHPHY_MDIO_R</i>	<i>0xf0002008</i>

ETHPHY_CRG_RESET

Address: $0xf0002000 + 0x0 = 0xf0002000$

ETHPHY_MDIO_W

Address: $0xf0002000 + 0x4 = 0xf0002004$

Field	Name	Description

ETHPHY_MDIO_R

Address: $0xf0002000 + 0x8 = 0xf0002008$

Field	Name	Description

12.2.6 ROWHAMMER

Row Hammer DMA attacker

This module allows to perform a Row Hammer attack by configuring it with two addresses that map to different rows of a single bank. When enabled, it will perform alternating DMA reads from the given locations, which will result in the DRAM controller having to repeatedly open/close rows at each read access.

Register Listing for ROWHAMMER

Register	Address
<i>ROWHAMMER_ENABLED</i>	<i>0xf0002800</i>
<i>ROWHAMMER_ADDRESS1</i>	<i>0xf0002804</i>
<i>ROWHAMMER_ADDRESS2</i>	<i>0xf0002808</i>
<i>ROWHAMMER_COUNT</i>	<i>0xf000280c</i>

ROWHAMMER_ENABLED

Address: $0xf0002800 + 0x0 = 0xf0002800$

Used to start/stop the operation of the module

ROWHAMMER_ADDRESS1

Address: $0xf0002800 + 0x4 = 0xf0002804$

First attacked address

ROWHAMMER_ADDRESS2

Address: $0xf0002800 + 0x8 = 0xf0002808$

Second attacked address

ROWHAMMER_COUNT

Address: $0xf0002800 + 0xc = 0xf000280c$

This is the number of DMA accesses performed. When the module is enabled, the value can be freely read. When the module is disabled, the register is clear-on-write and has to be read before the next attack.

12.2.7 WRITER

DMA DRAM writer.

Allows to fill DRAM with a predefined pattern using DMA.

Pattern

Provides access to RAM to store access pattern: *mem_addr* and *mem_data*. The pattern address space can be limited using the *data_mask*.

For example, having *mem_addr* filled with [0x04, 0x02, 0x03, ...] and *mem_data* filled with [0xff, 0xaa, 0x55, ...] and setting *data_mask* = 0b01, the pattern [(address, data), ...] written will be: [(0x04, 0xff), (0x02, 0xaa), (0x04, 0xff), ...] (wraps due to masking).

DRAM memory range that is being accessed can be configured using *mem_mask*.

To use this module, make sure that *ready* is 1, then write the desired number of transfers to *count*. Writing to the *start* CSR will initialize the operation. When the operation is ongoing *ready* will be 0.

Register Listing for WRITER

Register	Address
WRITER_START	0xf0003000
WRITER_READY	0xf0003004
WRITER_MODULO	0xf0003008
WRITER_COUNT	0xf000300c
WRITER_DONE	0xf0003010
WRITER_MEM_MASK	0xf0003014
WRITER_DATA_MASK	0xf0003018
WRITER_DATA_DIV	0xf000301c
WRITER_INVERTER_DIVISOR_MASK	0xf0003020
WRITER_INVERTER_SELECTION_MASK	0xf0003024
WRITER_LAST_ADDRESS	0xf0003028

WRITER_START

Address: $0xf0003000 + 0x0 = 0xf0003000$

Write to the register starts the transfer (if ready=1)

WRITER_READY

Address: $0xf0003000 + 0x4 = 0xf0003004$

Indicates that the transfer is not ongoing

WRITER_MODULO

Address: $0xf0003000 + 0x8 = 0xf0003008$

When set use modulo to calculate DMA transfers address rather than bit masking

WRITER_COUNT

Address: $0xf0003000 + 0xc = 0xf000300c$

Desired number of DMA transfers

WRITER_DONE

Address: $0xf0003000 + 0x10 = 0xf0003010$

Number of completed DMA transfers

WRITER_MEM_MASK

Address: $0xf0003000 + 0x14 = 0xf0003014$

DRAM address mask for DMA transfers

WRITER_DATA_MASK

Address: $0xf0003000 + 0x18 = 0xf0003018$

Pattern memory address mask

WRITER_DATA_DIV

Address: $0xf0003000 + 0x1c = 0xf000301c$

Pattern memory address divisor-1

WRITER_INVERTER_DIVISOR_MASK

Address: $0xf0003000 + 0x20 = 0xf0003020$

Divisor mask for selecting rows for which pattern data gets inverted

WRITER_INVERTER_SELECTION_MASK

Address: $0xf0003000 + 0x24 = 0xf0003024$

Selection mask for selecting rows for which pattern data gets inverted

WRITER_LAST_ADDRESS

Address: $0xf0003000 + 0x28 = 0xf0003028$

Number of completed DMA transfers

12.2.8 READER

DMA DRAM reader.

Allows to check DRAM contents against a predefined pattern using DMA.

Pattern

Provides access to RAM to store access pattern: *mem_addr* and *mem_data*. The pattern address space can be limited using the *data_mask*.

For example, having *mem_addr* filled with [0x04, 0x02, 0x03, ...] and *mem_data* filled with [0xff, 0xaa, 0x55, ...] and setting *data_mask* = 0b01, the pattern [(address, data), ...] written will be: [(0x04, 0xff), (0x02, 0xaa), (0x04, 0xff), ...] (wraps due to masking).

DRAM memory range that is being accessed can be configured using *mem_mask*.

To use this module, make sure that *ready* is 1, then write the desired number of transfers to *count*. Writing to the *start* CSR will initialize the operation. When the operation is ongoing *ready* will be 0.

Reading errors

This module allows to check the locations of errors in the memory. It scans the configured memory area and compares the values read to the predefined pattern. If *skip_fifo* is 0, this module will stop after each error encountered, so that it can be examined. Wait until the *error_ready* CSR is 1. Then use the CSRs *error_offset*, *error_data* and *error_expected* to examine the errors in the current transfer. To continue reading, write 1 to *error_continue* CSR. Setting *skip_fifo* to 1 will disable this behaviour entirely.

The final number of errors can be read from *error_count*. NOTE: This value represents the number of erroneous *DMA transfers*.

The current progress can be read from the *done* CSR.

Register Listing for READER

Register	Address
<i>READER_START</i>	0xf0003800
<i>READER_READY</i>	0xf0003804
<i>READER_MODULO</i>	0xf0003808
<i>READER_COUNT</i>	0xf000380c
<i>READER_DONE</i>	0xf0003810
<i>READER_MEM_MASK</i>	0xf0003814
<i>READER_DATA_MASK</i>	0xf0003818
<i>READER_DATA_DIV</i>	0xf000381c
<i>READER_INVERTER_DIVISOR_MASK</i>	0xf0003820
<i>READER_INVERTER_SELECTION_MASK</i>	0xf0003824
<i>READER_ERROR_COUNT</i>	0xf0003828
<i>READER_SKIP_FIFO</i>	0xf000382c
<i>READER_ERROR_OFFSET</i>	0xf0003830
<i>READER_ERROR_DATA3</i>	0xf0003834
<i>READER_ERROR_DATA2</i>	0xf0003838
<i>READER_ERROR_DATA1</i>	0xf000383c
<i>READER_ERROR_DATA0</i>	0xf0003840
<i>READER_ERROR_EXPECTED3</i>	0xf0003844
<i>READER_ERROR_EXPECTED2</i>	0xf0003848
<i>READER_ERROR_EXPECTED1</i>	0xf000384c
<i>READER_ERROR_EXPECTED0</i>	0xf0003850
<i>READER_ERROR_READY</i>	0xf0003854
<i>READER_ERROR_CONTINUE</i>	0xf0003858

READER_START

Address: $0xf0003800 + 0x0 = 0xf0003800$

Write to the register starts the transfer (if ready=1)

READER_READY

Address: $0xf0003800 + 0x4 = 0xf0003804$

Indicates that the transfer is not ongoing

READER_MODULO

Address: $0xf0003800 + 0x8 = 0xf0003808$

When set use modulo to calculate DMA transfers address rather than bit masking

READER_COUNT

Address: $0xf0003800 + 0xc = 0xf000380c$

Desired number of DMA transfers

READER_DONE

Address: $0xf0003800 + 0x10 = 0xf0003810$

Number of completed DMA transfers

READER_MEM_MASK

Address: $0xf0003800 + 0x14 = 0xf0003814$

DRAM address mask for DMA transfers

READER_DATA_MASK

Address: $0xf0003800 + 0x18 = 0xf0003818$

Pattern memory address mask

READER_DATA_DIV

Address: $0xf0003800 + 0x1c = 0xf000381c$

Pattern memory address divisor-1

READER_INVERTER_DIVISOR_MASK

Address: $0xf0003800 + 0x20 = 0xf0003820$

Divisor mask for selecting rows for which pattern data gets inverted

READER_INVERTER_SELECTION_MASK

Address: $0xf0003800 + 0x24 = 0xf0003824$

Selection mask for selecting rows for which pattern data gets inverted

READER_ERROR_COUNT

Address: $0xf0003800 + 0x28 = 0xf0003828$

Number of errors detected

READER_SKIP_FIFO

Address: $0xf0003800 + 0x2c = 0xf000382c$

Skip waiting for user to read the errors FIFO

READER_ERROR_OFFSET

Address: $0xf0003800 + 0x30 = 0xf0003830$

Current offset of the error

READER_ERROR_DATA3

Address: $0xf0003800 + 0x34 = 0xf0003834$

Bits 96-127 of *READER_ERROR_DATA*. Erroneous value read from DRAM memory

READER_ERROR_DATA2

Address: $0xf0003800 + 0x38 = 0xf0003838$

Bits 64-95 of *READER_ERROR_DATA*.

READER_ERROR_DATA1

Address: $0xf0003800 + 0x3c = 0xf000383c$

Bits 32-63 of *READER_ERROR_DATA*.

READER_ERROR_DATA0

Address: $0xf0003800 + 0x40 = 0xf0003840$

Bits 0-31 of *READER_ERROR_DATA*.

READER_ERROR_EXPECTED3

Address: $0xf0003800 + 0x44 = 0xf0003844$

Bits 96-127 of *READER_ERROR_EXPECTED*. Value expected to be read from DRAM memory

READER_ERROR_EXPECTED2

Address: $0xf0003800 + 0x48 = 0xf0003848$

Bits 64-95 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED1

Address: $0xf0003800 + 0x4c = 0xf000384c$

Bits 32-63 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED0

Address: $0xf0003800 + 0x50 = 0xf0003850$

Bits 0-31 of *READER_ERROR_EXPECTED*.

READER_ERROR_READY

Address: $0xf0003800 + 0x54 = 0xf0003854$

Error detected and ready to read

READER_ERROR_CONTINUE

Address: $0xf0003800 + 0x58 = 0xf0003858$

Continue reading until the next error

12.2.9 DFI_SWITCH

Register Listing for DFI_SWITCH

Register	Address
<i>DFI_SWITCH_REFRESH_COUNT</i>	<i>0xf0004000</i>
<i>DFI_SWITCH_AT_REFRESH</i>	<i>0xf0004004</i>
<i>DFI_SWITCH_REFRESH_UPDATE</i>	<i>0xf0004008</i>

DFI_SWITCH_REFRESH_COUNT

Address: $0xf0004000 + 0x0 = 0xf0004000$

Count of all refresh commands issued (both by Memory Controller and Payload Executor). Value is latched from internal counter on mode transition: MC -> PE or by writing to the *refresh_update* CSR.

DFI_SWITCH_AT_REFRESH

Address: $0xf0004000 + 0x4 = 0xf0004004$

If set to a value different than 0 the mode transition MC -> PE will be performed only when the value of this register matches the current refresh commands count.

DFI_SWITCH_REFRESH_UPDATE

Address: $0xf0004000 + 0x8 = 0xf0004008$

Force an update of the *refresh_count* CSR.

12.2.10 PAYLOAD_EXECUTOR

Executes the DRAM payload from memory

Instruction decoder

All instructions are 32-bit. The format of most instructions is the same, except for the LOOP instruction, which has a constant TIMESLICE of 1.

NOOP with a TIMESLICE of 0 is a special case which is interpreted as STOP instruction. When this instruction is encountered execution gets finished immediately.

NOTE: TIMESLICE is the number of cycles the instruction will take. This means that instructions other than NOOP that use TIMESLICE=0 are illegal (although will silently be executed as having TIMESLICE=1).

NOTE2: LOOP instruction will *jump* COUNT times, meaning that the “code” inside the loop will effectively be executed COUNT+1 times.

Op codes:

Op	Value
NOOP	0b000
LOOP	0b111
ACT	0b100
PRE	0b101
REF	0b110
ZQC	0b001
READ	0b010

Instruction format:

LSB	MSB
dfi: OP_CODE TIMESLICE ADDRESS	
noop: OP_CODE TIMESLICE_NOOP	
loop: OP_CODE COUNT JUMP	
stop: <NOOP> 0	

Where ADDRESS depends on the DFI command and is one of:

LSB	MSB
RANK BANK COLUMN	
RANK BANK ROW	

Register Listing for PAYLOAD_EXECUTOR

Register	Address
PAYLOAD_EXECUTOR_START	0xf0004800
PAYLOAD_EXECUTOR_STATUS	0xf0004804
PAYLOAD_EXECUTOR_READ_COUNT	0xf0004808

PAYLOAD_EXECUTOR_START

Address: $0xf0004800 + 0x0 = 0xf0004800$

Writing to this register initializes payload execution

PAYLOAD_EXECUTOR_STATUS

Address: $0xf0004800 + 0x4 = 0xf0004804$

Payload executor status register

Field	Name	Description
[0]	READY	Indicates that the executor is not running
[1]	OVERFLOW	Indicates the scratchpad memory address counter has overflowed due to the number of READ commands sent during execution

PAYLOAD_EXECUTOR_READ_COUNT

Address: $0xf0004800 + 0x8 = 0xf0004808$

Number of data from READ commands that is stored in the scratchpad memory

12.2.11 CTRL

Register Listing for CTRL

Register	Address
CTRL_RESET	0xf0005000
CTRL_SCRATCH	0xf0005004
CTRL_BUS_ERRORS	0xf0005008

CTRL__RESET

Address: $0xf0005000 + 0x0 = 0xf0005000$

Field	Name	Description
[0]	SOC_RST	Write 1 to this register to reset the full SoC (Pulse Reset)
[1]	CPU_RST	Write 1 to this register to reset the CPU(s) of the SoC (Hold Reset)

CTRL_SCRATCH

Address: $0xf0005000 + 0x4 = 0xf0005004$

Use this register as a scratch space to verify that software read/write accesses to the Wishbone/CSR bus are working correctly. The initial reset value of 0x1234578 can be used to verify endianness.

CTRL_BUS_ERRORS

Address: $0xf0005000 + 0x8 = 0xf0005008$

Total number of Wishbone bus errors (timeouts) since start.

12.2.12 IDENTIFIER_MEM

Register Listing for IDENTIFIER_MEM

Register	Address
<i>IDENTIFIER_MEM</i>	<i>0xf0005800</i>

IDENTIFIER_MEM

Address: $0xf0005800 + 0x0 = 0xf0005800$

8 x 110-bit memory

12.2.13 SDRAM

Register Listing for SDRAM

Register	Address
<i>SDRAM_DFII_CONTROL</i>	0xf0006000
<i>SDRAM_DFII_PIO_COMMAND</i>	0xf0006004
<i>SDRAM_DFII_PIO_COMMAND_ISSUE</i>	0xf0006008
<i>SDRAM_DFII_PIO_ADDRESS</i>	0xf000600c
<i>SDRAM_DFII_PIO_BADDRESS</i>	0xf0006010
<i>SDRAM_DFII_PIO_WRDATA</i>	0xf0006014
<i>SDRAM_DFII_PIO_RDDATA</i>	0xf0006018
<i>SDRAM_DFII_PI1_COMMAND</i>	0xf000601c
<i>SDRAM_DFII_PI1_COMMAND_ISSUE</i>	0xf0006020
<i>SDRAM_DFII_PI1_ADDRESS</i>	0xf0006024
<i>SDRAM_DFII_PI1_BADDRESS</i>	0xf0006028
<i>SDRAM_DFII_PI1_WRDATA</i>	0xf000602c
<i>SDRAM_DFII_PI1_RDDATA</i>	0xf0006030
<i>SDRAM_DFII_PI2_COMMAND</i>	0xf0006034
<i>SDRAM_DFII_PI2_COMMAND_ISSUE</i>	0xf0006038
<i>SDRAM_DFII_PI2_ADDRESS</i>	0xf000603c
<i>SDRAM_DFII_PI2_BADDRESS</i>	0xf0006040
<i>SDRAM_DFII_PI2_WRDATA</i>	0xf0006044
<i>SDRAM_DFII_PI2_RDDATA</i>	0xf0006048
<i>SDRAM_DFII_PI3_COMMAND</i>	0xf000604c
<i>SDRAM_DFII_PI3_COMMAND_ISSUE</i>	0xf0006050
<i>SDRAM_DFII_PI3_ADDRESS</i>	0xf0006054
<i>SDRAM_DFII_PI3_BADDRESS</i>	0xf0006058
<i>SDRAM_DFII_PI3_WRDATA</i>	0xf000605c
<i>SDRAM_DFII_PI3_RDDATA</i>	0xf0006060
<i>SDRAM_CONTROLLER_TRP</i>	0xf0006064
<i>SDRAM_CONTROLLER_TRCD</i>	0xf0006068
<i>SDRAM_CONTROLLER_TWR</i>	0xf000606c
<i>SDRAM_CONTROLLER_TWTR</i>	0xf0006070
<i>SDRAM_CONTROLLER_TREFI</i>	0xf0006074
<i>SDRAM_CONTROLLER_TRFC</i>	0xf0006078
<i>SDRAM_CONTROLLER_TFAW</i>	0xf000607c
<i>SDRAM_CONTROLLER_TCCD</i>	0xf0006080
<i>SDRAM_CONTROLLER_TCCD_WR</i>	0xf0006084
<i>SDRAM_CONTROLLER_TRTP</i>	0xf0006088
<i>SDRAM_CONTROLLER_TRRD</i>	0xf000608c
<i>SDRAM_CONTROLLER_TRC</i>	0xf0006090
<i>SDRAM_CONTROLLER_TRAS</i>	0xf0006094
<i>SDRAM_CONTROLLER_TZQCS</i>	0xf0006098
<i>SDRAM_CONTROLLER_LAST_ADDR_0</i>	0xf000609c
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_0</i>	0xf00060a0
<i>SDRAM_CONTROLLER_LAST_ADDR_1</i>	0xf00060a4
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_1</i>	0xf00060a8
<i>SDRAM_CONTROLLER_LAST_ADDR_2</i>	0xf00060ac

continues on next page

Table 12.1 – continued from previous page

Register	Address
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_2</i>	0xf00060b0
<i>SDRAM_CONTROLLER_LAST_ADDR_3</i>	0xf00060b4
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_3</i>	0xf00060b8
<i>SDRAM_CONTROLLER_LAST_ADDR_4</i>	0xf00060bc
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_4</i>	0xf00060c0
<i>SDRAM_CONTROLLER_LAST_ADDR_5</i>	0xf00060c4
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_5</i>	0xf00060c8
<i>SDRAM_CONTROLLER_LAST_ADDR_6</i>	0xf00060cc
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_6</i>	0xf00060d0
<i>SDRAM_CONTROLLER_LAST_ADDR_7</i>	0xf00060d4
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_7</i>	0xf00060d8

SDRAM_DFII_CONTROL

Address: 0xf0006000 + 0x0 = 0xf0006000

Control DFI signals common to all phases

Field	Name	Description							
[0]	SEL	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>0b0</td> <td>Software control. (CPU)</td> </tr> <tr> <td>0b1</td> <td>Hardware control (default).</td> </tr> </tbody> </table>		Value	Description	0b0	Software control. (CPU)	0b1	Hardware control (default).
Value	Description								
0b0	Software control. (CPU)								
0b1	Hardware control (default).								
[1]	CKE	DFI clock enable bus							
[2]	ODT	DFI on-die termination bus							
[3]	RESET_N	DFI clock reset bus							

SDRAM_DFII_PIO_COMMAND

Address: 0xf0006000 + 0x4 = 0xf0006004

Control DFI signals on a single phase

Field	Name	Description
[0]	CS	DFI chip select bus
[1]	WE	DFI write enable bus
[2]	CAS	DFI column address strobe bus
[3]	RAS	DFI row address strobe bus
[4]	WREN	DFI write data enable bus
[5]	RDEN	DFI read data enable bus

SDRAM_DFIIPIO_COMMAND_ISSUE

Address: $0xf0006000 + 0x8 = 0xf0006008$

SDRAM_DFIIPIO_ADDRESS

Address: $0xf0006000 + 0xc = 0xf000600c$

DFI address bus

SDRAM_DFIIPIO_BADDRESS

Address: $0xf0006000 + 0x10 = 0xf0006010$

DFI bank address bus

SDRAM_DFIIPIO_WRDATA

Address: $0xf0006000 + 0x14 = 0xf0006014$

DFI write data bus

SDRAM_DFIIPIO_RDDATA

Address: $0xf0006000 + 0x18 = 0xf0006018$

DFI read data bus

SDRAM_DFIIPIO_COMMAND

Address: $0xf0006000 + 0x1c = 0xf000601c$

Control DFI signals on a single phase

Field	Name	Description
[0]	CS	DFI chip select bus
[1]	WE	DFI write enable bus
[2]	CAS	DFI column address strobe bus
[3]	RAS	DFI row address strobe bus
[4]	WREN	DFI write data enable bus
[5]	RDEN	DFI read data enable bus

SDRAM_DFII_PI1_COMMAND_ISSUE

Address: 0xf0006000 + 0x20 = 0xf0006020

SDRAM_DFII_PI1_ADDRESS

Address: 0xf0006000 + 0x24 = 0xf0006024

DFI address bus

SDRAM_DFII_PI1_BADDRESS

Address: 0xf0006000 + 0x28 = 0xf0006028

DFI bank address bus

SDRAM_DFII_PI1_WRDATA

Address: 0xf0006000 + 0x2c = 0xf000602c

DFI write data bus

SDRAM_DFII_PI1_RDDATA

Address: 0xf0006000 + 0x30 = 0xf0006030

DFI read data bus

SDRAM_DFII_PI2_COMMAND

Address: 0xf0006000 + 0x34 = 0xf0006034

Control DFI signals on a single phase

Field	Name	Description
[0]	CS	DFI chip select bus
[1]	WE	DFI write enable bus
[2]	CAS	DFI column address strobe bus
[3]	RAS	DFI row address strobe bus
[4]	WREN	DFI write data enable bus
[5]	RDEN	DFI read data enable bus

SDRAM_DFII_PI2_COMMAND_ISSUE

Address: 0xf0006000 + 0x38 = 0xf0006038

SDRAM_DFII_PI2_ADDRESS

Address: 0xf0006000 + 0x3c = 0xf000603c

DFI address bus

SDRAM_DFII_PI2_BADDRESS

Address: 0xf0006000 + 0x40 = 0xf0006040

DFI bank address bus

SDRAM_DFII_PI2_WRDATA

Address: 0xf0006000 + 0x44 = 0xf0006044

DFI write data bus

SDRAM_DFII_PI2_RDDATA

Address: 0xf0006000 + 0x48 = 0xf0006048

DFI read data bus

SDRAM_DFII_PI3_COMMAND

Address: 0xf0006000 + 0x4c = 0xf000604c

Control DFI signals on a single phase

Field	Name	Description
[0]	CS	DFI chip select bus
[1]	WE	DFI write enable bus
[2]	CAS	DFI column address strobe bus
[3]	RAS	DFI row address strobe bus
[4]	WREN	DFI write data enable bus
[5]	RDEN	DFI read data enable bus

SDRAM_DFII_PI3_COMMAND_ISSUE

Address: 0xf0006000 + 0x50 = 0xf0006050

SDRAM_DFII_PI3_ADDRESS

Address: 0xf0006000 + 0x54 = 0xf0006054

DFI address bus

SDRAM_DFII_PI3_BADDRESS

Address: 0xf0006000 + 0x58 = 0xf0006058

DFI bank address bus

SDRAM_DFII_PI3_WRDATA

Address: 0xf0006000 + 0x5c = 0xf000605c

DFI write data bus

SDRAM_DFII_PI3_RDDATA

Address: 0xf0006000 + 0x60 = 0xf0006060

DFI read data bus

SDRAM_CONTROLLER_TRP

Address: 0xf0006000 + 0x64 = 0xf0006064

SDRAM_CONTROLLER_TRCD

Address: 0xf0006000 + 0x68 = 0xf0006068

SDRAM_CONTROLLER_TWR

Address: 0xf0006000 + 0x6c = 0xf000606c

SDRAM_CONTROLLER_TWTR

Address: 0xf0006000 + 0x70 = 0xf0006070

SDRAM_CONTROLLER_TREFI

Address: 0xf0006000 + 0x74 = 0xf0006074

SDRAM_CONTROLLER_TRFC

Address: 0xf0006000 + 0x78 = 0xf0006078

SDRAM_CONTROLLER_TFAW

Address: 0xf0006000 + 0x7c = 0xf000607c

SDRAM_CONTROLLER_TCCD

Address: 0xf0006000 + 0x80 = 0xf0006080

SDRAM_CONTROLLER_TCCD_WR

Address: 0xf0006000 + 0x84 = 0xf0006084

SDRAM_CONTROLLER_TRTP

Address: $0xf0006000 + 0x88 = 0xf0006088$

SDRAM_CONTROLLER_TRRD

Address: $0xf0006000 + 0x8c = 0xf000608c$

SDRAM_CONTROLLER_TRC

Address: $0xf0006000 + 0x90 = 0xf0006090$

SDRAM_CONTROLLER_TRAS

Address: $0xf0006000 + 0x94 = 0xf0006094$

SDRAM_CONTROLLER_TZQCS

Address: $0xf0006000 + 0x98 = 0xf0006098$

SDRAM_CONTROLLER_LAST_ADDR_0

Address: $0xf0006000 + 0x9c = 0xf000609c$

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_0

Address: $0xf0006000 + 0xa0 = 0xf00060a0$

SDRAM_CONTROLLER_LAST_ADDR_1

Address: 0xf0006000 + 0xa4 = 0xf00060a4

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_1

Address: 0xf0006000 + 0xa8 = 0xf00060a8

SDRAM_CONTROLLER_LAST_ADDR_2

Address: 0xf0006000 + 0xac = 0xf00060ac

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_2

Address: 0xf0006000 + 0xb0 = 0xf00060b0

SDRAM_CONTROLLER_LAST_ADDR_3

Address: 0xf0006000 + 0xb4 = 0xf00060b4

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_3

Address: 0xf0006000 + 0xb8 = 0xf00060b8

SDRAM_CONTROLLER_LAST_ADDR_4

Address: 0xf0006000 + 0xbc = 0xf00060bc

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_4

Address: 0xf0006000 + 0xc0 = 0xf00060c0

SDRAM_CONTROLLER_LAST_ADDR_5

Address: 0xf0006000 + 0xc4 = 0xf00060c4

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_5

Address: 0xf0006000 + 0xc8 = 0xf00060c8

SDRAM_CONTROLLER_LAST_ADDR_6

Address: 0xf0006000 + 0xcc = 0xf00060cc

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_6

Address: 0xf0006000 + 0xd0 = 0xf00060d0

SDRAM_CONTROLLER_LAST_ADDR_7

Address: 0xf0006000 + 0xd4 = 0xf00060d4

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_7

Address: 0xf0006000 + 0xd8 = 0xf00060d8

12.2.14 SDRAM_CHECKER

Register Listing for SDRAM_CHECKER

Register	Address
<i>SDRAM_CHECKER_RESET</i>	0xf0006800
<i>SDRAM_CHECKER_START</i>	0xf0006804
<i>SDRAM_CHECKER_DONE</i>	0xf0006808
<i>SDRAM_CHECKER_BASE</i>	0xf000680c
<i>SDRAM_CHECKER_END</i>	0xf0006810
<i>SDRAM_CHECKER_LENGTH</i>	0xf0006814
<i>SDRAM_CHECKER_RANDOM</i>	0xf0006818
<i>SDRAM_CHECKER_TICKS</i>	0xf000681c
<i>SDRAM_CHECKER_ERRORS</i>	0xf0006820

SDRAM_CHECKER_RESET

Address: $0xf0006800 + 0x0 = 0xf0006800$

SDRAM_CHECKER_START

Address: $0xf0006800 + 0x4 = 0xf0006804$

SDRAM_CHECKER_DONE

Address: $0xf0006800 + 0x8 = 0xf0006808$

SDRAM_CHECKER_BASE

Address: $0xf0006800 + 0xc = 0xf000680c$

SDRAM_CHECKER_END

Address: $0xf0006800 + 0x10 = 0xf0006810$

SDRAM_CHECKER_LENGTH

Address: $0xf0006800 + 0x14 = 0xf0006814$

SDRAM_CHECKER_RANDOM

Address: $0xf0006800 + 0x18 = 0xf0006818$

Field	Name	Description

SDRAM_CHECKER_TICKS

Address: $0xf0006800 + 0x1c = 0xf000681c$

SDRAM_CHECKER_ERRORS

Address: $0xf0006800 + 0x20 = 0xf0006820$

12.2.15 SDRAM_GENERATOR

Register Listing for SDRAM_GENERATOR

Register	Address
<i>SDRAM_GENERATOR_RESET</i>	0xf0007000
<i>SDRAM_GENERATOR_START</i>	0xf0007004
<i>SDRAM_GENERATOR_DONE</i>	0xf0007008
<i>SDRAM_GENERATOR_BASE</i>	0xf000700c
<i>SDRAM_GENERATOR_END</i>	0xf0007010
<i>SDRAM_GENERATOR_LENGTH</i>	0xf0007014
<i>SDRAM_GENERATOR_RANDOM</i>	0xf0007018
<i>SDRAM_GENERATOR_TICKS</i>	0xf000701c

SDRAM_GENERATOR_RESET

Address: $0xf0007000 + 0x0 = 0xf0007000$

SDRAM_GENERATOR_START

Address: $0xf0007000 + 0x4 = 0xf0007004$

SDRAM_GENERATOR_DONE

Address: $0xf0007000 + 0x8 = 0xf0007008$

SDRAM_GENERATOR_BASE

Address: $0xf0007000 + 0xc = 0xf000700c$

SDRAM_GENERATOR_END

Address: $0xf0007000 + 0x10 = 0xf0007010$

SDRAM_GENERATOR_LENGTH

Address: $0xf0007000 + 0x14 = 0xf0007014$

SDRAM_GENERATOR_RANDOM

Address: $0xf0007000 + 0x18 = 0xf0007018$

Field	Name	Description

SDRAM_GENERATOR_TICKS

Address: $0xf0007000 + 0x1c = 0xf000701c$

12.2.16 TIMERO

Timer

Provides a generic Timer core.

The Timer is implemented as a countdown timer that can be used in various modes:

- Polling : Returns current countdown value to software
- One-Shot: Loads itself and stops when value reaches 0
- Periodic: (Re-)Loads itself when value reaches 0

en register allows the user to enable/disable the Timer. When the Timer is enabled, it is automatically loaded with the value of *load* register.

When the Timer reaches 0, it is automatically reloaded with value of *reload* register.

The user can latch the current countdown value by writing to *update_value* register, it will update *value* register with current countdown value.

To use the Timer in One-Shot mode, the user needs to:

- Disable the timer
- Set the load register to the expected duration
- (Re-)Enable the Timer

To use the Timer in Periodic mode, the user needs to:

- Disable the Timer

- Set the load register to 0
- Set the reload register to the expected period
- Enable the Timer

For both modes, the CPU can be advertised by an IRQ that the duration/period has elapsed. (The CPU can also do software polling with `update_value` and `value` to know the elapsed duration)

Register Listing for TIMERO

Register	Address
<code>TIMERO_LOAD</code>	<code>0xf0007800</code>
<code>TIMERO_RELOAD</code>	<code>0xf0007804</code>
<code>TIMERO_EN</code>	<code>0xf0007808</code>
<code>TIMERO_UPDATE_VALUE</code>	<code>0xf000780c</code>
<code>TIMERO_VALUE</code>	<code>0xf0007810</code>
<code>TIMERO_EV_STATUS</code>	<code>0xf0007814</code>
<code>TIMERO_EV_PENDING</code>	<code>0xf0007818</code>
<code>TIMERO_EV_ENABLE</code>	<code>0xf000781c</code>

TIMERO_LOAD

Address: `0xf0007800 + 0x0 = 0xf0007800`

Load value when Timer is (re-)enabled. In One-Shot mode, the value written to this register specifies the Timer's duration in clock cycles.

TIMERO_RELOAD

Address: `0xf0007800 + 0x4 = 0xf0007804`

Reload value when Timer reaches 0. In Periodic mode, the value written to this register specify the Timer's period in clock cycles.

TIMERO_EN

Address: `0xf0007800 + 0x8 = 0xf0007808`

Enable flag of the Timer. Set this flag to 1 to enable/start the Timer. Set to 0 to disable the Timer.

TIMER0_UPDATE_VALUE

Address: $0xf0007800 + 0xc = 0xf000780c$

Update trigger for the current countdown value. A write to this register latches the current countdown value to value register.

TIMER0_VALUE

Address: $0xf0007800 + 0x10 = 0xf0007810$

Latched countdown value. This value is updated by writing to update_value.

TIMER0_EV_STATUS

Address: $0xf0007800 + 0x14 = 0xf0007814$

This register contains the current raw level of the zero event trigger. Writes to this register have no effect.

Field	Name	Description
[0]	ZERO	Level of the zero event

TIMER0_EV_PENDING

Address: $0xf0007800 + 0x18 = 0xf0007818$

When a zero event occurs, the corresponding bit will be set in this register. To clear the Event, set the corresponding bit in this register.

Field	Name	Description
[0]	ZERO	1 if a zero event occurred. This Event is triggered on a falling edge.

TIMER0_EV_ENABLE

Address: $0xf0007800 + 0x1c = 0xf000781c$

This register enables the corresponding zero events. Write a 0 to this register to disable individual events.

Field	Name	Description
[0]	ZERO	Write a 1 to enable the zero Event

12.2.17 UART

Register Listing for UART

Register	Address
<i>UART_RXTX</i>	0xf0008000
<i>UART_TXFULL</i>	0xf0008004
<i>UART_RXEMPTY</i>	0xf0008008
<i>UART_EV_STATUS</i>	0xf000800c
<i>UART_EV_PENDING</i>	0xf0008010
<i>UART_EV_ENABLE</i>	0xf0008014
<i>UART_TXEMPTY</i>	0xf0008018
<i>UART_RXFULL</i>	0xf000801c
<i>UART_XOVER_RXTX</i>	0xf0008020
<i>UART_XOVER_TXFULL</i>	0xf0008024
<i>UART_XOVER_RXEMPTY</i>	0xf0008028
<i>UART_XOVER_EV_STATUS</i>	0xf000802c
<i>UART_XOVER_EV_PENDING</i>	0xf0008030
<i>UART_XOVER_EV_ENABLE</i>	0xf0008034
<i>UART_XOVER_TXEMPTY</i>	0xf0008038
<i>UART_XOVER_RXFULL</i>	0xf000803c

UART_RXTX

Address: $0xf0008000 + 0x0 = 0xf0008000$

UART_TXFULL

Address: $0xf0008000 + 0x4 = 0xf0008004$

TX FIFO Full.

UART_RXEMPTY

Address: $0xf0008000 + 0x8 = 0xf0008008$

RX FIFO Empty.

UART_EV_STATUS

Address: $0xf0008000 + 0xc = 0xf000800c$

This register contains the current raw level of the rx event trigger. Writes to this register have no effect.

Field	Name	Description
[0]	TX	Level of the tx event
[1]	RX	Level of the rx event

UART_EV_PENDING

Address: $0xf0008000 + 0x10 = 0xf0008010$

When a rx event occurs, the corresponding bit will be set in this register. To clear the Event, set the corresponding bit in this register.

Field	Name	Description
[0]	TX	1 if a tx event occurred. This Event is triggered on a falling edge.
[1]	RX	1 if a rx event occurred. This Event is triggered on a falling edge.

UART_EV_ENABLE

Address: $0xf0008000 + 0x14 = 0xf0008014$

This register enables the corresponding rx events. Write a 0 to this register to disable individual events.

Field	Name	Description
[0]	TX	Write a 1 to enable the tx Event
[1]	RX	Write a 1 to enable the rx Event

UART_TXEMPTY

Address: $0xf0008000 + 0x18 = 0xf0008018$

TX FIFO Empty.

UART_RXFULL

Address: $0xf0008000 + 0x1c = 0xf000801c$

RX FIFO Full.

UART_XOVER_RXTX

Address: $0xf0008000 + 0x20 = 0xf0008020$

UART_XOVER_TXFULL

Address: $0xf0008000 + 0x24 = 0xf0008024$

TX FIFO Full.

UART_XOVER_RXEMPTY

Address: $0xf0008000 + 0x28 = 0xf0008028$

RX FIFO Empty.

UART_XOVER_EV_STATUS

Address: $0xf0008000 + 0x2c = 0xf000802c$

This register contains the current raw level of the rx event trigger. Writes to this register have no effect.

Field	Name	Description
[0]	TX	Level of the tx event
[1]	RX	Level of the rx event

UART_XOVER_EV_PENDING

Address: $0xf0008000 + 0x30 = 0xf0008030$

When a rx event occurs, the corresponding bit will be set in this register. To clear the Event, set the corresponding bit in this register.

Field	Name	Description
[0]	TX	1 if a tx event occurred. This Event is triggered on a falling edge .
[1]	RX	1 if a rx event occurred. This Event is triggered on a falling edge .

UART_XOVER_EV_ENABLE

Address: $0xf0008000 + 0x34 = 0xf0008034$

This register enables the corresponding rx events. Write a 0 to this register to disable individual events.

Field	Name	Description
[0]	TX	Write a 1 to enable the tx Event
[1]	RX	Write a 1 to enable the rx Event

UART_XOVER_TXEMPTY

Address: $0xf0008000 + 0x38 = 0xf0008038$

TX FIFO Empty.

UART_XOVER_RXFULL

Address: $0xf0008000 + 0x3c = 0xf000803c$

RX FIFO Full.

CHAPTER
THIRTEEN

DOCUMENTATION FOR ROW HAMMER TESTER ZCU104

13.1 Modules

13.1.1 Interrupt Controller

This device has an EventManager-based interrupt system. Individual modules generate *events* which are wired into a central interrupt controller.

When an interrupt occurs, you should look the interrupt number up in the CPU- specific interrupt table and then call the relevant module.

Assigned Interrupts

The following interrupts are assigned on this system:

Interrupt	Module
1	<i>TIMER0</i>
0	<i>UART</i>

13.2 Register Groups

13.2.1 LEDs

Register Listing for LEDs

Register	Address
<i>LEDS_OUT</i>	<i>0xf0000000</i>

LEDS_OUT

Address: $0xf0000000 + 0x0 = 0xf0000000$

Led Output(s) Control.

13.2.2 DDRPHY

Register Listing for DDRPHY

Register	Address
<i>DDRPHY_RST</i>	<i>0xf0000800</i>
<i>DDRPHY_EN_VTC</i>	<i>0xf0000804</i>
<i>DDRPHY_HALF_SYS8X_TAPS</i>	<i>0xf0000808</i>
<i>DDRPHY_WLEVEL_EN</i>	<i>0xf000080c</i>
<i>DDRPHY_WLEVEL_STROBE</i>	<i>0xf0000810</i>
<i>DDRPHY_CDLY_RST</i>	<i>0xf0000814</i>
<i>DDRPHY_CDLY_INC</i>	<i>0xf0000818</i>
<i>DDRPHY_CDLY_VALUE</i>	<i>0xf000081c</i>
<i>DDRPHY_DLY_SEL</i>	<i>0xf0000820</i>
<i>DDRPHY_RDLY_DQ_RST</i>	<i>0xf0000824</i>
<i>DDRPHY_RDLY_DQ_INC</i>	<i>0xf0000828</i>
<i>DDRPHY_RDLY_DQ_BITSLIP_RST</i>	<i>0xf000082c</i>
<i>DDRPHY_RDLY_DQ_BITSLIP</i>	<i>0xf0000830</i>
<i>DDRPHY_WDLY_DQ_RST</i>	<i>0xf0000834</i>
<i>DDRPHY_WDLY_DQ_INC</i>	<i>0xf0000838</i>
<i>DDRPHY_WDLY_DQS_RST</i>	<i>0xf000083c</i>
<i>DDRPHY_WDLY_DQS_INC</i>	<i>0xf0000840</i>
<i>DDRPHY_WDLY_DQS_INC_COUNT</i>	<i>0xf0000844</i>
<i>DDRPHY_WDLY_DQ_BITSLIP_RST</i>	<i>0xf0000848</i>
<i>DDRPHY_WDLY_DQ_BITSLIP</i>	<i>0xf000084c</i>
<i>DDRPHY_RDPHASE</i>	<i>0xf0000850</i>
<i>DDRPHY_WRPHASE</i>	<i>0xf0000854</i>

DDRPHY_RST

Address: $0xf0000800 + 0x0 = 0xf0000800$

DDRPHY_EN_VTC

Address: $0xf0000800 + 0x4 = 0xf0000804$

DDRPHY_HALF_SYS8X_TAPS

Address: $0xf0000800 + 0x8 = 0xf0000808$

DDRPHY_WLEVEL_EN

Address: $0xf0000800 + 0xc = 0xf000080c$

DDRPHY_WLEVEL_STROBE

Address: $0xf0000800 + 0x10 = 0xf0000810$

DDRPHY_CDLY_RST

Address: $0xf0000800 + 0x14 = 0xf0000814$

DDRPHY_CDLY_INC

Address: $0xf0000800 + 0x18 = 0xf0000818$

DDRPHY_CDLY_VALUE

Address: 0xf0000800 + 0x1c = 0xf000081c

DDRPHY_DLY_SEL

Address: 0xf0000800 + 0x20 = 0xf0000820

DDRPHY_RDLY_DQ_RST

Address: 0xf0000800 + 0x24 = 0xf0000824

DDRPHY_RDLY_DQ_INC

Address: 0xf0000800 + 0x28 = 0xf0000828

DDRPHY_RDLY_DQ_BITSLIP_RST

Address: 0xf0000800 + 0x2c = 0xf000082c

DDRPHY_RDLY_DQ_BITSLIP

Address: 0xf0000800 + 0x30 = 0xf0000830

DDRPHY_WDLY_DQ_RST

Address: 0xf0000800 + 0x34 = 0xf0000834

DDRPHY_WDLY_DQ_INC

Address: $0xf0000800 + 0x38 = 0xf0000838$

DDRPHY_WDLY_DQS_RST

Address: $0xf0000800 + 0x3c = 0xf000083c$

DDRPHY_WDLY_DQS_INC

Address: $0xf0000800 + 0x40 = 0xf0000840$

DDRPHY_WDLY_DQS_INC_COUNT

Address: $0xf0000800 + 0x44 = 0xf0000844$

DDRPHY_WDLY_DQ_BITSLIP_RST

Address: $0xf0000800 + 0x48 = 0xf0000848$

DDRPHY_WDLY_DQ_BITSLIP

Address: $0xf0000800 + 0x4c = 0xf000084c$

DDRPHY_RDPHASE

Address: $0xf0000800 + 0x50 = 0xf0000850$

DDRPHY_WRPAGE

Address: $0xf0000800 + 0x54 = 0xf0000854$

13.2.3 CONTROLLER_SETTINGS

Allows to change LiteDRAMController behaviour at runtime

Register Listing for CONTROLLER_SETTINGS

Register	Address
<i>CONTROLLER_SETTINGS_REFRESH</i>	<i>0xf0001000</i>

CONTROLLER_SETTINGS_REFRESH

Address: $0xf0001000 + 0x0 = 0xf0001000$

Enable/disable Refresh commands sending

13.2.4 DDRCTRL

Register Listing for DDRCTRL

Register	Address
<i>DDRCTRL_INIT_DONE</i>	<i>0xf0001800</i>
<i>DDRCTRL_INIT_ERROR</i>	<i>0xf0001804</i>

DDRCTRL_INIT_DONE

Address: $0xf0001800 + 0x0 = 0xf0001800$

DDRCTRL_INIT_ERROR

Address: $0xf0001800 + 0x4 = 0xf0001804$

13.2.5 ROWHAMMER

Row Hammer DMA attacker

This module allows to perform a Row Hammer attack by configuring it with two addresses that map to different rows of a single bank. When enabled, it will perform alternating DMA reads from the given locations, which will result in the DRAM controller having to repeatedly open/close rows at each read access.

Register Listing for ROWHAMMER

Register	Address
<i>ROWHAMMER_ENABLED</i>	<i>0xf0002000</i>
<i>ROWHAMMER_ADDRESS1</i>	<i>0xf0002004</i>
<i>ROWHAMMER_ADDRESS2</i>	<i>0xf0002008</i>
<i>ROWHAMMER_COUNT</i>	<i>0xf000200c</i>

ROWHAMMER_ENABLED

Address: 0xf0002000 + 0x0 = 0xf0002000

Used to start/stop the operation of the module

ROWHAMMER_ADDRESS1

Address: 0xf0002000 + 0x4 = 0xf0002004

First attacked address

ROWHAMMER_ADDRESS2

Address: 0xf0002000 + 0x8 = 0xf0002008

Second attacked address

ROWHAMMER_COUNT

Address: 0xf0002000 + 0xc = 0xf000200c

This is the number of DMA accesses performed. When the module is enabled, the value can be freely read. When the module is disabled, the register is clear-on-write and has to be read before the next attack.

13.2.6 WRITER

DMA DRAM writer.

Allows to fill DRAM with a predefined pattern using DMA.

Pattern

Provides access to RAM to store access pattern: *mem_addr* and *mem_data*. The pattern address space can be limited using the *data_mask*.

For example, having *mem_addr* filled with [0x04, 0x02, 0x03, ...] and *mem_data* filled with [0xff, 0xaa, 0x55, ...] and setting *data_mask* = 0b01, the pattern [(address, data), ...] written will be: [(0x04, 0xff), (0x02, 0xaa), (0x04, 0xff), ...] (wraps due to masking).

DRAM memory range that is being accessed can be configured using *mem_mask*.

To use this module, make sure that *ready* is 1, then write the desired number of transfers to *count*. Writing to the *start* CSR will initialize the operation. When the operation is ongoing *ready* will be 0.

Register Listing for WRITER

Register	Address
WRITER_START	0xf0002800
WRITER_READY	0xf0002804
WRITER_MODULO	0xf0002808
WRITER_COUNT	0xf000280c
WRITER_DONE	0xf0002810
WRITER_MEM_MASK	0xf0002814
WRITER_DATA_MASK	0xf0002818
WRITER_DATA_DIV	0xf000281c
WRITER_INVERTER_DIVISOR_MASK	0xf0002820
WRITER_INVERTER_SELECTION_MASK	0xf0002824
WRITER_LAST_ADDRESS	0xf0002828

WRITER_START

Address: 0xf0002800 + 0x0 = 0xf0002800

Write to the register starts the transfer (if ready=1)

WRITER_READY

Address: $0xf0002800 + 0x4 = 0xf0002804$

Indicates that the transfer is not ongoing

WRITER_MODULO

Address: $0xf0002800 + 0x8 = 0xf0002808$

When set use modulo to calculate DMA transfers address rather than bit masking

WRITER_COUNT

Address: $0xf0002800 + 0xc = 0xf000280c$

Desired number of DMA transfers

WRITER_DONE

Address: $0xf0002800 + 0x10 = 0xf0002810$

Number of completed DMA transfers

WRITER_MEM_MASK

Address: $0xf0002800 + 0x14 = 0xf0002814$

DRAM address mask for DMA transfers

WRITER_DATA_MASK

Address: $0xf0002800 + 0x18 = 0xf0002818$

Pattern memory address mask

WRITER_DATA_DIV

Address: $0xf0002800 + 0x1c = 0xf000281c$

Pattern memory address divisor-1

WRITER_INVERTER_DIVISOR_MASK

Address: $0xf0002800 + 0x20 = 0xf0002820$

Divisor mask for selecting rows for which pattern data gets inverted

WRITER_INVERTER_SELECTION_MASK

Address: $0xf0002800 + 0x24 = 0xf0002824$

Selection mask for selecting rows for which pattern data gets inverted

WRITER_LAST_ADDRESS

Address: $0xf0002800 + 0x28 = 0xf0002828$

Number of completed DMA transfers

13.2.7 READER

DMA DRAM reader.

Allows to check DRAM contents against a predefined pattern using DMA.

Pattern

Provides access to RAM to store access pattern: *mem_addr* and *mem_data*. The pattern address space can be limited using the *data_mask*.

For example, having *mem_addr* filled with [0x04, 0x02, 0x03, ...] and *mem_data* filled with [0xff, 0xaa, 0x55, ...] and setting *data_mask* = 0b01, the pattern [(address, data), ...] written will be: [(0x04, 0xff), (0x02, 0xaa), (0x04, 0xff), ...] (wraps due to masking).

DRAM memory range that is being accessed can be configured using *mem_mask*.

To use this module, make sure that *ready* is 1, then write the desired number of transfers to *count*. Writing to the *start* CSR will initialize the operation. When the operation is ongoing *ready* will be 0.

Reading errors

This module allows to check the locations of errors in the memory. It scans the configured memory area and compares the values read to the predefined pattern. If *skip_fifo* is 0, this module will stop after each error encountered, so that it can be examined. Wait until the *error_ready* CSR is 1. Then use the CSRs *error_offset*, *error_data* and *error_expected* to examine the errors in the current transfer. To continue reading, write 1 to *error_continue* CSR. Setting *skip_fifo* to 1 will disable this behaviour entirely.

The final number of errors can be read from *error_count*. NOTE: This value represents the number of erroneous DMA transfers.

The current progress can be read from the *done* CSR.

Register Listing for READER

Register	Address
READER_START	0xf0003000
READER_READY	0xf0003004
READER_MODULO	0xf0003008
READER_COUNT	0xf000300c
READER_DONE	0xf0003010
READER_MEM_MASK	0xf0003014
READER_DATA_MASK	0xf0003018
READER_DATA_DIV	0xf000301c
READER_INVERTER_DIVISOR_MASK	0xf0003020
READER_INVERTER_SELECTION_MASK	0xf0003024
READER_ERROR_COUNT	0xf0003028
READER_SKIP_FIFO	0xf000302c
READER_ERROR_OFFSET	0xf0003030
READER_ERROR_DATA15	0xf0003034
READER_ERROR_DATA14	0xf0003038
READER_ERROR_DATA13	0xf000303c
READER_ERROR_DATA12	0xf0003040
READER_ERROR_DATA11	0xf0003044
READER_ERROR_DATA10	0xf0003048
READER_ERROR_DATA9	0xf000304c
READER_ERROR_DATA8	0xf0003050
READER_ERROR_DATA7	0xf0003054
READER_ERROR_DATA6	0xf0003058
READER_ERROR_DATA5	0xf000305c
READER_ERROR_DATA4	0xf0003060
READER_ERROR_DATA3	0xf0003064
READER_ERROR_DATA2	0xf0003068
READER_ERROR_DATA1	0xf000306c
READER_ERROR_DATA0	0xf0003070
READER_ERROR_EXPECTED15	0xf0003074
READER_ERROR_EXPECTED14	0xf0003078
READER_ERROR_EXPECTED13	0xf000307c
READER_ERROR_EXPECTED12	0xf0003080
READER_ERROR_EXPECTED11	0xf0003084
READER_ERROR_EXPECTED10	0xf0003088
READER_ERROR_EXPECTED9	0xf000308c
READER_ERROR_EXPECTED8	0xf0003090
READER_ERROR_EXPECTED7	0xf0003094
READER_ERROR_EXPECTED6	0xf0003098
READER_ERROR_EXPECTED5	0xf000309c
READER_ERROR_EXPECTED4	0xf00030a0
READER_ERROR_EXPECTED3	0xf00030a4
READER_ERROR_EXPECTED2	0xf00030a8

continues on next page

Table 13.1 – continued from previous page

Register	Address
<i>READER_ERROR_EXPECTED1</i>	<i>0xf00030ac</i>
<i>READER_ERROR_EXPECTED0</i>	<i>0xf00030b0</i>
<i>READER_ERROR_READY</i>	<i>0xf00030b4</i>
<i>READER_ERROR_CONTINUE</i>	<i>0xf00030b8</i>

READER_START

Address: 0xf0003000 + 0x0 = 0xf0003000

Write to the register starts the transfer (if ready=1)

READER_READY

Address: 0xf0003000 + 0x4 = 0xf0003004

Indicates that the transfer is not ongoing

READER_MODULO

Address: 0xf0003000 + 0x8 = 0xf0003008

When set use modulo to calculate DMA transfers address rather than bit masking

READER_COUNT

Address: 0xf0003000 + 0xc = 0xf000300c

Desired number of DMA transfers

READER_DONE

Address: 0xf0003000 + 0x10 = 0xf0003010

Number of completed DMA transfers

READER_MEM_MASK

Address: 0xf0003000 + 0x14 = 0xf0003014

DRAM address mask for DMA transfers

READER_DATA_MASK

Address: $0xf0003000 + 0x18 = 0xf0003018$

Pattern memory address mask

READER_DATA_DIV

Address: $0xf0003000 + 0x1c = 0xf000301c$

Pattern memory address divisor-1

READER_INVERTER_DIVISOR_MASK

Address: $0xf0003000 + 0x20 = 0xf0003020$

Divisor mask for selecting rows for which pattern data gets inverted

READER_INVERTER_SELECTION_MASK

Address: $0xf0003000 + 0x24 = 0xf0003024$

Selection mask for selecting rows for which pattern data gets inverted

READER_ERROR_COUNT

Address: $0xf0003000 + 0x28 = 0xf0003028$

Number of errors detected

READER_SKIP_FIFO

Address: $0xf0003000 + 0x2c = 0xf000302c$

Skip waiting for user to read the errors FIFO

READER_ERROR_OFFSET

Address: $0xf0003000 + 0x30 = 0xf0003030$

Current offset of the error

READER_ERROR_DATA15

Address: $0xf0003000 + 0x34 = 0xf0003034$

Bits 480-511 of *READER_ERROR_DATA*. Erroneous value read from DRAM memory

READER_ERROR_DATA14

Address: $0xf0003000 + 0x38 = 0xf0003038$

Bits 448-479 of *READER_ERROR_DATA*.

READER_ERROR_DATA13

Address: $0xf0003000 + 0x3c = 0xf000303c$

Bits 416-447 of *READER_ERROR_DATA*.

READER_ERROR_DATA12

Address: $0xf0003000 + 0x40 = 0xf0003040$

Bits 384-415 of *READER_ERROR_DATA*.

READER_ERROR_DATA11

Address: $0xf0003000 + 0x44 = 0xf0003044$

Bits 352-383 of *READER_ERROR_DATA*.

READER_ERROR_DATA10

Address: $0xf0003000 + 0x48 = 0xf0003048$

Bits 320-351 of *READER_ERROR_DATA*.

READER_ERROR_DATA9

Address: $0xf0003000 + 0x4c = 0xf000304c$

Bits 288-319 of *READER_ERROR_DATA*.

READER_ERROR_DATA8

Address: $0xf0003000 + 0x50 = 0xf0003050$

Bits 256-287 of *READER_ERROR_DATA*.

READER_ERROR_DATA7

Address: $0xf0003000 + 0x54 = 0xf0003054$

Bits 224-255 of *READER_ERROR_DATA*.

READER_ERROR_DATA6

Address: $0xf0003000 + 0x58 = 0xf0003058$

Bits 192-223 of *READER_ERROR_DATA*.

READER_ERROR_DATA5

Address: $0xf0003000 + 0x5c = 0xf000305c$

Bits 160-191 of *READER_ERROR_DATA*.

READER_ERROR_DATA4

Address: $0xf0003000 + 0x60 = 0xf0003060$

Bits 128-159 of *READER_ERROR_DATA*.

READER_ERROR_DATA3

Address: $0xf0003000 + 0x64 = 0xf0003064$

Bits 96-127 of *READER_ERROR_DATA*.

READER_ERROR_DATA2

Address: $0xf0003000 + 0x68 = 0xf0003068$

Bits 64-95 of *READER_ERROR_DATA*.

READER_ERROR_DATA1

Address: $0xf0003000 + 0x6c = 0xf000306c$

Bits 32-63 of *READER_ERROR_DATA*.

READER_ERROR_DATA0

Address: $0xf0003000 + 0x70 = 0xf0003070$

Bits 0-31 of *READER_ERROR_DATA*.

READER_ERROR_EXPECTED15

Address: $0xf0003000 + 0x74 = 0xf0003074$

Bits 480-511 of *READER_ERROR_EXPECTED*. Value expected to be read from DRAM memory

READER_ERROR_EXPECTED14

Address: $0xf0003000 + 0x78 = 0xf0003078$

Bits 448-479 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED13

Address: $0xf0003000 + 0x7c = 0xf000307c$

Bits 416-447 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED12

Address: $0xf0003000 + 0x80 = 0xf0003080$

Bits 384-415 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED11

Address: $0xf0003000 + 0x84 = 0xf0003084$

Bits 352-383 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED10

Address: $0xf0003000 + 0x88 = 0xf0003088$

Bits 320-351 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED9

Address: $0xf0003000 + 0x8c = 0xf000308c$

Bits 288-319 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED8

Address: $0xf0003000 + 0x90 = 0xf0003090$

Bits 256-287 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED7

Address: $0xf0003000 + 0x94 = 0xf0003094$

Bits 224-255 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED6

Address: $0xf0003000 + 0x98 = 0xf0003098$

Bits 192-223 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED5

Address: $0xf0003000 + 0x9c = 0xf000309c$

Bits 160-191 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED4

Address: $0xf0003000 + 0xa0 = 0xf00030a0$

Bits 128-159 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED3

Address: 0xf0003000 + 0xa4 = 0xf00030a4

Bits 96-127 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED2

Address: 0xf0003000 + 0xa8 = 0xf00030a8

Bits 64-95 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED1

Address: 0xf0003000 + 0xac = 0xf00030ac

Bits 32-63 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED0

Address: 0xf0003000 + 0xb0 = 0xf00030b0

Bits 0-31 of *READER_ERROR_EXPECTED*.

READER_ERROR_READY

Address: 0xf0003000 + 0xb4 = 0xf00030b4

Error detected and ready to read

READER_ERROR_CONTINUE

Address: 0xf0003000 + 0xb8 = 0xf00030b8

Continue reading until the next error

13.2.8 DFI_SWITCH

Register Listing for DFI_SWITCH

Register	Address
<i>DFI_SWITCH_REFRESH_COUNT</i>	<i>0xf0003800</i>
<i>DFI_SWITCH_AT_REFRESH</i>	<i>0xf0003804</i>
<i>DFI_SWITCH_REFRESH_UPDATE</i>	<i>0xf0003808</i>

DFI_SWITCH_REFRESH_COUNT

Address: $0xf0003800 + 0x0 = 0xf0003800$

Count of all refresh commands issued (both by Memory Controller and Payload Executor). Value is latched from internal counter on mode transition: MC -> PE or by writing to the *refresh_update* CSR.

DFI_SWITCH_AT_REFRESH

Address: $0xf0003800 + 0x4 = 0xf0003804$

If set to a value different than 0 the mode transition MC -> PE will be performed only when the value of this register matches the current refresh commands count.

DFI_SWITCH_REFRESH_UPDATE

Address: $0xf0003800 + 0x8 = 0xf0003808$

Force an update of the *refresh_count* CSR.

13.2.9 PAYLOAD_EXECUTOR

Executes the DRAM payload from memory

Instruction decoder

All instructions are 32-bit. The format of most instructions is the same, except for the LOOP instruction, which has a constant TIMESLICE of 1.

NOOP with a TIMESLICE of 0 is a special case which is interpreted as STOP instruction. When this instruction is encountered execution gets finished immediately.

NOTE: TIMESLICE is the number of cycles the instruction will take. This means that instructions other than NOOP that use TIMESLICE=0 are illegal (although will silently be executed as having TIMESLICE=1).

NOTE2: LOOP instruction will *jump* COUNT times, meaning that the “code” inside the loop will effectively be executed COUNT+1 times.

Op codes:

Op	Value
NOOP	0b000
LOOP	0b111
ACT	0b100
PRE	0b101
REF	0b110
ZQC	0b001
READ	0b010

Instruction format:

LSB	MSB
dfi:	OP_CODE TIMESLICE ADDRESS
noop:	OP_CODE TIMESLICE_NOOP
loop:	OP_CODE COUNT JUMP
stop:	<NOOP> 0

Where ADDRESS depends on the DFI command and is one of:

LSB	MSB
RANK BANK COLUMN	
RANK BANK ROW	

Register Listing for PAYLOAD_EXECUTOR

Register	Address
PAYLOAD_EXECUTOR_START	0xf0004000
PAYLOAD_EXECUTOR_STATUS	0xf0004004
PAYLOAD_EXECUTOR_READ_COUNT	0xf0004008

PAYLOAD_EXECUTOR_START

Address: $0xf0004000 + 0x0 = 0xf0004000$

Writing to this register initializes payload execution

PAYLOAD_EXECUTOR_STATUS

Address: $0xf0004000 + 0x4 = 0xf0004004$

Payload executor status register

Field	Name	Description
[0]	READY	Indicates that the executor is not running
[1]	OVER-FLOW	Indicates the scratchpad memory address counter has overflowed due to the number of READ commands sent during execution

PAYLOAD_EXECUTOR_READ_COUNT

Address: $0xf0004000 + 0x8 = 0xf0004008$

Number of data from READ commands that is stored in the scratchpad memory

13.2.10 I2C

Register Listing for I2C

Register	Address
I2C_W	<i>0xf0004800</i>
I2C_R	<i>0xf0004804</i>

I2C_W

Address: $0xf0004800 + 0x0 = 0xf0004800$

Field	Name	Description

I2C_R

Address: $0xf0004800 + 0x4 = 0xf0004804$

Field	Name	Description

13.2.11 CTRL

Register Listing for CTRL

Register	Address
<i>CTRL_RESET</i>	<i>0xf0005000</i>
<i>CTRL_SCRATCH</i>	<i>0xf0005004</i>
<i>CTRL_BUS_ERRORS</i>	<i>0xf0005008</i>

CTRL__RESET

Address: $0xf0005000 + 0x0 = 0xf0005000$

Field	Name	Description
[0]	SOC_RST	Write 1 to this register to reset the full SoC (Pulse Reset)
[1]	CPU_RST	Write 1 to this register to reset the CPU(s) of the SoC (Hold Reset)

CTRL_SCRATCH

Address: $0xf0005000 + 0x4 = 0xf0005004$

Use this register as a scratch space to verify that software read/write accesses to the Wishbone/CSR bus are working correctly. The initial reset value of 0x1234578 can be used to verify endianness.

CTRL_BUS_ERRORS

Address: $0xf0005000 + 0x8 = 0xf0005008$

Total number of Wishbone bus errors (timeouts) since start.

13.2.12 IDENTIFIER_MEM

Register Listing for IDENTIFIER_MEM

Register	Address
<i>IDENTIFIER_MEM</i>	<i>0xf0005800</i>

IDENTIFIER_MEM

Address: $0xf0005800 + 0x0 = 0xf0005800$

8 x 113-bit memory

13.2.13 SDRAM

Register Listing for SDRAM

Register	Address
<i>SDRAM_DFII_CONTROL</i>	<i>0xf0006000</i>
<i>SDRAM_DFII_PIO_COMMAND</i>	<i>0xf0006004</i>
<i>SDRAM_DFII_PIO_COMMAND_ISSUE</i>	<i>0xf0006008</i>
<i>SDRAM_DFII_PIO_ADDRESS</i>	<i>0xf000600c</i>
<i>SDRAM_DFII_PIO_BADDRESS</i>	<i>0xf0006010</i>
<i>SDRAM_DFII_PIO_WRDATA3</i>	<i>0xf0006014</i>
<i>SDRAM_DFII_PIO_WRDATA2</i>	<i>0xf0006018</i>
<i>SDRAM_DFII_PIO_WRDATA1</i>	<i>0xf000601c</i>
<i>SDRAM_DFII_PIO_WRDATA0</i>	<i>0xf0006020</i>
<i>SDRAM_DFII_PIO_RDDATA3</i>	<i>0xf0006024</i>
<i>SDRAM_DFII_PIO_RDDATA2</i>	<i>0xf0006028</i>
<i>SDRAM_DFII_PIO_RDDATA1</i>	<i>0xf000602c</i>
<i>SDRAM_DFII_PIO_RDDATA0</i>	<i>0xf0006030</i>
<i>SDRAM_DFII_PI1_COMMAND</i>	<i>0xf0006034</i>
<i>SDRAM_DFII_PI1_COMMAND_ISSUE</i>	<i>0xf0006038</i>
<i>SDRAM_DFII_PI1_ADDRESS</i>	<i>0xf000603c</i>
<i>SDRAM_DFII_PI1_BADDRESS</i>	<i>0xf0006040</i>
<i>SDRAM_DFII_PI1_WRDATA3</i>	<i>0xf0006044</i>
<i>SDRAM_DFII_PI1_WRDATA2</i>	<i>0xf0006048</i>
<i>SDRAM_DFII_PI1_WRDATA1</i>	<i>0xf000604c</i>
<i>SDRAM_DFII_PI1_WRDATA0</i>	<i>0xf0006050</i>

continues on next page

Table 13.2 – continued from previous page

Register	Address
<i>SDRAM_DFII_PI1_RDDATA3</i>	0xf0006054
<i>SDRAM_DFII_PI1_RDDATA2</i>	0xf0006058
<i>SDRAM_DFII_PI1_RDDATA1</i>	0xf000605c
<i>SDRAM_DFII_PI1_RDDATA0</i>	0xf0006060
<i>SDRAM_DFII_PI2_COMMAND</i>	0xf0006064
<i>SDRAM_DFII_PI2_COMMAND_ISSUE</i>	0xf0006068
<i>SDRAM_DFII_PI2_ADDRESS</i>	0xf000606c
<i>SDRAM_DFII_PI2_BADDRESS</i>	0xf0006070
<i>SDRAM_DFII_PI2_WRDATA3</i>	0xf0006074
<i>SDRAM_DFII_PI2_WRDATA2</i>	0xf0006078
<i>SDRAM_DFII_PI2_WRDATA1</i>	0xf000607c
<i>SDRAM_DFII_PI2_WRDATA0</i>	0xf0006080
<i>SDRAM_DFII_PI2_RDDATA3</i>	0xf0006084
<i>SDRAM_DFII_PI2_RDDATA2</i>	0xf0006088
<i>SDRAM_DFII_PI2_RDDATA1</i>	0xf000608c
<i>SDRAM_DFII_PI2_RDDATA0</i>	0xf0006090
<i>SDRAM_DFII_PI3_COMMAND</i>	0xf0006094
<i>SDRAM_DFII_PI3_COMMAND_ISSUE</i>	0xf0006098
<i>SDRAM_DFII_PI3_ADDRESS</i>	0xf000609c
<i>SDRAM_DFII_PI3_BADDRESS</i>	0xf00060a0
<i>SDRAM_DFII_PI3_WRDATA3</i>	0xf00060a4
<i>SDRAM_DFII_PI3_WRDATA2</i>	0xf00060a8
<i>SDRAM_DFII_PI3_WRDATA1</i>	0xf00060ac
<i>SDRAM_DFII_PI3_WRDATA0</i>	0xf00060b0
<i>SDRAM_DFII_PI3_RDDATA3</i>	0xf00060b4
<i>SDRAM_DFII_PI3_RDDATA2</i>	0xf00060b8
<i>SDRAM_DFII_PI3_RDDATA1</i>	0xf00060bc
<i>SDRAM_DFII_PI3_RDDATA0</i>	0xf00060c0
<i>SDRAM_CONTROLLER_TRP</i>	0xf00060c4
<i>SDRAM_CONTROLLER_TRCD</i>	0xf00060c8
<i>SDRAM_CONTROLLER_TWR</i>	0xf00060cc
<i>SDRAM_CONTROLLER_TWTR</i>	0xf00060d0
<i>SDRAM_CONTROLLER_TREFI</i>	0xf00060d4
<i>SDRAM_CONTROLLER_TRFC</i>	0xf00060d8
<i>SDRAM_CONTROLLER_TFAW</i>	0xf00060dc
<i>SDRAM_CONTROLLER_TCCD</i>	0xf00060e0
<i>SDRAM_CONTROLLER_TCCD_WR</i>	0xf00060e4
<i>SDRAM_CONTROLLER_TRTP</i>	0xf00060e8
<i>SDRAM_CONTROLLER_TRRD</i>	0xf00060ec
<i>SDRAM_CONTROLLER_TRC</i>	0xf00060f0
<i>SDRAM_CONTROLLER_TRAS</i>	0xf00060f4
<i>SDRAM_CONTROLLER_TZQCS</i>	0xf00060f8
<i>SDRAM_CONTROLLER_LAST_ADDR_0</i>	0xf00060fc
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_0</i>	0xf0006100
<i>SDRAM_CONTROLLER_LAST_ADDR_1</i>	0xf0006104
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_1</i>	0xf0006108
<i>SDRAM_CONTROLLER_LAST_ADDR_2</i>	0xf000610c
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_2</i>	0xf0006110

continues on next page

Table 13.2 – continued from previous page

Register	Address
<i>SDRAM_CONTROLLER_LAST_ADDR_3</i>	0xf0006114
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_3</i>	0xf0006118
<i>SDRAM_CONTROLLER_LAST_ADDR_4</i>	0xf000611c
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_4</i>	0xf0006120
<i>SDRAM_CONTROLLER_LAST_ADDR_5</i>	0xf0006124
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_5</i>	0xf0006128
<i>SDRAM_CONTROLLER_LAST_ADDR_6</i>	0xf000612c
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_6</i>	0xf0006130
<i>SDRAM_CONTROLLER_LAST_ADDR_7</i>	0xf0006134
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_7</i>	0xf0006138

SDRAM_DFII_CONTROL

Address: $0xf0006000 + 0x0 = 0xf0006000$

Control DFI signals common to all phases

Field	Name	Description						
[0]	SEL	<table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>0b0</td> <td>Software control. (CPU)</td> </tr> <tr> <td>0b1</td> <td>Hardware control (default).</td> </tr> </tbody> </table>	Value	Description	0b0	Software control. (CPU)	0b1	Hardware control (default).
Value	Description							
0b0	Software control. (CPU)							
0b1	Hardware control (default).							
[1]	CKE	DFI clock enable bus						
[2]	ODT	DFI on-die termination bus						
[3]	RESET_N	DFI clock reset bus						

SDRAM_DFII_PIO_COMMAND

Address: $0xf0006000 + 0x4 = 0xf0006004$

Control DFI signals on a single phase

Field	Name	Description
[0]	CS	DFI chip select bus
[1]	WE	DFI write enable bus
[2]	CAS	DFI column address strobe bus
[3]	RAS	DFI row address strobe bus
[4]	WREN	DFI write data enable bus
[5]	RDEN	DFI read data enable bus

SDRAM_DFIIPIO_COMMAND_ISSUE

Address: $0xf0006000 + 0x8 = 0xf0006008$

SDRAM_DFIIPIO_ADDRESS

Address: $0xf0006000 + 0xc = 0xf000600c$

DFI address bus

SDRAM_DFIIPIO_BADDRESS

Address: $0xf0006000 + 0x10 = 0xf0006010$

DFI bank address bus

SDRAM_DFIIPIO_WRDAT3

Address: $0xf0006000 + 0x14 = 0xf0006014$

Bits 96-127 of *SDRAM_DFIIPIO_WRDAT3*. DFI write data bus

SDRAM_DFIIPIO_WRDAT2

Address: $0xf0006000 + 0x18 = 0xf0006018$

Bits 64-95 of *SDRAM_DFIIPIO_WRDAT2*.

SDRAM_DFIIPIO_WRDAT1

Address: $0xf0006000 + 0x1c = 0xf000601c$

Bits 32-63 of *SDRAM_DFIIPIO_WRDAT1*.

SDRAM_DFIIPIO_WRDAT0

Address: $0xf0006000 + 0x20 = 0xf0006020$

Bits 0-31 of *SDRAM_DFIIPIO_WRDAT0*.

SDRAM_DFII_PIO_RDDATA3

Address: 0xf0006000 + 0x24 = 0xf0006024

Bits 96-127 of *SDRAM_DFII_PIO_RDDATA*. DFI read data bus

SDRAM_DFII_PIO_RDDATA2

Address: 0xf0006000 + 0x28 = 0xf0006028

Bits 64-95 of *SDRAM_DFII_PIO_RDDATA*.

SDRAM_DFII_PIO_RDDATA1

Address: 0xf0006000 + 0x2c = 0xf000602c

Bits 32-63 of *SDRAM_DFII_PIO_RDDATA*.

SDRAM_DFII_PIO_RDDATA0

Address: 0xf0006000 + 0x30 = 0xf0006030

Bits 0-31 of *SDRAM_DFII_PIO_RDDATA*.

SDRAM_DFII_PI1_COMMAND

Address: 0xf0006000 + 0x34 = 0xf0006034

Control DFI signals on a single phase

Field	Name	Description
[0]	CS	DFI chip select bus
[1]	WE	DFI write enable bus
[2]	CAS	DFI column address strobe bus
[3]	RAS	DFI row address strobe bus
[4]	WREN	DFI write data enable bus
[5]	RDEN	DFI read data enable bus

SDRAM_DFII_PI1_COMMAND_ISSUE

Address: 0xf0006000 + 0x38 = 0xf0006038

SDRAM_DFII_PI1_ADDRESS

Address: $0xf0006000 + 0x3c = 0xf000603c$

DFI address bus

SDRAM_DFII_PI1_BADDRESS

Address: $0xf0006000 + 0x40 = 0xf0006040$

DFI bank address bus

SDRAM_DFII_PI1_WRDATA3

Address: $0xf0006000 + 0x44 = 0xf0006044$

Bits 96-127 of *SDRAM_DFII_PI1_WRDATA*. DFI write data bus

SDRAM_DFII_PI1_WRDATA2

Address: $0xf0006000 + 0x48 = 0xf0006048$

Bits 64-95 of *SDRAM_DFII_PI1_WRDATA*.

SDRAM_DFII_PI1_WRDATA1

Address: $0xf0006000 + 0x4c = 0xf000604c$

Bits 32-63 of *SDRAM_DFII_PI1_WRDATA*.

SDRAM_DFII_PI1_WRDATA0

Address: $0xf0006000 + 0x50 = 0xf0006050$

Bits 0-31 of *SDRAM_DFII_PI1_WRDATA*.

SDRAM_DFII_PI1_RDDATA3

Address: $0xf0006000 + 0x54 = 0xf0006054$

Bits 96-127 of *SDRAM_DFII_PI1_RDDATA*. DFI read data bus

SDRAM_DFII_PI1_RDDATA2

Address: 0xf0006000 + 0x58 = 0xf0006058

Bits 64-95 of *SDRAM_DFII_PI1_RDDATA*.

SDRAM_DFII_PI1_RDDATA1

Address: 0xf0006000 + 0x5c = 0xf000605c

Bits 32-63 of *SDRAM_DFII_PI1_RDDATA*.

SDRAM_DFII_PI1_RDDATA0

Address: 0xf0006000 + 0x60 = 0xf0006060

Bits 0-31 of *SDRAM_DFII_PI1_RDDATA*.

SDRAM_DFII_PI2_COMMAND

Address: 0xf0006000 + 0x64 = 0xf0006064

Control DFI signals on a single phase

Field	Name	Description
[0]	CS	DFI chip select bus
[1]	WE	DFI write enable bus
[2]	CAS	DFI column address strobe bus
[3]	RAS	DFI row address strobe bus
[4]	WREN	DFI write data enable bus
[5]	RDEN	DFI read data enable bus

SDRAM_DFII_PI2_COMMAND_ISSUE

Address: 0xf0006000 + 0x68 = 0xf0006068

SDRAM_DFII_PI2_ADDRESS

Address: 0xf0006000 + 0x6c = 0xf000606c

DFI address bus

SDRAM_DFII_PI2_BADDRESS

Address: $0xf0006000 + 0x70 = 0xf0006070$

DFI bank address bus

SDRAM_DFII_PI2_WRDATA3

Address: $0xf0006000 + 0x74 = 0xf0006074$

Bits 96-127 of *SDRAM_DFII_PI2_WRDATA*. DFI write data bus

SDRAM_DFII_PI2_WRDATA2

Address: $0xf0006000 + 0x78 = 0xf0006078$

Bits 64-95 of *SDRAM_DFII_PI2_WRDATA*.

SDRAM_DFII_PI2_WRDATA1

Address: $0xf0006000 + 0x7c = 0xf000607c$

Bits 32-63 of *SDRAM_DFII_PI2_WRDATA*.

SDRAM_DFII_PI2_WRDATA0

Address: $0xf0006000 + 0x80 = 0xf0006080$

Bits 0-31 of *SDRAM_DFII_PI2_WRDATA*.

SDRAM_DFII_PI2_RDDATA3

Address: $0xf0006000 + 0x84 = 0xf0006084$

Bits 96-127 of *SDRAM_DFII_PI2_RDDATA*. DFI read data bus

SDRAM_DFII_PI2_RDDATA2

Address: $0xf0006000 + 0x88 = 0xf0006088$

Bits 64-95 of *SDRAM_DFII_PI2_RDDATA*.

SDRAM_DFII_PI2_RDDATA1

Address: 0xf0006000 + 0x8c = 0xf000608c

Bits 32-63 of *SDRAM_DFII_PI2_RDDATA*.

SDRAM_DFII_PI2_RDDATA0

Address: 0xf0006000 + 0x90 = 0xf0006090

Bits 0-31 of *SDRAM_DFII_PI2_RDDATA*.

SDRAM_DFII_PI3_COMMAND

Address: 0xf0006000 + 0x94 = 0xf0006094

Control DFI signals on a single phase

Field	Name	Description
[0]	CS	DFI chip select bus
[1]	WE	DFI write enable bus
[2]	CAS	DFI column address strobe bus
[3]	RAS	DFI row address strobe bus
[4]	WREN	DFI write data enable bus
[5]	RDEN	DFI read data enable bus

SDRAM_DFII_PI3_COMMAND_ISSUE

Address: 0xf0006000 + 0x98 = 0xf0006098

SDRAM_DFII_PI3_ADDRESS

Address: 0xf0006000 + 0x9c = 0xf000609c

DFI address bus

SDRAM_DFII_PI3_BADDRESS

Address: 0xf0006000 + 0xa0 = 0xf00060a0

DFI bank address bus

SDRAM_DFII_PI3_WRDATA3

Address: $0xf0006000 + 0xa4 = 0xf00060a4$

Bits 96-127 of *SDRAM_DFII_PI3_WRDATA*. DFI write data bus

SDRAM_DFII_PI3_WRDATA2

Address: $0xf0006000 + 0xa8 = 0xf00060a8$

Bits 64-95 of *SDRAM_DFII_PI3_WRDATA*.

SDRAM_DFII_PI3_WRDATA1

Address: $0xf0006000 + 0xac = 0xf00060ac$

Bits 32-63 of *SDRAM_DFII_PI3_WRDATA*.

SDRAM_DFII_PI3_WRDATA0

Address: $0xf0006000 + 0xb0 = 0xf00060b0$

Bits 0-31 of *SDRAM_DFII_PI3_WRDATA*.

SDRAM_DFII_PI3_RDDATA3

Address: $0xf0006000 + 0xb4 = 0xf00060b4$

Bits 96-127 of *SDRAM_DFII_PI3_RDDATA*. DFI read data bus

SDRAM_DFII_PI3_RDDATA2

Address: $0xf0006000 + 0xb8 = 0xf00060b8$

Bits 64-95 of *SDRAM_DFII_PI3_RDDATA*.

SDRAM_DFII_PI3_RDDATA1

Address: $0xf0006000 + 0xbc = 0xf00060bc$

Bits 32-63 of *SDRAM_DFII_PI3_RDDATA*.

SDRAM_DFII_PI3_RDDATA0

Address: $0xf0006000 + 0xc0 = 0xf00060c0$

Bits 0-31 of *SDRAM_DFII_PI3_RDDATA*.

SDRAM_CONTROLLER_TRP

Address: $0xf0006000 + 0xc4 = 0xf00060c4$

SDRAM_CONTROLLER_TRCD

Address: $0xf0006000 + 0xc8 = 0xf00060c8$

SDRAM_CONTROLLER_TWR

Address: $0xf0006000 + 0xcc = 0xf00060cc$

SDRAM_CONTROLLER_TWTR

Address: $0xf0006000 + 0xd0 = 0xf00060d0$

SDRAM_CONTROLLER_TREFI

Address: $0xf0006000 + 0xd4 = 0xf00060d4$

SDRAM_CONTROLLER_TRFC

Address: $0xf0006000 + 0xd8 = 0xf00060d8$

SDRAM_CONTROLLER_TFAW

Address: 0xf0006000 + 0xdc = 0xf00060dc

SDRAM_CONTROLLER_TCCD

Address: 0xf0006000 + 0xe0 = 0xf00060e0

SDRAM_CONTROLLER_TCCD_WR

Address: 0xf0006000 + 0xe4 = 0xf00060e4

SDRAM_CONTROLLER_TRTP

Address: 0xf0006000 + 0xe8 = 0xf00060e8

SDRAM_CONTROLLER_TRRD

Address: 0xf0006000 + 0xec = 0xf00060ec

SDRAM_CONTROLLER_TRC

Address: 0xf0006000 + 0xf0 = 0xf00060f0

SDRAM_CONTROLLER_TRAS

Address: 0xf0006000 + 0xf4 = 0xf00060f4

SDRAM_CONTROLLER_TZQCS

Address: $0xf0006000 + 0xf8 = 0xf00060f8$

SDRAM_CONTROLLER_LAST_ADDR_0

Address: $0xf0006000 + 0xfc = 0xf00060fc$

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_0

Address: $0xf0006000 + 0x100 = 0xf0006100$

SDRAM_CONTROLLER_LAST_ADDR_1

Address: $0xf0006000 + 0x104 = 0xf0006104$

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_1

Address: $0xf0006000 + 0x108 = 0xf0006108$

SDRAM_CONTROLLER_LAST_ADDR_2

Address: $0xf0006000 + 0x10c = 0xf000610c$

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_2

Address: $0xf0006000 + 0x110 = 0xf0006110$

SDRAM_CONTROLLER_LAST_ADDR_3

Address: 0xf0006000 + 0x114 = 0xf0006114

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_3

Address: 0xf0006000 + 0x118 = 0xf0006118

SDRAM_CONTROLLER_LAST_ADDR_4

Address: 0xf0006000 + 0x11c = 0xf000611c

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_4

Address: 0xf0006000 + 0x120 = 0xf0006120

SDRAM_CONTROLLER_LAST_ADDR_5

Address: 0xf0006000 + 0x124 = 0xf0006124

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_5

Address: 0xf0006000 + 0x128 = 0xf0006128

SDRAM_CONTROLLER_LAST_ADDR_6

Address: 0xf0006000 + 0x12c = 0xf000612c

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_6

Address: $0xf0006000 + 0x130 = 0xf0006130$

SDRAM_CONTROLLER_LAST_ADDR_7

Address: $0xf0006000 + 0x134 = 0xf0006134$

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_7

Address: $0xf0006000 + 0x138 = 0xf0006138$

13.2.14 SDRAM_CHECKER

Register Listing for SDRAM_CHECKER

Register	Address
<i>SDRAM_CHECKER_RESET</i>	<i>0xf0006800</i>
<i>SDRAM_CHECKER_START</i>	<i>0xf0006804</i>
<i>SDRAM_CHECKER_DONE</i>	<i>0xf0006808</i>
<i>SDRAM_CHECKER_BASE</i>	<i>0xf000680c</i>
<i>SDRAM_CHECKER_END</i>	<i>0xf0006810</i>
<i>SDRAM_CHECKER_LENGTH</i>	<i>0xf0006814</i>
<i>SDRAM_CHECKER_RANDOM</i>	<i>0xf0006818</i>
<i>SDRAM_CHECKER_TICKS</i>	<i>0xf000681c</i>
<i>SDRAM_CHECKER_ERRORS</i>	<i>0xf0006820</i>

SDRAM_CHECKER_RESET

Address: $0xf0006800 + 0x0 = 0xf0006800$

SDRAM_CHECKER_START

Address: $0xf0006800 + 0x4 = 0xf0006804$

SDRAM_CHECKER_DONE

Address: $0xf0006800 + 0x8 = 0xf0006808$

SDRAM_CHECKER_BASE

Address: $0xf0006800 + 0xc = 0xf000680c$

SDRAM_CHECKER_END

Address: $0xf0006800 + 0x10 = 0xf0006810$

SDRAM_CHECKER_LENGTH

Address: $0xf0006800 + 0x14 = 0xf0006814$

SDRAM_CHECKER_RANDOM

Address: $0xf0006800 + 0x18 = 0xf0006818$

Field	Name	Description

SDRAM_CHECKER_TICKS

Address: $0xf0006800 + 0x1c = 0xf000681c$

SDRAM_CHECKER_ERRORS

Address: $0xf0006800 + 0x20 = 0xf0006820$

13.2.15 SDRAM_GENERATOR

Register Listing for SDRAM_GENERATOR

Register	Address
<i>SDRAM_GENERATOR_RESET</i>	<i>0xf0007000</i>
<i>SDRAM_GENERATOR_START</i>	<i>0xf0007004</i>
<i>SDRAM_GENERATOR_DONE</i>	<i>0xf0007008</i>
<i>SDRAM_GENERATOR_BASE</i>	<i>0xf000700c</i>
<i>SDRAM_GENERATOR_END</i>	<i>0xf0007010</i>
<i>SDRAM_GENERATOR_LENGTH</i>	<i>0xf0007014</i>
<i>SDRAM_GENERATOR_RANDOM</i>	<i>0xf0007018</i>
<i>SDRAM_GENERATOR_TICKS</i>	<i>0xf000701c</i>

SDRAM_GENERATOR_RESET

Address: $0xf0007000 + 0x0 = 0xf0007000$

SDRAM_GENERATOR_START

Address: $0xf0007000 + 0x4 = 0xf0007004$

SDRAM_GENERATOR_DONE

Address: $0xf0007000 + 0x8 = 0xf0007008$

SDRAM_GENERATOR_BASE

Address: $0xf0007000 + 0xc = 0xf000700c$

SDRAM_GENERATOR_END

Address: $0xf0007000 + 0x10 = 0xf0007010$

SDRAM_GENERATOR_LENGTH

Address: $0xf0007000 + 0x14 = 0xf0007014$

SDRAM_GENERATOR_RANDOM

Address: $0xf0007000 + 0x18 = 0xf0007018$

Field	Name	Description

SDRAM_GENERATOR_TICKS

Address: $0xf0007000 + 0x1c = 0xf000701c$

13.2.16 TIMERO

Timer

Provides a generic Timer core.

The Timer is implemented as a countdown timer that can be used in various modes:

- Polling : Returns current countdown value to software
- One-Shot: Loads itself and stops when value reaches 0
- Periodic: (Re-)Loads itself when value reaches 0

en register allows the user to enable/disable the Timer. When the Timer is enabled, it is automatically loaded with the value of *load* register.

When the Timer reaches 0, it is automatically reloaded with value of *reload* register.

The user can latch the current countdown value by writing to *update_value* register, it will update *value* register with current countdown value.

To use the Timer in One-Shot mode, the user needs to:

- Disable the timer
- Set the load register to the expected duration
- (Re-)Enable the Timer

To use the Timer in Periodic mode, the user needs to:

- Disable the Timer
- Set the load register to 0
- Set the reload register to the expected period
- Enable the Timer

For both modes, the CPU can be advertised by an IRQ that the duration/period has elapsed. (The CPU can also do software polling with *update_value* and *value* to know the elapsed duration)

Register Listing for TIMERO

Register	Address
<i>TIMERO_LOAD</i>	0xf0007800
<i>TIMERO_RELOAD</i>	0xf0007804
<i>TIMERO_EN</i>	0xf0007808
<i>TIMERO_UPDATE_VALUE</i>	0xf000780c
<i>TIMERO_VALUE</i>	0xf0007810
<i>TIMERO_EV_STATUS</i>	0xf0007814
<i>TIMERO_EV_PENDING</i>	0xf0007818
<i>TIMERO_EV_ENABLE</i>	0xf000781c

TIMER0_LOAD

Address: $0xf0007800 + 0x0 = 0xf0007800$

Load value when Timer is (re-)enabled. In One-Shot mode, the value written to this register specifies the Timer's duration in clock cycles.

TIMER0_RELOAD

Address: $0xf0007800 + 0x4 = 0xf0007804$

Reload value when Timer reaches 0. In Periodic mode, the value written to this register specify the Timer's period in clock cycles.

TIMER0_EN

Address: $0xf0007800 + 0x8 = 0xf0007808$

Enable flag of the Timer. Set this flag to 1 to enable/start the Timer. Set to 0 to disable the Timer.

TIMER0_UPDATE_VALUE

Address: $0xf0007800 + 0xc = 0xf000780c$

Update trigger for the current countdown value. A write to this register latches the current countdown value to value register.

TIMER0_VALUE

Address: $0xf0007800 + 0x10 = 0xf0007810$

Latched countdown value. This value is updated by writing to update_value.

TIMER0_EV_STATUS

Address: $0xf0007800 + 0x14 = 0xf0007814$

This register contains the current raw level of the zero event trigger. Writes to this register have no effect.

Field	Name	Description
[0]	ZERO	Level of the zero event

TIMER0_EV_PENDING

Address: $0xf0007800 + 0x18 = 0xf0007818$

When a zero event occurs, the corresponding bit will be set in this register. To clear the Event, set the corresponding bit in this register.

Field	Name	Description
[0]	ZERO	1 if a zero event occurred. This Event is triggered on a falling edge.

TIMER0_EV_ENABLE

Address: $0xf0007800 + 0x1c = 0xf000781c$

This register enables the corresponding zero events. Write a 0 to this register to disable individual events.

Field	Name	Description
[0]	ZERO	Write a 1 to enable the zero Event

13.2.17 UART

Register Listing for UART

Register	Address
UART_RXTX	0xf0008000
UART_TXFULL	0xf0008004
UART_RXEMPTY	0xf0008008
UART_EV_STATUS	0xf000800c
UART_EV_PENDING	0xf0008010
UART_EV_ENABLE	0xf0008014
UART_TXEMPTY	0xf0008018
UART_RXFULL	0xf000801c
UART_XOVER_RXTX	0xf0008020
UART_XOVER_TXFULL	0xf0008024
UART_XOVER_RXEMPTY	0xf0008028
UART_XOVER_EV_STATUS	0xf000802c
UART_XOVER_EV_PENDING	0xf0008030
UART_XOVER_EV_ENABLE	0xf0008034
UART_XOVER_TXEMPTY	0xf0008038
UART_XOVER_RXFULL	0xf000803c

UART_RXTX

Address: $0xf0008000 + 0x0 = 0xf0008000$

UART_TXFULL

Address: $0xf0008000 + 0x4 = 0xf0008004$

TX FIFO Full.

UART_RXEMPTY

Address: $0xf0008000 + 0x8 = 0xf0008008$

RX FIFO Empty.

UART_EV_STATUS

Address: $0xf0008000 + 0xc = 0xf000800c$

This register contains the current raw level of the rx event trigger. Writes to this register have no effect.

Field	Name	Description
[0]	TX	Level of the tx event
[1]	RX	Level of the rx event

UART_EV_PENDING

Address: $0xf0008000 + 0x10 = 0xf0008010$

When a rx event occurs, the corresponding bit will be set in this register. To clear the Event, set the corresponding bit in this register.

Field	Name	Description
[0]	TX	1 if a tx event occurred. This Event is triggered on a falling edge.
[1]	RX	1 if a rx event occurred. This Event is triggered on a falling edge.

UART_EV_ENABLE

Address: $0xf0008000 + 0x14 = 0xf0008014$

This register enables the corresponding rx events. Write a 0 to this register to disable individual events.

Field	Name	Description
[0]	TX	Write a 1 to enable the tx Event
[1]	RX	Write a 1 to enable the rx Event

UART_TXEMPTY

Address: $0xf0008000 + 0x18 = 0xf0008018$

TX FIFO Empty.

UART_RXFULL

Address: $0xf0008000 + 0x1c = 0xf000801c$

RX FIFO Full.

UART_XOVER_RXTX

Address: $0xf0008000 + 0x20 = 0xf0008020$

UART_XOVER_TXFULL

Address: $0xf0008000 + 0x24 = 0xf0008024$

TX FIFO Full.

UART_XOVER_RXEMPTY

Address: $0xf0008000 + 0x28 = 0xf0008028$

RX FIFO Empty.

UART_XOVER_EV_STATUS

Address: $0xf0008000 + 0x2c = 0xf000802c$

This register contains the current raw level of the rx event trigger. Writes to this register have no effect.

Field	Name	Description
[0]	TX	Level of the tx event
[1]	RX	Level of the rx event

UART_XOVER_EV_PENDING

Address: $0xf0008000 + 0x30 = 0xf0008030$

When a rx event occurs, the corresponding bit will be set in this register. To clear the Event, set the corresponding bit in this register.

Field	Name	Description
[0]	TX	1 if a tx event occurred. This Event is triggered on a falling edge.
[1]	RX	1 if a rx event occurred. This Event is triggered on a falling edge.

UART_XOVER_EV_ENABLE

Address: $0xf0008000 + 0x34 = 0xf0008034$

This register enables the corresponding rx events. Write a 0 to this register to disable individual events.

Field	Name	Description
[0]	TX	Write a 1 to enable the tx Event
[1]	RX	Write a 1 to enable the rx Event

UART_XOVER_TXEMPTY

Address: $0xf0008000 + 0x38 = 0xf0008038$

TX FIFO Empty.

UART_XOVER_RXFULL

Address: $0xf0008000 + 0x3c = 0xf000803c$

RX FIFO Full.

CHAPTER
FOURTEEN

DOCUMENTATION FOR ROW HAMMER TESTER DATA CENTER
DRAM TESTER

14.1 Modules

14.1.1 Interrupt Controller

This device has an EventManager-based interrupt system. Individual modules generate *events* which are wired into a central interrupt controller.

When an interrupt occurs, you should look the interrupt number up in the CPU- specific interrupt table and then call the relevant module.

Assigned Interrupts

The following interrupts are assigned on this system:

Interrupt	Module
1	<i>TIMER0</i>
0	<i>UART</i>

14.2 Register Groups

14.2.1 LEDS

Register Listing for LEDS

Register	Address
<i>LEDS_OUT</i>	<i>0xf0000000</i>

LEDS_OUT

Address: $0xf0000000 + 0x0 = 0xf0000000$

Led Output(s) Control.

14.2.2 DDRPHY

Register Listing for DDRPHY

Register	Address
<i>DDRPHY_RST</i>	<i>0xf0000800</i>
<i>DDRPHY_DLY_SEL</i>	<i>0xf0000804</i>
<i>DDRPHY_HALF_SYS8X_TAPS</i>	<i>0xf0000808</i>
<i>DDRPHY_WLEVEL_EN</i>	<i>0xf000080c</i>
<i>DDRPHY_WLEVEL_STROBE</i>	<i>0xf0000810</i>
<i>DDRPHY_CDLY_RST</i>	<i>0xf0000814</i>
<i>DDRPHY_CDLY_INC</i>	<i>0xf0000818</i>
<i>DDRPHY_RDLY_DQ_RST</i>	<i>0xf000081c</i>
<i>DDRPHY_RDLY_DQ_INC</i>	<i>0xf0000820</i>
<i>DDRPHY_RDLY_DQ_BITSLIP_RST</i>	<i>0xf0000824</i>
<i>DDRPHY_RDLY_DQ_BITSLIP</i>	<i>0xf0000828</i>
<i>DDRPHY_WDLY_DQ_RST</i>	<i>0xf000082c</i>
<i>DDRPHY_WDLY_DQ_INC</i>	<i>0xf0000830</i>
<i>DDRPHY_WDLY_DQS_RST</i>	<i>0xf0000834</i>
<i>DDRPHY_WDLY_DQS_INC</i>	<i>0xf0000838</i>
<i>DDRPHY_WDLY_DQ_BITSLIP_RST</i>	<i>0xf000083c</i>
<i>DDRPHY_WDLY_DQ_BITSLIP</i>	<i>0xf0000840</i>
<i>DDRPHY_RDPHASE</i>	<i>0xf0000844</i>
<i>DDRPHY_WRPHASE</i>	<i>0xf0000848</i>
<i>DDRPHY_ALERT</i>	<i>0xf000084c</i>
<i>DDRPHY_RST_ALERT</i>	<i>0xf0000850</i>

DDRPHY_RST

Address: $0xf0000800 + 0x0 = 0xf0000800$

DDRPHY_DLY_SEL

Address: $0xf0000800 + 0x4 = 0xf0000804$

DDRPHY_HALF_SYS8X_TAPS

Address: $0xf0000800 + 0x8 = 0xf0000808$

DDRPHY_WLEVEL_EN

Address: $0xf0000800 + 0xc = 0xf000080c$

DDRPHY_WLEVEL_STROBE

Address: $0xf0000800 + 0x10 = 0xf0000810$

DDRPHY_CDLY_RST

Address: $0xf0000800 + 0x14 = 0xf0000814$

DDRPHY_CDLY_INC

Address: $0xf0000800 + 0x18 = 0xf0000818$

DDRPHY_RDLY_DQ_RST

Address: $0xf0000800 + 0x1c = 0xf000081c$

DDRPHY_RDLY_DQ_INC

Address: $0xf0000800 + 0x20 = 0xf0000820$

DDRPHY_RDLY_DQ_BITSLIP_RST

Address: $0xf0000800 + 0x24 = 0xf0000824$

DDRPHY_RDLY_DQ_BITSLIP

Address: $0xf0000800 + 0x28 = 0xf0000828$

DDRPHY_WDLY_DQ_RST

Address: $0xf0000800 + 0x2c = 0xf000082c$

DDRPHY_WDLY_DQ_INC

Address: $0xf0000800 + 0x30 = 0xf0000830$

DDRPHY_WDLY_DQS_RST

Address: $0xf0000800 + 0x34 = 0xf0000834$

DDRPHY_WDLY_DQS_INC

Address: $0xf0000800 + 0x38 = 0xf0000838$

DDRPHY_WDLY_DQ_BITSLIP_RST

Address: $0xf0000800 + 0x3c = 0xf000083c$

DDRPHY_WDLY_DQ_BITSLIP

Address: $0xf0000800 + 0x40 = 0xf0000840$

DDRPHY_RDPHASE

Address: $0xf0000800 + 0x44 = 0xf0000844$

DDRPHY_WRPHASE

Address: $0xf0000800 + 0x48 = 0xf0000848$

DDRPHY_ALERT

Address: $0xf0000800 + 0x4c = 0xf000084c$

DDRPHY_RST_ALERT

Address: $0xf0000800 + 0x50 = 0xf0000850$

14.2.3 CONTROLLER_SETTINGS

Allows to change LiteDRAMController behaviour at runtime

Register Listing for CONTROLLER_SETTINGS

Register	Address
<i>CONTROLLER_SETTINGS_REFRESH</i>	<i>0xf0001000</i>

CONTROLLER_SETTINGS_REFRESH

Address: $0xf0001000 + 0x0 = 0xf0001000$

Enable/disable Refresh commands sending

14.2.4 DDRCTRL

Register Listing for DDRCTRL

Register	Address
<i>DDRCTRL_INIT_DONE</i>	<i>0xf0001800</i>
<i>DDRCTRL_INIT_ERROR</i>	<i>0xf0001804</i>

DDRCTRL_INIT_DONE

Address: $0xf0001800 + 0x0 = 0xf0001800$

DDRCTRL_INIT_ERROR

Address: $0xf0001800 + 0x4 = 0xf0001804$

14.2.5 ROWHAMMER

Row Hammer DMA attacker

This module allows to perform a Row Hammer attack by configuring it with two addresses that map to different rows of a single bank. When enabled, it will perform alternating DMA reads from the given locations, which will result in the DRAM controller having to repeatedly open/close rows at each read access.

Register Listing for ROWHAMMER

Register	Address
<code>ROWHAMMER_ENABLED</code>	<code>0xf0002000</code>
<code>ROWHAMMER_ADDRESS1</code>	<code>0xf0002004</code>
<code>ROWHAMMER_ADDRESS2</code>	<code>0xf0002008</code>
<code>ROWHAMMER_COUNT</code>	<code>0xf000200c</code>

`ROWHAMMER_ENABLED`

Address: $0xf0002000 + 0x0 = 0xf0002000$

Used to start/stop the operation of the module

`ROWHAMMER_ADDRESS1`

Address: $0xf0002000 + 0x4 = 0xf0002004$

First attacked address

`ROWHAMMER_ADDRESS2`

Address: $0xf0002000 + 0x8 = 0xf0002008$

Second attacked address

`ROWHAMMER_COUNT`

Address: $0xf0002000 + 0xc = 0xf000200c$

This is the number of DMA accesses performed. When the module is enabled, the value can be freely read. When the module is disabled, the register is clear-on-write and has to be read before the next attack.

14.2.6 WRITER

DMA DRAM writer.

Allows to fill DRAM with a predefined pattern using DMA.

Pattern

Provides access to RAM to store access pattern: *mem_addr* and *mem_data*. The pattern address space can be limited using the *data_mask*.

For example, having *mem_addr* filled with [0x04, 0x02, 0x03, ...] and *mem_data* filled with [0xff, 0xaa, 0x55, ...] and setting *data_mask* = 0b01, the pattern [(address, data), ...] written will be: [(0x04, 0xff), (0x02, 0xaa), (0x04, 0xff), ...] (wraps due to masking).

DRAM memory range that is being accessed can be configured using *mem_mask*.

To use this module, make sure that *ready* is 1, then write the desired number of transfers to *count*. Writing to the *start* CSR will initialize the operation. When the operation is ongoing *ready* will be 0.

Register Listing for WRITER

Register	Address
WRITER_START	0xf0002800
WRITER_READY	0xf0002804
WRITER_MODULO	0xf0002808
WRITER_COUNT	0xf000280c
WRITER_DONE	0xf0002810
WRITER_MEM_MASK	0xf0002814
WRITER_DATA_MASK	0xf0002818
WRITER_DATA_DIV	0xf000281c
WRITER_INVERTER_DIVISOR_MASK	0xf0002820
WRITER_INVERTER_SELECTION_MASK	0xf0002824
WRITER_LAST_ADDRESS	0xf0002828

WRITER_START

Address: 0xf0002800 + 0x0 = 0xf0002800

Write to the register starts the transfer (if ready=1)

WRITER_READY

Address: 0xf0002800 + 0x4 = 0xf0002804

Indicates that the transfer is not ongoing

WRITER_MODULO

Address: $0xf0002800 + 0x8 = 0xf0002808$

When set use modulo to calculate DMA transfers address rather than bit masking

WRITER_COUNT

Address: $0xf0002800 + 0xc = 0xf000280c$

Desired number of DMA transfers

WRITER_DONE

Address: $0xf0002800 + 0x10 = 0xf0002810$

Number of completed DMA transfers

WRITER_MEM_MASK

Address: $0xf0002800 + 0x14 = 0xf0002814$

DRAM address mask for DMA transfers

WRITER_DATA_MASK

Address: $0xf0002800 + 0x18 = 0xf0002818$

Pattern memory address mask

WRITER_DATA_DIV

Address: $0xf0002800 + 0x1c = 0xf000281c$

Pattern memory address divisor-1

WRITER_INVERTER_DIVISOR_MASK

Address: $0xf0002800 + 0x20 = 0xf0002820$

Divisor mask for selecting rows for which pattern data gets inverted

WRITER_INVERTER_SELECTION_MASK

Address: $0xf0002800 + 0x24 = 0xf0002824$

Selection mask for selecting rows for which pattern data gets inverted

WRITER_LAST_ADDRESS

Address: $0xf0002800 + 0x28 = 0xf0002828$

Number of completed DMA transfers

14.2.7 READER

DMA DRAM reader.

Allows to check DRAM contents against a predefined pattern using DMA.

Pattern

Provides access to RAM to store access pattern: *mem_addr* and *mem_data*. The pattern address space can be limited using the *data_mask*.

For example, having *mem_addr* filled with [0x04, 0x02, 0x03, ...] and *mem_data* filled with [0xff, 0xaa, 0x55, ...] and setting *data_mask* = 0b01, the pattern [(address, data), ...] written will be: [(0x04, 0xff), (0x02, 0xaa), (0x04, 0xff), ...] (wraps due to masking).

DRAM memory range that is being accessed can be configured using *mem_mask*.

To use this module, make sure that *ready* is 1, then write the desired number of transfers to *count*. Writing to the *start* CSR will initialize the operation. When the operation is ongoing *ready* will be 0.

Reading errors

This module allows to check the locations of errors in the memory. It scans the configured memory area and compares the values read to the predefined pattern. If *skip_fifo* is 0, this module will stop after each error encountered, so that it can be examined. Wait until the *error_ready* CSR is 1. Then use the CSRs *error_offset*, *error_data* and *error_expected* to examine the errors in the current transfer. To continue reading, write 1 to *error_continue* CSR. Setting *skip_fifo* to 1 will disable this behaviour entirely.

The final number of errors can be read from *error_count*. NOTE: This value represents the number of erroneous DMA transfers.

The current progress can be read from the *done* CSR.

Register Listing for READER

Register	Address
READER_START	0xf0003000
READER_READY	0xf0003004
READER_MODULO	0xf0003008
READER_COUNT	0xf000300c
READER_DONE	0xf0003010
READER_MEM_MASK	0xf0003014
READER_DATA_MASK	0xf0003018
READER_DATA_DIV	0xf000301c
READER_INVERTER_DIVISOR_MASK	0xf0003020
READER_INVERTER_SELECTION_MASK	0xf0003024
READER_ERROR_COUNT	0xf0003028
READER_SKIP_FIFO	0xf000302c
READER_ERROR_OFFSET	0xf0003030
READER_ERROR_DATA7	0xf0003034
READER_ERROR_DATA6	0xf0003038
READER_ERROR_DATA5	0xf000303c
READER_ERROR_DATA4	0xf0003040
READER_ERROR_DATA3	0xf0003044
READER_ERROR_DATA2	0xf0003048
READER_ERROR_DATA1	0xf000304c
READER_ERROR_DATA0	0xf0003050
READER_ERROR_EXPECTED7	0xf0003054
READER_ERROR_EXPECTED6	0xf0003058
READER_ERROR_EXPECTED5	0xf000305c
READER_ERROR_EXPECTED4	0xf0003060
READER_ERROR_EXPECTED3	0xf0003064
READER_ERROR_EXPECTED2	0xf0003068
READER_ERROR_EXPECTED1	0xf000306c
READER_ERROR_EXPECTED0	0xf0003070
READER_ERROR_READY	0xf0003074
READER_ERROR_CONTINUE	0xf0003078

READER_START

Address: 0xf0003000 + 0x0 = 0xf0003000

Write to the register starts the transfer (if ready=1)

READER_READY

Address: $0xf0003000 + 0x4 = 0xf0003004$

Indicates that the transfer is not ongoing

READER_MODULO

Address: $0xf0003000 + 0x8 = 0xf0003008$

When set use modulo to calculate DMA transfers address rather than bit masking

READER_COUNT

Address: $0xf0003000 + 0xc = 0xf000300c$

Desired number of DMA transfers

READER_DONE

Address: $0xf0003000 + 0x10 = 0xf0003010$

Number of completed DMA transfers

READER_MEM_MASK

Address: $0xf0003000 + 0x14 = 0xf0003014$

DRAM address mask for DMA transfers

READER_DATA_MASK

Address: $0xf0003000 + 0x18 = 0xf0003018$

Pattern memory address mask

READER_DATA_DIV

Address: $0xf0003000 + 0x1c = 0xf000301c$

Pattern memory address divisor-1

READER_INVERTER_DIVISOR_MASK

Address: $0xf0003000 + 0x20 = 0xf0003020$

Divisor mask for selecting rows for which pattern data gets inverted

READER_INVERTER_SELECTION_MASK

Address: $0xf0003000 + 0x24 = 0xf0003024$

Selection mask for selecting rows for which pattern data gets inverted

READER_ERROR_COUNT

Address: $0xf0003000 + 0x28 = 0xf0003028$

Number of errors detected

READER_SKIP_FIFO

Address: $0xf0003000 + 0x2c = 0xf000302c$

Skip waiting for user to read the errors FIFO

READER_ERROR_OFFSET

Address: $0xf0003000 + 0x30 = 0xf0003030$

Current offset of the error

READER_ERROR_DATA7

Address: $0xf0003000 + 0x34 = 0xf0003034$

Bits 224-255 of *READER_ERROR_DATA*. Erroneous value read from DRAM memory

READER_ERROR_DATA6

Address: $0xf0003000 + 0x38 = 0xf0003038$

Bits 192-223 of *READER_ERROR_DATA*.

READER_ERROR_DATA5

Address: $0xf0003000 + 0x3c = 0xf000303c$

Bits 160-191 of *READER_ERROR_DATA*.

READER_ERROR_DATA4

Address: $0xf0003000 + 0x40 = 0xf0003040$

Bits 128-159 of *READER_ERROR_DATA*.

READER_ERROR_DATA3

Address: $0xf0003000 + 0x44 = 0xf0003044$

Bits 96-127 of *READER_ERROR_DATA*.

READER_ERROR_DATA2

Address: $0xf0003000 + 0x48 = 0xf0003048$

Bits 64-95 of *READER_ERROR_DATA*.

READER_ERROR_DATA1

Address: $0xf0003000 + 0x4c = 0xf000304c$

Bits 32-63 of *READER_ERROR_DATA*.

READER_ERROR_DATA0

Address: $0xf0003000 + 0x50 = 0xf0003050$

Bits 0-31 of *READER_ERROR_DATA*.

READER_ERROR_EXPECTED7

Address: $0xf0003000 + 0x54 = 0xf0003054$

Bits 224-255 of *READER_ERROR_EXPECTED*. Value expected to be read from DRAM memory

READER_ERROR_EXPECTED6

Address: $0xf0003000 + 0x58 = 0xf0003058$

Bits 192-223 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED5

Address: $0xf0003000 + 0x5c = 0xf000305c$

Bits 160-191 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED4

Address: $0xf0003000 + 0x60 = 0xf0003060$

Bits 128-159 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED3

Address: $0xf0003000 + 0x64 = 0xf0003064$

Bits 96-127 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED2

Address: $0xf0003000 + 0x68 = 0xf0003068$

Bits 64-95 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED1

Address: $0xf0003000 + 0x6c = 0xf000306c$

Bits 32-63 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED0

Address: $0xf0003000 + 0x70 = 0xf0003070$

Bits 0-31 of *READER_ERROR_EXPECTED*.

READER_ERROR_READY

Address: $0xf0003000 + 0x74 = 0xf0003074$

Error detected and ready to read

READER_ERROR_CONTINUE

Address: $0xf0003000 + 0x78 = 0xf0003078$

Continue reading until the next error

14.2.8 DFI_SWITCH

Register Listing for DFI_SWITCH

Register	Address
<i>DFI_SWITCH_REFRESH_COUNT</i>	<i>0xf0003800</i>
<i>DFI_SWITCH_AT_REFRESH</i>	<i>0xf0003804</i>
<i>DFI_SWITCH_REFRESH_UPDATE</i>	<i>0xf0003808</i>

DFI_SWITCH_REFRESH_COUNT

Address: $0xf0003800 + 0x0 = 0xf0003800$

Count of all refresh commands issued (both by Memory Controller and Payload Executor). Value is latched from internal counter on mode transition: MC -> PE or by writing to the *refresh_update* CSR.

DFI_SWITCH_AT_REFRESH

Address: $0xf0003800 + 0x4 = 0xf0003804$

If set to a value different than 0 the mode transition MC -> PE will be performed only when the value of this register matches the current refresh commands count.

DFI_SWITCH_REFRESH_UPDATE

Address: $0xf0003800 + 0x8 = 0xf0003808$

Force an update of the *refresh_count* CSR.

14.2.9 PAYLOAD_EXECUTOR

Executes the DRAM payload from memory

Instruction decoder

All instructions are 32-bit. The format of most instructions is the same, except for the LOOP instruction, which has a constant TIMESLICE of 1.

NOOP with a TIMESLICE of 0 is a special case which is interpreted as STOP instruction. When this instruction is encountered execution gets finished immediately.

NOTE: TIMESLICE is the number of cycles the instruction will take. This means that instructions other than NOOP that use TIMESLICE=0 are illegal (although will silently be executed as having TIMESLICE=1).

NOTE2: LOOP instruction will *jump* COUNT times, meaning that the “code” inside the loop will effectively be executed COUNT+1 times.

Op codes:

Op	Value
NOOP	0b000
LOOP	0b111
ACT	0b100
PRE	0b101
REF	0b110
ZQC	0b001
READ	0b010

Instruction format:

LSB	MSB
dfi: OP_CODE TIMESLICE ADDRESS	
noop: OP_CODE TIMESLICE_NOOP	
loop: OP_CODE COUNT JUMP	
stop: <NOOP> 0	

Where ADDRESS depends on the DFI command and is one of:

LSB	MSB
RANK BANK COLUMN	
RANK BANK ROW	

Register Listing for PAYLOAD_EXECUTOR

Register	Address
PAYLOAD_EXECUTOR_START	0xf0004000
PAYLOAD_EXECUTOR_STATUS	0xf0004004
PAYLOAD_EXECUTOR_READ_COUNT	0xf0004008

PAYLOAD_EXECUTOR_START

Address: $0xf0004000 + 0x0 = 0xf0004000$

Writing to this register initializes payload execution

PAYLOAD_EXECUTOR_STATUS

Address: $0xf0004000 + 0x4 = 0xf0004004$

Payload executor status register

Field	Name	Description
[0]	READY	Indicates that the executor is not running
[1]	OVERFLOW	Indicates the scratchpad memory address counter has overflowed due to the number of READ commands sent during execution

PAYLOAD_EXECUTOR_READ_COUNT

Address: $0xf0004000 + 0x8 = 0xf0004008$

Number of data from READ commands that is stored in the scratchpad memory

14.2.10 I2C

Register Listing for I2C

Register	Address
I2C_W	0xf0004800
I2C_R	0xf0004804

I2C_W

Address: $0xf0004800 + 0x0 = 0xf0004800$

Field	Name	Description

I2C_R

Address: $0xf0004800 + 0x4 = 0xf0004804$

Field	Name	Description

14.2.11 CTRL

Register Listing for CTRL

Register	Address
<i>CTRL_RESET</i>	<i>0xf0005000</i>
<i>CTRL_SCRATCH</i>	<i>0xf0005004</i>
<i>CTRL_BUS_ERRORS</i>	<i>0xf0005008</i>

CTRL__RESET

Address: $0xf0005000 + 0x0 = 0xf0005000$

Field	Name	Description
[0]	SOC_RST	Write 1 to this register to reset the full SoC (Pulse Reset)
[1]	CPU_RST	Write 1 to this register to reset the CPU(s) of the SoC (Hold Reset)

CTRL_SCRATCH

Address: $0xf0005000 + 0x4 = 0xf0005004$

Use this register as a scratch space to verify that software read/write accesses to the Wishbone/CSR bus are working correctly. The initial reset value of 0x1234578 can be used to verify endianness.

CTRL_BUS_ERRORS

Address: $0xf0005000 + 0x8 = 0xf0005008$

Total number of Wishbone bus errors (timeouts) since start.

14.2.12 ETPHY

Register Listing for ETPHY

Register	Address
<i>ETPHY_CRG_RESET</i>	<i>0xf0005800</i>
<i>ETPHY_MDIO_W</i>	<i>0xf0005804</i>
<i>ETPHY_MDIO_R</i>	<i>0xf0005808</i>

ETPHY_CRG_RESET

Address: $0xf0005800 + 0x0 = 0xf0005800$

ETPHY_MDIO_W

Address: $0xf0005800 + 0x4 = 0xf0005804$

Field	Name	Description

ETHPHY_MDIO_R

Address: $0xf0005800 + 0x8 = 0xf0005808$

Field	Name	Description

14.2.13 IDENTIFIER_MEM

Register Listing for IDENTIFIER_MEM

Register	Address
<i>IDENTIFIER_MEM</i>	<i>0xf0006000</i>

IDENTIFIER_MEM

Address: $0xf0006000 + 0x0 = 0xf0006000$

8 x 109-bit memory

14.2.14 SDRAM

Register Listing for SDRAM

Register	Address
<i>SDRAM_DFII_CONTROL</i>	<i>0xf0006800</i>
<i>SDRAM_DFII_PIO_COMMAND</i>	<i>0xf0006804</i>
<i>SDRAM_DFII_PIO_COMMAND_ISSUE</i>	<i>0xf0006808</i>
<i>SDRAM_DFII_PIO_ADDRESS</i>	<i>0xf000680c</i>
<i>SDRAM_DFII_PIO_BADDRESS</i>	<i>0xf0006810</i>
<i>SDRAM_DFII_PIO_WRDATA1</i>	<i>0xf0006814</i>
<i>SDRAM_DFII_PIO_WRDATA0</i>	<i>0xf0006818</i>
<i>SDRAM_DFII_PIO_RDDATA1</i>	<i>0xf000681c</i>
<i>SDRAM_DFII_PIO_RDDATA0</i>	<i>0xf0006820</i>
<i>SDRAM_DFII_PI1_COMMAND</i>	<i>0xf0006824</i>
<i>SDRAM_DFII_PI1_COMMAND_ISSUE</i>	<i>0xf0006828</i>
<i>SDRAM_DFII_PI1_ADDRESS</i>	<i>0xf000682c</i>
<i>SDRAM_DFII_PI1_BADDRESS</i>	<i>0xf0006830</i>
<i>SDRAM_DFII_PI1_WRDATA1</i>	<i>0xf0006834</i>
<i>SDRAM_DFII_PI1_WRDATA0</i>	<i>0xf0006838</i>
<i>SDRAM_DFII_PI1_RDDATA1</i>	<i>0xf000683c</i>
<i>SDRAM_DFII_PI1_RDDATA0</i>	<i>0xf0006840</i>

continues on next page

Table 14.2 – continued from previous page

Register	Address
<i>SDRAM_DFII_PI2_COMMAND</i>	0xf0006844
<i>SDRAM_DFII_PI2_COMMAND_ISSUE</i>	0xf0006848
<i>SDRAM_DFII_PI2_ADDRESS</i>	0xf000684c
<i>SDRAM_DFII_PI2_BADDRESS</i>	0xf0006850
<i>SDRAM_DFII_PI2_WRDATA1</i>	0xf0006854
<i>SDRAM_DFII_PI2_WRDATA0</i>	0xf0006858
<i>SDRAM_DFII_PI2_RDDATA1</i>	0xf000685c
<i>SDRAM_DFII_PI2_RDDATA0</i>	0xf0006860
<i>SDRAM_DFII_PI3_COMMAND</i>	0xf0006864
<i>SDRAM_DFII_PI3_COMMAND_ISSUE</i>	0xf0006868
<i>SDRAM_DFII_PI3_ADDRESS</i>	0xf000686c
<i>SDRAM_DFII_PI3_BADDRESS</i>	0xf0006870
<i>SDRAM_DFII_PI3_WRDATA1</i>	0xf0006874
<i>SDRAM_DFII_PI3_WRDATA0</i>	0xf0006878
<i>SDRAM_DFII_PI3_RDDATA1</i>	0xf000687c
<i>SDRAM_DFII_PI3_RDDATA0</i>	0xf0006880
<i>SDRAM_CONTROLLER_TRP</i>	0xf0006884
<i>SDRAM_CONTROLLER_TRCD</i>	0xf0006888
<i>SDRAM_CONTROLLER_TWR</i>	0xf000688c
<i>SDRAM_CONTROLLER_TWTR</i>	0xf0006890
<i>SDRAM_CONTROLLER_TREFI</i>	0xf0006894
<i>SDRAM_CONTROLLER_TRFC</i>	0xf0006898
<i>SDRAM_CONTROLLER_TFAW</i>	0xf000689c
<i>SDRAM_CONTROLLER_TCCD</i>	0xf00068a0
<i>SDRAM_CONTROLLER_TCCD_WR</i>	0xf00068a4
<i>SDRAM_CONTROLLER_TRTP</i>	0xf00068a8
<i>SDRAM_CONTROLLER_TRRD</i>	0xf00068ac
<i>SDRAM_CONTROLLER_TRC</i>	0xf00068b0
<i>SDRAM_CONTROLLER_TRAS</i>	0xf00068b4
<i>SDRAM_CONTROLLER_TZQCS</i>	0xf00068b8
<i>SDRAM_CONTROLLER_LAST_ADDR_0</i>	0xf00068bc
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_0</i>	0xf00068c0
<i>SDRAM_CONTROLLER_LAST_ADDR_1</i>	0xf00068c4
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_1</i>	0xf00068c8
<i>SDRAM_CONTROLLER_LAST_ADDR_2</i>	0xf00068cc
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_2</i>	0xf00068d0
<i>SDRAM_CONTROLLER_LAST_ADDR_3</i>	0xf00068d4
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_3</i>	0xf00068d8
<i>SDRAM_CONTROLLER_LAST_ADDR_4</i>	0xf00068dc
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_4</i>	0xf00068e0
<i>SDRAM_CONTROLLER_LAST_ADDR_5</i>	0xf00068e4
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_5</i>	0xf00068e8
<i>SDRAM_CONTROLLER_LAST_ADDR_6</i>	0xf00068ec
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_6</i>	0xf00068f0
<i>SDRAM_CONTROLLER_LAST_ADDR_7</i>	0xf00068f4
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_7</i>	0xf00068f8
<i>SDRAM_CONTROLLER_LAST_ADDR_8</i>	0xf00068fc
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_8</i>	0xf0006900

continues on next page

Table 14.2 – continued from previous page

Register	Address
<i>SDRAM_CONTROLLER_LAST_ADDR_9</i>	0xf0006904
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_9</i>	0xf0006908
<i>SDRAM_CONTROLLER_LAST_ADDR_10</i>	0xf000690c
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_10</i>	0xf0006910
<i>SDRAM_CONTROLLER_LAST_ADDR_11</i>	0xf0006914
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_11</i>	0xf0006918
<i>SDRAM_CONTROLLER_LAST_ADDR_12</i>	0xf000691c
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_12</i>	0xf0006920
<i>SDRAM_CONTROLLER_LAST_ADDR_13</i>	0xf0006924
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_13</i>	0xf0006928
<i>SDRAM_CONTROLLER_LAST_ADDR_14</i>	0xf000692c
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_14</i>	0xf0006930
<i>SDRAM_CONTROLLER_LAST_ADDR_15</i>	0xf0006934
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_15</i>	0xf0006938

SDRAM_DFII_CONTROL

Address: 0xf0006800 + 0x0 = 0xf0006800

Control DFI signals common to all phases

Field	Name	Description							
[0]	SEL	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>0b0</td><td>Software control. (CPU)</td></tr> <tr> <td>0b1</td><td>Hardware control (default).</td></tr> </tbody> </table>		Value	Description	0b0	Software control. (CPU)	0b1	Hardware control (default).
Value	Description								
0b0	Software control. (CPU)								
0b1	Hardware control (default).								
[1]	CKE	DFI clock enable bus							
[2]	ODT	DFI on-die termination bus							
[3]	RESET_N	DFI clock reset bus							

SDRAM_DFII_PIO_COMMAND

Address: 0xf0006800 + 0x4 = 0xf0006804

Control DFI signals on a single phase

Field	Name	Description
[0]	CS	DFI chip select bus
[1]	WE	DFI write enable bus
[2]	CAS	DFI column address strobe bus
[3]	RAS	DFI row address strobe bus
[4]	WREN	DFI write data enable bus
[5]	RDEN	DFI read data enable bus

SDRAM_DFII_PIO_COMMAND_ISSUE

Address: $0xf0006800 + 0x8 = 0xf0006808$

SDRAM_DFII_PIO_ADDRESS

Address: $0xf0006800 + 0xc = 0xf000680c$

DFI address bus

SDRAM_DFII_PIO_BADDRESS

Address: $0xf0006800 + 0x10 = 0xf0006810$

DFI bank address bus

SDRAM_DFII_PIO_WRDATA1

Address: $0xf0006800 + 0x14 = 0xf0006814$

Bits 32-63 of *SDRAM_DFII_PIO_WRDATA*. DFI write data bus

SDRAM_DFII_PIO_WRDATA0

Address: $0xf0006800 + 0x18 = 0xf0006818$

Bits 0-31 of *SDRAM_DFII_PIO_WRDATA*.

SDRAM_DFII PIO RDDATA1

Address: 0xf0006800 + 0x1c = 0xf000681c

Bits 32-63 of *SDRAM_DFII_PIO_RDDATA*. DFI read data bus

SDRAM_DFII PIO RDDATA0

Address: 0xf0006800 + 0x20 = 0xf0006820

Bits 0-31 of *SDRAM_DFII_PIO_RDDATA*.

SDRAM_DFII PI1 COMMAND

Address: 0xf0006800 + 0x24 = 0xf0006824

Control DFI signals on a single phase

Field	Name	Description
[0]	CS	DFI chip select bus
[1]	WE	DFI write enable bus
[2]	CAS	DFI column address strobe bus
[3]	RAS	DFI row address strobe bus
[4]	WREN	DFI write data enable bus
[5]	RDEN	DFI read data enable bus

SDRAM_DFII PI1 COMMAND_ISSUE

Address: 0xf0006800 + 0x28 = 0xf0006828

SDRAM_DFII PI1 ADDRESS

Address: 0xf0006800 + 0x2c = 0xf000682c

DFI address bus

SDRAM_DFII PI1 BADDRESS

Address: 0xf0006800 + 0x30 = 0xf0006830

DFI bank address bus

SDRAM_DFII_PI1_WRDATA1

Address: 0xf0006800 + 0x34 = 0xf0006834

Bits 32-63 of *SDRAM_DFII_PI1_WRDATA*. DFI write data bus

SDRAM_DFII_PI1_WRDATA0

Address: 0xf0006800 + 0x38 = 0xf0006838

Bits 0-31 of *SDRAM_DFII_PI1_WRDATA*.

SDRAM_DFII_PI1_RDDATA1

Address: 0xf0006800 + 0x3c = 0xf000683c

Bits 32-63 of *SDRAM_DFII_PI1_RDDATA*. DFI read data bus

SDRAM_DFII_PI1_RDDATA0

Address: 0xf0006800 + 0x40 = 0xf0006840

Bits 0-31 of *SDRAM_DFII_PI1_RDDATA*.

SDRAM_DFII_PI2_COMMAND

Address: 0xf0006800 + 0x44 = 0xf0006844

Control DFI signals on a single phase

Field	Name	Description
[0]	CS	DFI chip select bus
[1]	WE	DFI write enable bus
[2]	CAS	DFI column address strobe bus
[3]	RAS	DFI row address strobe bus
[4]	WREN	DFI write data enable bus
[5]	RDEN	DFI read data enable bus

SDRAM_DFII_PI2_COMMAND_ISSUE

Address: 0xf0006800 + 0x48 = 0xf0006848

SDRAM_DFII_PI2_ADDRESS

Address: 0xf0006800 + 0x4c = 0xf000684c

DFI address bus

SDRAM_DFII_PI2_BADDRESS

Address: 0xf0006800 + 0x50 = 0xf0006850

DFI bank address bus

SDRAM_DFII_PI2_WRDATA1

Address: 0xf0006800 + 0x54 = 0xf0006854

Bits 32-63 of *SDRAM_DFII_PI2_WRDATA*. DFI write data bus

SDRAM_DFII_PI2_WRDATA0

Address: 0xf0006800 + 0x58 = 0xf0006858

Bits 0-31 of *SDRAM_DFII_PI2_WRDATA*.

SDRAM_DFII_PI2_RDDATA1

Address: 0xf0006800 + 0x5c = 0xf000685c

Bits 32-63 of *SDRAM_DFII_PI2_RDDATA*. DFI read data bus

SDRAM_DFII_PI2_RDDATA0

Address: 0xf0006800 + 0x60 = 0xf0006860

Bits 0-31 of *SDRAM_DFII_PI2_RDDATA*.

SDRAM_DFII_PI3_COMMAND

Address: 0xf0006800 + 0x64 = 0xf0006864

Control DFI signals on a single phase

Field	Name	Description
[0]	CS	DFI chip select bus
[1]	WE	DFI write enable bus
[2]	CAS	DFI column address strobe bus
[3]	RAS	DFI row address strobe bus
[4]	WREN	DFI write data enable bus
[5]	RDEN	DFI read data enable bus

SDRAM_DFII_PI3_COMMAND_ISSUE

Address: $0xf0006800 + 0x68 = 0xf0006868$

SDRAM_DFII_PI3_ADDRESS

Address: $0xf0006800 + 0x6c = 0xf000686c$

DFI address bus

SDRAM_DFII_PI3_BADDRESS

Address: $0xf0006800 + 0x70 = 0xf0006870$

DFI bank address bus

SDRAM_DFII_PI3_WRDATA1

Address: $0xf0006800 + 0x74 = 0xf0006874$

Bits 32-63 of *SDRAM_DFII_PI3_WRDATA*. DFI write data bus

SDRAM_DFII_PI3_WRDATA0

Address: $0xf0006800 + 0x78 = 0xf0006878$

Bits 0-31 of *SDRAM_DFII_PI3_WRDATA*.

SDRAM_DFII_PI3_RDDATA1

Address: 0xf0006800 + 0x7c = 0xf000687c

Bits 32-63 of *SDRAM_DFII_PI3_RDDATA*. DFI read data bus

SDRAM_DFII_PI3_RDDATA0

Address: 0xf0006800 + 0x80 = 0xf0006880

Bits 0-31 of *SDRAM_DFII_PI3_RDDATA*.

SDRAM_CONTROLLER_TRP

Address: 0xf0006800 + 0x84 = 0xf0006884

SDRAM_CONTROLLER_TRCD

Address: 0xf0006800 + 0x88 = 0xf0006888

SDRAM_CONTROLLER_TWR

Address: 0xf0006800 + 0x8c = 0xf000688c

SDRAM_CONTROLLER_TWTR

Address: 0xf0006800 + 0x90 = 0xf0006890

SDRAM_CONTROLLER_TREFI

Address: 0xf0006800 + 0x94 = 0xf0006894

SDRAM_CONTROLLER_TRFC

Address: 0xf0006800 + 0x98 = 0xf0006898

SDRAM_CONTROLLER_TFAW

Address: 0xf0006800 + 0x9c = 0xf000689c

SDRAM_CONTROLLER_TCCD

Address: 0xf0006800 + 0xa0 = 0xf00068a0

SDRAM_CONTROLLER_TCCD_WR

Address: 0xf0006800 + 0xa4 = 0xf00068a4

SDRAM_CONTROLLER_TRTP

Address: 0xf0006800 + 0xa8 = 0xf00068a8

SDRAM_CONTROLLER_TRRD

Address: 0xf0006800 + 0xac = 0xf00068ac

SDRAM_CONTROLLER_TRC

Address: 0xf0006800 + 0xb0 = 0xf00068b0

SDRAM_CONTROLLER_TRAS

Address: 0xf0006800 + 0xb4 = 0xf00068b4

SDRAM_CONTROLLER_TZQCS

Address: 0xf0006800 + 0xb8 = 0xf00068b8

SDRAM_CONTROLLER_LAST_ADDR_0

Address: 0xf0006800 + 0xbc = 0xf00068bc

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_0

Address: 0xf0006800 + 0xc0 = 0xf00068c0

SDRAM_CONTROLLER_LAST_ADDR_1

Address: 0xf0006800 + 0xc4 = 0xf00068c4

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_1

Address: 0xf0006800 + 0xc8 = 0xf00068c8

SDRAM_CONTROLLER_LAST_ADDR_2

Address: 0xf0006800 + 0xcc = 0xf00068cc

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_2

Address: 0xf0006800 + 0xd0 = 0xf00068d0

SDRAM_CONTROLLER_LAST_ADDR_3

Address: 0xf0006800 + 0xd4 = 0xf00068d4

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_3

Address: 0xf0006800 + 0xd8 = 0xf00068d8

SDRAM_CONTROLLER_LAST_ADDR_4

Address: 0xf0006800 + 0xdc = 0xf00068dc

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_4

Address: 0xf0006800 + 0xe0 = 0xf00068e0

SDRAM_CONTROLLER_LAST_ADDR_5

Address: 0xf0006800 + 0xe4 = 0xf00068e4

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_5

Address: 0xf0006800 + 0xe8 = 0xf00068e8

SDRAM_CONTROLLER_LAST_ADDR_6

Address: 0xf0006800 + 0xec = 0xf00068ec

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_6

Address: 0xf0006800 + 0xf0 = 0xf00068f0

SDRAM_CONTROLLER_LAST_ADDR_7

Address: 0xf0006800 + 0xf4 = 0xf00068f4

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_7

Address: 0xf0006800 + 0xf8 = 0xf00068f8

SDRAM_CONTROLLER_LAST_ADDR_8

Address: 0xf0006800 + 0xfc = 0xf00068fc

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_8

Address: 0xf0006800 + 0x100 = 0xf0006900

SDRAM_CONTROLLER_LAST_ADDR_9

Address: 0xf0006800 + 0x104 = 0xf0006904

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_9

Address: 0xf0006800 + 0x108 = 0xf0006908

SDRAM_CONTROLLER_LAST_ADDR_10

Address: 0xf0006800 + 0x10c = 0xf000690c

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_10

Address: 0xf0006800 + 0x110 = 0xf0006910

SDRAM_CONTROLLER_LAST_ADDR_11

Address: 0xf0006800 + 0x114 = 0xf0006914

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_11

Address: 0xf0006800 + 0x118 = 0xf0006918

SDRAM_CONTROLLER_LAST_ADDR_12

Address: 0xf0006800 + 0x11c = 0xf000691c

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_12

Address: 0xf0006800 + 0x120 = 0xf0006920

SDRAM_CONTROLLER_LAST_ADDR_13

Address: 0xf0006800 + 0x124 = 0xf0006924

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_13

Address: 0xf0006800 + 0x128 = 0xf0006928

SDRAM_CONTROLLER_LAST_ADDR_14

Address: 0xf0006800 + 0x12c = 0xf000692c

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_14

Address: 0xf0006800 + 0x130 = 0xf0006930

SDRAM_CONTROLLER_LAST_ADDR_15

Address: 0xf0006800 + 0x134 = 0xf0006934

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_15

Address: 0xf0006800 + 0x138 = 0xf0006938

14.2.15 SDRAM_CHECKER

Register Listing for SDRAM_CHECKER

Register	Address
<i>SDRAM_CHECKER_RESET</i>	0xf0007000
<i>SDRAM_CHECKER_START</i>	0xf0007004
<i>SDRAM_CHECKER_DONE</i>	0xf0007008
<i>SDRAM_CHECKER_BASE1</i>	0xf000700c
<i>SDRAM_CHECKER_BASE0</i>	0xf0007010
<i>SDRAM_CHECKER_END1</i>	0xf0007014
<i>SDRAM_CHECKER_END0</i>	0xf0007018
<i>SDRAM_CHECKER_LENGTH1</i>	0xf000701c
<i>SDRAM_CHECKER_LENGTH0</i>	0xf0007020
<i>SDRAM_CHECKER_RANDOM</i>	0xf0007024
<i>SDRAM_CHECKER_TICKS</i>	0xf0007028
<i>SDRAM_CHECKER_ERRORS</i>	0xf000702c

SDRAM_CHECKER_RESET

Address: $0xf0007000 + 0x0 = 0xf0007000$

SDRAM_CHECKER_START

Address: $0xf0007000 + 0x4 = 0xf0007004$

SDRAM_CHECKER_DONE

Address: $0xf0007000 + 0x8 = 0xf0007008$

SDRAM_CHECKER_BASE1

Address: $0xf0007000 + 0xc = 0xf000700c$

Bits 32-32 of *SDRAM_CHECKER_BASE*.

SDRAM_CHECKER_BASE0

Address: $0xf0007000 + 0x10 = 0xf0007010$

Bits 0-31 of *SDRAM_CHECKER_BASE*.

SDRAM_CHECKER_END1

Address: $0xf0007000 + 0x14 = 0xf0007014$

Bits 32-32 of *SDRAM_CHECKER_END*.

SDRAM_CHECKER_END0

Address: $0xf0007000 + 0x18 = 0xf0007018$

Bits 0-31 of *SDRAM_CHECKER_END*.

SDRAM_CHECKER_LENGTH1

Address: $0xf0007000 + 0x1c = 0xf000701c$

Bits 32-32 of *SDRAM_CHECKER_LENGTH*.

SDRAM_CHECKER_LENGTH0

Address: $0xf0007000 + 0x20 = 0xf0007020$

Bits 0-31 of *SDRAM_CHECKER_LENGTH*.

SDRAM_CHECKER_RANDOM

Address: $0xf0007000 + 0x24 = 0xf0007024$

Field	Name	Description

SDRAM_CHECKER_TICKS

Address: $0xf0007000 + 0x28 = 0xf0007028$

SDRAM_CHECKER_ERRORS

Address: $0xf0007000 + 0x2c = 0xf000702c$

14.2.16 SDRAM_GENERATOR

Register Listing for SDRAM_GENERATOR

Register	Address
<i>SDRAM_GENERATOR_RESET</i>	<i>0xf0007800</i>
<i>SDRAM_GENERATOR_START</i>	<i>0xf0007804</i>
<i>SDRAM_GENERATOR_DONE</i>	<i>0xf0007808</i>
<i>SDRAM_GENERATOR_BASE1</i>	<i>0xf000780c</i>
<i>SDRAM_GENERATOR_BASE0</i>	<i>0xf0007810</i>
<i>SDRAM_GENERATOR_END1</i>	<i>0xf0007814</i>
<i>SDRAM_GENERATOR_END0</i>	<i>0xf0007818</i>
<i>SDRAM_GENERATOR_LENGTH1</i>	<i>0xf000781c</i>
<i>SDRAM_GENERATOR_LENGTH0</i>	<i>0xf0007820</i>
<i>SDRAM_GENERATOR_RANDOM</i>	<i>0xf0007824</i>
<i>SDRAM_GENERATOR_TICKS</i>	<i>0xf0007828</i>

SDRAM_GENERATOR_RESET

Address: $0xf0007800 + 0x0 = 0xf0007800$

SDRAM_GENERATOR_START

Address: $0xf0007800 + 0x4 = 0xf0007804$

SDRAM_GENERATOR_DONE

Address: 0xf0007800 + 0x8 = 0xf0007808

SDRAM_GENERATOR_BASE1

Address: 0xf0007800 + 0xc = 0xf000780c

Bits 32-32 of *SDRAM_GENERATOR_BASE*.

SDRAM_GENERATOR_BASE0

Address: 0xf0007800 + 0x10 = 0xf0007810

Bits 0-31 of *SDRAM_GENERATOR_BASE*.

SDRAM_GENERATOR_END1

Address: 0xf0007800 + 0x14 = 0xf0007814

Bits 32-32 of *SDRAM_GENERATOR_END*.

SDRAM_GENERATOR_END0

Address: 0xf0007800 + 0x18 = 0xf0007818

Bits 0-31 of *SDRAM_GENERATOR_END*.

SDRAM_GENERATOR_LENGTH1

Address: 0xf0007800 + 0x1c = 0xf000781c

Bits 32-32 of *SDRAM_GENERATOR_LENGTH*.

SDRAM_GENERATOR_LENGTH0

Address: 0xf0007800 + 0x20 = 0xf0007820

Bits 0-31 of *SDRAM_GENERATOR_LENGTH*.

SDRAM_GENERATOR_RANDOM

Address: $0xf0007800 + 0x24 = 0xf0007824$

Field	Name	Description

SDRAM_GENERATOR_TICKS

Address: $0xf0007800 + 0x28 = 0xf0007828$

14.2.17 TIMERO

Timer

Provides a generic Timer core.

The Timer is implemented as a countdown timer that can be used in various modes:

- Polling : Returns current countdown value to software
- One-Shot: Loads itself and stops when value reaches 0
- Periodic: (Re-)Loads itself when value reaches 0

en register allows the user to enable/disable the Timer. When the Timer is enabled, it is automatically loaded with the value of *load* register.

When the Timer reaches 0, it is automatically reloaded with value of *reload* register.

The user can latch the current countdown value by writing to *update_value* register, it will update *value* register with current countdown value.

To use the Timer in One-Shot mode, the user needs to:

- Disable the timer
- Set the load register to the expected duration
- (Re-)Enable the Timer

To use the Timer in Periodic mode, the user needs to:

- Disable the Timer
- Set the load register to 0
- Set the reload register to the expected period
- Enable the Timer

For both modes, the CPU can be advertised by an IRQ that the duration/period has elapsed. (The CPU can also do software polling with `update_value` and `value` to know the elapsed duration)

Register Listing for TIMERO

Register	Address
<code>TIMERO_LOAD</code>	<code>0xf0008000</code>
<code>TIMERO_RELOAD</code>	<code>0xf0008004</code>
<code>TIMERO_EN</code>	<code>0xf0008008</code>
<code>TIMERO_UPDATE_VALUE</code>	<code>0xf000800c</code>
<code>TIMERO_VALUE</code>	<code>0xf0008010</code>
<code>TIMERO_EV_STATUS</code>	<code>0xf0008014</code>
<code>TIMERO_EV_PENDING</code>	<code>0xf0008018</code>
<code>TIMERO_EV_ENABLE</code>	<code>0xf000801c</code>

`TIMERO_LOAD`

Address: $0xf0008000 + 0x0 = 0xf0008000$

Load value when Timer is (re-)enabled. In One-Shot mode, the value written to this register specifies the Timer's duration in clock cycles.

`TIMERO_RELOAD`

Address: $0xf0008000 + 0x4 = 0xf0008004$

Reload value when Timer reaches 0. In Periodic mode, the value written to this register specify the Timer's period in clock cycles.

`TIMERO_EN`

Address: $0xf0008000 + 0x8 = 0xf0008008$

Enable flag of the Timer. Set this flag to 1 to enable/start the Timer. Set to 0 to disable the Timer.

`TIMERO_UPDATE_VALUE`

Address: $0xf0008000 + 0xc = 0xf000800c$

Update trigger for the current countdown value. A write to this register latches the current countdown value to `value` register.

TIMER0_VALUE

Address: $0xf0008000 + 0x10 = 0xf0008010$

Latched countdown value. This value is updated by writing to update_value.

TIMER0_EV_STATUS

Address: $0xf0008000 + 0x14 = 0xf0008014$

This register contains the current raw level of the zero event trigger. Writes to this register have no effect.

Field	Name	Description
[0]	ZERO	Level of the zero event

TIMER0_EV_PENDING

Address: $0xf0008000 + 0x18 = 0xf0008018$

When a zero event occurs, the corresponding bit will be set in this register. To clear the Event, set the corresponding bit in this register.

Field	Name	Description
[0]	ZERO	1 if a zero event occurred. This Event is triggered on a falling edge.

TIMER0_EV_ENABLE

Address: $0xf0008000 + 0x1c = 0xf000801c$

This register enables the corresponding zero events. Write a 0 to this register to disable individual events.

Field	Name	Description
[0]	ZERO	Write a 1 to enable the zero Event

14.2.18 UART

Register Listing for UART

Register	Address
<i>UART_RXTX</i>	0xf0008800
<i>UART_TXFULL</i>	0xf0008804
<i>UART_RXEMPTY</i>	0xf0008808
<i>UART_EV_STATUS</i>	0xf000880c
<i>UART_EV_PENDING</i>	0xf0008810
<i>UART_EV_ENABLE</i>	0xf0008814
<i>UART_TXEMPTY</i>	0xf0008818
<i>UART_RXFULL</i>	0xf000881c
<i>UART_XOVER_RXTX</i>	0xf0008820
<i>UART_XOVER_TXFULL</i>	0xf0008824
<i>UART_XOVER_RXEMPTY</i>	0xf0008828
<i>UART_XOVER_EV_STATUS</i>	0xf000882c
<i>UART_XOVER_EV_PENDING</i>	0xf0008830
<i>UART_XOVER_EV_ENABLE</i>	0xf0008834
<i>UART_XOVER_TXEMPTY</i>	0xf0008838
<i>UART_XOVER_RXFULL</i>	0xf000883c

UART_RXTX

Address: $0xf0008800 + 0x0 = 0xf0008800$

UART_TXFULL

Address: $0xf0008800 + 0x4 = 0xf0008804$

TX FIFO Full.

UART_RXEMPTY

Address: $0xf0008800 + 0x8 = 0xf0008808$

RX FIFO Empty.

UART_EV_STATUS

Address: $0xf0008800 + 0xc = 0xf000880c$

This register contains the current raw level of the rx event trigger. Writes to this register have no effect.

Field	Name	Description
[0]	TX	Level of the tx event
[1]	RX	Level of the rx event

UART_EV_PENDING

Address: $0xf0008800 + 0x10 = 0xf0008810$

When a rx event occurs, the corresponding bit will be set in this register. To clear the Event, set the corresponding bit in this register.

Field	Name	Description
[0]	TX	1 if a tx event occurred. This Event is triggered on a falling edge.
[1]	RX	1 if a rx event occurred. This Event is triggered on a falling edge.

UART_EV_ENABLE

Address: $0xf0008800 + 0x14 = 0xf0008814$

This register enables the corresponding rx events. Write a 0 to this register to disable individual events.

Field	Name	Description
[0]	TX	Write a 1 to enable the tx Event
[1]	RX	Write a 1 to enable the rx Event

UART_TXEMPTY

Address: $0xf0008800 + 0x18 = 0xf0008818$

TX FIFO Empty.

UART_RXFULL

Address: $0xf0008800 + 0x1c = 0xf000881c$

RX FIFO Full.

UART_XOVER_RXTX

Address: $0xf0008800 + 0x20 = 0xf0008820$

UART_XOVER_TXFULL

Address: $0xf0008800 + 0x24 = 0xf0008824$

TX FIFO Full.

UART_XOVER_RXEMPTY

Address: $0xf0008800 + 0x28 = 0xf0008828$

RX FIFO Empty.

UART_XOVER_EV_STATUS

Address: $0xf0008800 + 0x2c = 0xf000882c$

This register contains the current raw level of the rx event trigger. Writes to this register have no effect.

Field	Name	Description
[0]	TX	Level of the tx event
[1]	RX	Level of the rx event

UART_XOVER_EV_PENDING

Address: $0xf0008800 + 0x30 = 0xf0008830$

When a rx event occurs, the corresponding bit will be set in this register. To clear the Event, set the corresponding bit in this register.

Field	Name	Description
[0]	TX	1 if a tx event occurred. This Event is triggered on a falling edge.
[1]	RX	1 if a rx event occurred. This Event is triggered on a falling edge.

UART_XOVER_EV_ENABLE

Address: $0xf0008800 + 0x34 = 0xf0008834$

This register enables the corresponding rx events. Write a 0 to this register to disable individual events.

Field	Name	Description
[0]	TX	Write a 1 to enable the tx Event
[1]	RX	Write a 1 to enable the rx Event

UART_XOVER_TXEMPTY

Address: $0xf0008800 + 0x38 = 0xf0008838$

TX FIFO Empty.

UART_XOVER_RXFULL

Address: $0xf0008800 + 0x3c = 0xf000883c$

RX FIFO Full.

CHAPTER
FIFTEEN

DOCUMENTATION FOR ROW HAMMER TESTER LPDDR4 TEST BOARD

15.1 Modules

15.1.1 Interrupt Controller

This device has an EventManager-based interrupt system. Individual modules generate *events* which are wired into a central interrupt controller.

When an interrupt occurs, you should look the interrupt number up in the CPU- specific interrupt table and then call the relevant module.

Assigned Interrupts

The following interrupts are assigned on this system:

Interrupt	Module
1	<i>TIMERO</i>
0	<i>UART</i>

15.2 Register Groups

15.2.1 LEDS

Register Listing for LEDS

Register	Address
<i>LEDS_OUT</i>	<i>0xf0000000</i>

LEDS_OUT

Address: $0xf0000000 + 0x0 = 0xf0000000$

Led Output(s) Control.

15.2.2 DDRPHY

Register Listing for DDRPHY

Register	Address
<i>DDRPHY_RST</i>	<i>0xf0000800</i>
<i>DDRPHY_WLEVEL_EN</i>	<i>0xf0000804</i>
<i>DDRPHY_WLEVEL_STROBE</i>	<i>0xf0000808</i>
<i>DDRPHY_DL_Y_SEL</i>	<i>0xf000080c</i>
<i>DDRPHY_RDLY_DQ_BITSLIP_RST</i>	<i>0xf0000810</i>
<i>DDRPHY_RDLY_DQ_BITSLIP</i>	<i>0xf0000814</i>
<i>DDRPHY_WDLY_DQ_BITSLIP_RST</i>	<i>0xf0000818</i>
<i>DDRPHY_WDLY_DQ_BITSLIP</i>	<i>0xf000081c</i>
<i>DDRPHY_RDPHASE</i>	<i>0xf0000820</i>
<i>DDRPHY_WRPHASE</i>	<i>0xf0000824</i>
<i>DDRPHY_HALF_SYS8X_TAPS</i>	<i>0xf0000828</i>
<i>DDRPHY_RDLY_DQ_RST</i>	<i>0xf000082c</i>
<i>DDRPHY_RDLY_DQ_INC</i>	<i>0xf0000830</i>
<i>DDRPHY_RDLY_DQS_RST</i>	<i>0xf0000834</i>
<i>DDRPHY_RDLY_DQS_INC</i>	<i>0xf0000838</i>
<i>DDRPHY_CDLY_RST</i>	<i>0xf000083c</i>
<i>DDRPHY_CDLY_INC</i>	<i>0xf0000840</i>
<i>DDRPHY_WDLY_DQ_RST</i>	<i>0xf0000844</i>
<i>DDRPHY_WDLY_DQ_INC</i>	<i>0xf0000848</i>
<i>DDRPHY_WDLY_DQS_RST</i>	<i>0xf000084c</i>
<i>DDRPHY_WDLY_DQS_INC</i>	<i>0xf0000850</i>

DDRPHY_RST

Address: $0xf0000800 + 0x0 = 0xf0000800$

DDRPHY_WLEVEL_EN

Address: $0xf0000800 + 0x4 = 0xf0000804$

DDRPHY_WLEVEL_STROBE

Address: $0xf0000800 + 0x8 = 0xf0000808$

DDRPHY_DLY_SEL

Address: $0xf0000800 + 0xc = 0xf000080c$

DDRPHY_RDLY_DQ_BITSLIP_RST

Address: $0xf0000800 + 0x10 = 0xf0000810$

DDRPHY_RDLY_DQ_BITSLIP

Address: $0xf0000800 + 0x14 = 0xf0000814$

DDRPHY_WDLY_DQ_BITSLIP_RST

Address: $0xf0000800 + 0x18 = 0xf0000818$

DDRPHY_WDLY_DQ_BITSLIP

Address: $0xf0000800 + 0x1c = 0xf000081c$

DDRPHY_RDPHASE

Address: $0xf0000800 + 0x20 = 0xf0000820$

DDRPHY_WRPHASE

Address: $0xf0000800 + 0x24 = 0xf0000824$

DDRPHY_HALF_SYS8X_TAPS

Address: $0xf0000800 + 0x28 = 0xf0000828$

DDRPHY_RDLY_DQ_RST

Address: $0xf0000800 + 0x2c = 0xf000082c$

DDRPHY_RDLY_DQ_INC

Address: $0xf0000800 + 0x30 = 0xf0000830$

DDRPHY_RDLY_DQS_RST

Address: $0xf0000800 + 0x34 = 0xf0000834$

DDRPHY_RDLY_DQS_INC

Address: $0xf0000800 + 0x38 = 0xf0000838$

DDRPHY_CDLY_RST

Address: $0xf0000800 + 0x3c = 0xf000083c$

DDRPHY_CDLY_INC

Address: $0xf0000800 + 0x40 = 0xf0000840$

DDRPHY_WDLY_DQ_RST

Address: $0xf0000800 + 0x44 = 0xf0000844$

DDRPHY_WDLY_DQ_INC

Address: $0xf0000800 + 0x48 = 0xf0000848$

DDRPHY_WDLY_DQS_RST

Address: $0xf0000800 + 0x4c = 0xf000084c$

DDRPHY_WDLY_DQS_INC

Address: $0xf0000800 + 0x50 = 0xf0000850$

15.2.3 CONTROLLER_SETTINGS

Allows to change LiteDRAMController behaviour at runtime

Register Listing for CONTROLLER_SETTINGS

Register	Address
<i>CONTROLLER_SETTINGS_REFRESH</i>	<i>0xf0001000</i>

CONTROLLER_SETTINGS_REFRESH

Address: $0xf0001000 + 0x0 = 0xf0001000$

Enable/disable Refresh commands sending

15.2.4 DDRCTRL

Register Listing for DDRCTRL

Register	Address
<i>DDRCTRL_INIT_DONE</i>	<i>0xf0001800</i>
<i>DDRCTRL_INIT_ERROR</i>	<i>0xf0001804</i>

DDRCTRL_INIT_DONE

Address: $0xf0001800 + 0x0 = 0xf0001800$

DDRCTRL_INIT_ERROR

Address: $0xf0001800 + 0x4 = 0xf0001804$

15.2.5 ROWHAMMER

Row Hammer DMA attacker

This module allows to perform a Row Hammer attack by configuring it with two addresses that map to different rows of a single bank. When enabled, it will perform alternating DMA reads from the given locations, which will result in the DRAM controller having to repeatedly open/close rows at each read access.

Register Listing for ROWHAMMER

Register	Address
<code>ROWHAMMER_ENABLED</code>	<code>0xf0002000</code>
<code>ROWHAMMER_ADDRESS1</code>	<code>0xf0002004</code>
<code>ROWHAMMER_ADDRESS2</code>	<code>0xf0002008</code>
<code>ROWHAMMER_COUNT</code>	<code>0xf000200c</code>

`ROWHAMMER_ENABLED`

Address: $0xf0002000 + 0x0 = 0xf0002000$

Used to start/stop the operation of the module

`ROWHAMMER_ADDRESS1`

Address: $0xf0002000 + 0x4 = 0xf0002004$

First attacked address

`ROWHAMMER_ADDRESS2`

Address: $0xf0002000 + 0x8 = 0xf0002008$

Second attacked address

`ROWHAMMER_COUNT`

Address: $0xf0002000 + 0xc = 0xf000200c$

This is the number of DMA accesses performed. When the module is enabled, the value can be freely read. When the module is disabled, the register is clear-on-write and has to be read before the next attack.

15.2.6 WRITER

DMA DRAM writer.

Allows to fill DRAM with a predefined pattern using DMA.

Pattern

Provides access to RAM to store access pattern: *mem_addr* and *mem_data*. The pattern address space can be limited using the *data_mask*.

For example, having *mem_addr* filled with [0x04, 0x02, 0x03, ...] and *mem_data* filled with [0xff, 0xaa, 0x55, ...] and setting *data_mask* = 0b01, the pattern [(address, data), ...] written will be: [(0x04, 0xff), (0x02, 0xaa), (0x04, 0xff), ...] (wraps due to masking).

DRAM memory range that is being accessed can be configured using *mem_mask*.

To use this module, make sure that *ready* is 1, then write the desired number of transfers to *count*. Writing to the *start* CSR will initialize the operation. When the operation is ongoing *ready* will be 0.

Register Listing for WRITER

Register	Address
WRITER_START	0xf0002800
WRITER_READY	0xf0002804
WRITER_MODULO	0xf0002808
WRITER_COUNT	0xf000280c
WRITER_DONE	0xf0002810
WRITER_MEM_MASK	0xf0002814
WRITER_DATA_MASK	0xf0002818
WRITER_DATA_DIV	0xf000281c
WRITER_INVERTER_DIVISOR_MASK	0xf0002820
WRITER_INVERTER_SELECTION_MASK	0xf0002824
WRITER_LAST_ADDRESS	0xf0002828

WRITER_START

Address: 0xf0002800 + 0x0 = 0xf0002800

Write to the register starts the transfer (if ready=1)

WRITER_READY

Address: 0xf0002800 + 0x4 = 0xf0002804

Indicates that the transfer is not ongoing

WRITER_MODULO

Address: $0xf0002800 + 0x8 = 0xf0002808$

When set use modulo to calculate DMA transfers address rather than bit masking

WRITER_COUNT

Address: $0xf0002800 + 0xc = 0xf000280c$

Desired number of DMA transfers

WRITER_DONE

Address: $0xf0002800 + 0x10 = 0xf0002810$

Number of completed DMA transfers

WRITER_MEM_MASK

Address: $0xf0002800 + 0x14 = 0xf0002814$

DRAM address mask for DMA transfers

WRITER_DATA_MASK

Address: $0xf0002800 + 0x18 = 0xf0002818$

Pattern memory address mask

WRITER_DATA_DIV

Address: $0xf0002800 + 0x1c = 0xf000281c$

Pattern memory address divisor-1

WRITER_INVERTER_DIVISOR_MASK

Address: $0xf0002800 + 0x20 = 0xf0002820$

Divisor mask for selecting rows for which pattern data gets inverted

WRITER_INVERTER_SELECTION_MASK

Address: $0xf0002800 + 0x24 = 0xf0002824$

Selection mask for selecting rows for which pattern data gets inverted

WRITER_LAST_ADDRESS

Address: $0xf0002800 + 0x28 = 0xf0002828$

Number of completed DMA transfers

15.2.7 READER

DMA DRAM reader.

Allows to check DRAM contents against a predefined pattern using DMA.

Pattern

Provides access to RAM to store access pattern: *mem_addr* and *mem_data*. The pattern address space can be limited using the *data_mask*.

For example, having *mem_addr* filled with [0x04, 0x02, 0x03, ...] and *mem_data* filled with [0xff, 0xaa, 0x55, ...] and setting *data_mask* = 0b01, the pattern [(address, data), ...] written will be: [(0x04, 0xff), (0x02, 0xaa), (0x04, 0xff), ...] (wraps due to masking).

DRAM memory range that is being accessed can be configured using *mem_mask*.

To use this module, make sure that *ready* is 1, then write the desired number of transfers to *count*. Writing to the *start* CSR will initialize the operation. When the operation is ongoing *ready* will be 0.

Reading errors

This module allows to check the locations of errors in the memory. It scans the configured memory area and compares the values read to the predefined pattern. If *skip_fifo* is 0, this module will stop after each error encountered, so that it can be examined. Wait until the *error_ready* CSR is 1. Then use the CSRs *error_offset*, *error_data* and *error_expected* to examine the errors in the current transfer. To continue reading, write 1 to *error_continue* CSR. Setting *skip_fifo* to 1 will disable this behaviour entirely.

The final number of errors can be read from *error_count*. NOTE: This value represents the number of erroneous DMA transfers.

The current progress can be read from the *done* CSR.

Register Listing for READER

Register	Address
READER_START	0xf0003000
READER_READY	0xf0003004
READER_MODULO	0xf0003008
READER_COUNT	0xf000300c
READER_DONE	0xf0003010
READER_MEM_MASK	0xf0003014
READER_DATA_MASK	0xf0003018
READER_DATA_DIV	0xf000301c
READER_INVERTER_DIVISOR_MASK	0xf0003020
READER_INVERTER_SELECTION_MASK	0xf0003024
READER_ERROR_COUNT	0xf0003028
READER_SKIP_FIFO	0xf000302c
READER_ERROR_OFFSET	0xf0003030
READER_ERROR_DATA7	0xf0003034
READER_ERROR_DATA6	0xf0003038
READER_ERROR_DATA5	0xf000303c
READER_ERROR_DATA4	0xf0003040
READER_ERROR_DATA3	0xf0003044
READER_ERROR_DATA2	0xf0003048
READER_ERROR_DATA1	0xf000304c
READER_ERROR_DATA0	0xf0003050
READER_ERROR_EXPECTED7	0xf0003054
READER_ERROR_EXPECTED6	0xf0003058
READER_ERROR_EXPECTED5	0xf000305c
READER_ERROR_EXPECTED4	0xf0003060
READER_ERROR_EXPECTED3	0xf0003064
READER_ERROR_EXPECTED2	0xf0003068
READER_ERROR_EXPECTED1	0xf000306c
READER_ERROR_EXPECTED0	0xf0003070
READER_ERROR_READY	0xf0003074
READER_ERROR_CONTINUE	0xf0003078

READER_START

Address: 0xf0003000 + 0x0 = 0xf0003000

Write to the register starts the transfer (if ready=1)

READER_READY

Address: $0xf0003000 + 0x4 = 0xf0003004$

Indicates that the transfer is not ongoing

READER_MODULO

Address: $0xf0003000 + 0x8 = 0xf0003008$

When set use modulo to calculate DMA transfers address rather than bit masking

READER_COUNT

Address: $0xf0003000 + 0xc = 0xf000300c$

Desired number of DMA transfers

READER_DONE

Address: $0xf0003000 + 0x10 = 0xf0003010$

Number of completed DMA transfers

READER_MEM_MASK

Address: $0xf0003000 + 0x14 = 0xf0003014$

DRAM address mask for DMA transfers

READER_DATA_MASK

Address: $0xf0003000 + 0x18 = 0xf0003018$

Pattern memory address mask

READER_DATA_DIV

Address: $0xf0003000 + 0x1c = 0xf000301c$

Pattern memory address divisor-1

READER_INVERTER_DIVISOR_MASK

Address: $0xf0003000 + 0x20 = 0xf0003020$

Divisor mask for selecting rows for which pattern data gets inverted

READER_INVERTER_SELECTION_MASK

Address: $0xf0003000 + 0x24 = 0xf0003024$

Selection mask for selecting rows for which pattern data gets inverted

READER_ERROR_COUNT

Address: $0xf0003000 + 0x28 = 0xf0003028$

Number of errors detected

READER_SKIP_FIFO

Address: $0xf0003000 + 0x2c = 0xf000302c$

Skip waiting for user to read the errors FIFO

READER_ERROR_OFFSET

Address: $0xf0003000 + 0x30 = 0xf0003030$

Current offset of the error

READER_ERROR_DATA7

Address: $0xf0003000 + 0x34 = 0xf0003034$

Bits 224-255 of *READER_ERROR_DATA*. Erroneous value read from DRAM memory

READER_ERROR_DATA6

Address: $0xf0003000 + 0x38 = 0xf0003038$

Bits 192-223 of *READER_ERROR_DATA*.

READER_ERROR_DATA5

Address: $0xf0003000 + 0x3c = 0xf000303c$

Bits 160-191 of *READER_ERROR_DATA*.

READER_ERROR_DATA4

Address: $0xf0003000 + 0x40 = 0xf0003040$

Bits 128-159 of *READER_ERROR_DATA*.

READER_ERROR_DATA3

Address: $0xf0003000 + 0x44 = 0xf0003044$

Bits 96-127 of *READER_ERROR_DATA*.

READER_ERROR_DATA2

Address: $0xf0003000 + 0x48 = 0xf0003048$

Bits 64-95 of *READER_ERROR_DATA*.

READER_ERROR_DATA1

Address: $0xf0003000 + 0x4c = 0xf000304c$

Bits 32-63 of *READER_ERROR_DATA*.

READER_ERROR_DATA0

Address: $0xf0003000 + 0x50 = 0xf0003050$

Bits 0-31 of *READER_ERROR_DATA*.

READER_ERROR_EXPECTED7

Address: $0xf0003000 + 0x54 = 0xf0003054$

Bits 224-255 of *READER_ERROR_EXPECTED*. Value expected to be read from DRAM memory

READER_ERROR_EXPECTED6

Address: $0xf0003000 + 0x58 = 0xf0003058$

Bits 192-223 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED5

Address: $0xf0003000 + 0x5c = 0xf000305c$

Bits 160-191 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED4

Address: $0xf0003000 + 0x60 = 0xf0003060$

Bits 128-159 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED3

Address: $0xf0003000 + 0x64 = 0xf0003064$

Bits 96-127 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED2

Address: $0xf0003000 + 0x68 = 0xf0003068$

Bits 64-95 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED1

Address: $0xf0003000 + 0x6c = 0xf000306c$

Bits 32-63 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED0

Address: $0xf0003000 + 0x70 = 0xf0003070$

Bits 0-31 of *READER_ERROR_EXPECTED*.

READER_ERROR_READY

Address: $0xf0003000 + 0x74 = 0xf0003074$

Error detected and ready to read

READER_ERROR_CONTINUE

Address: $0xf0003000 + 0x78 = 0xf0003078$

Continue reading until the next error

15.2.8 DFI_SWITCH

Register Listing for DFI_SWITCH

Register	Address
<i>DFI_SWITCH_REFRESH_COUNT</i>	<i>0xf0003800</i>
<i>DFI_SWITCH_AT_REFRESH</i>	<i>0xf0003804</i>
<i>DFI_SWITCH_REFRESH_UPDATE</i>	<i>0xf0003808</i>

DFI_SWITCH_REFRESH_COUNT

Address: $0xf0003800 + 0x0 = 0xf0003800$

Count of all refresh commands issued (both by Memory Controller and Payload Executor). Value is latched from internal counter on mode transition: MC -> PE or by writing to the *refresh_update* CSR.

DFI_SWITCH_AT_REFRESH

Address: $0xf0003800 + 0x4 = 0xf0003804$

If set to a value different than 0 the mode transition MC -> PE will be performed only when the value of this register matches the current refresh commands count.

DFI_SWITCH_REFRESH_UPDATE

Address: $0xf0003800 + 0x8 = 0xf0003808$

Force an update of the *refresh_count* CSR.

15.2.9 PAYLOAD_EXECUTOR

Executes the DRAM payload from memory

Instruction decoder

All instructions are 32-bit. The format of most instructions is the same, except for the LOOP instruction, which has a constant TIMESLICE of 1.

NOOP with a TIMESLICE of 0 is a special case which is interpreted as STOP instruction. When this instruction is encountered execution gets finished immediately.

NOTE: TIMESLICE is the number of cycles the instruction will take. This means that instructions other than NOOP that use TIMESLICE=0 are illegal (although will silently be executed as having TIMESLICE=1).

NOTE2: LOOP instruction will *jump* COUNT times, meaning that the “code” inside the loop will effectively be executed COUNT+1 times.

Op codes:

Op	Value
NOOP	0b000
LOOP	0b111
ACT	0b100
PRE	0b101
REF	0b110
ZQC	0b001
READ	0b010

Instruction format:

LSB	MSB
dfi: OP_CODE TIMESLICE ADDRESS	
noop: OP_CODE TIMESLICE_NOOP	
loop: OP_CODE COUNT JUMP	
stop: <NOOP> 0	

Where ADDRESS depends on the DFI command and is one of:

LSB	MSB
RANK BANK COLUMN	
RANK BANK ROW	

Register Listing for PAYLOAD_EXECUTOR

Register	Address
PAYLOAD_EXECUTOR_START	0xf0004000
PAYLOAD_EXECUTOR_STATUS	0xf0004004
PAYLOAD_EXECUTOR_READ_COUNT	0xf0004008

PAYLOAD_EXECUTOR_START

Address: $0xf0004000 + 0x0 = 0xf0004000$

Writing to this register initializes payload execution

PAYLOAD_EXECUTOR_STATUS

Address: $0xf0004000 + 0x4 = 0xf0004004$

Payload executor status register

Field	Name	Description
[0]	READY	Indicates that the executor is not running
[1]	OVERFLOW	Indicates the scratchpad memory address counter has overflowed due to the number of READ commands sent during execution

PAYLOAD_EXECUTOR_READ_COUNT

Address: $0xf0004000 + 0x8 = 0xf0004008$

Number of data from READ commands that is stored in the scratchpad memory

15.2.10 CTRL

Register Listing for CTRL

Register	Address
CTRL_RESET	0xf0004800
CTRL_SCRATCH	0xf0004804
CTRL_BUS_ERRORS	0xf0004808

CTRL__RESET

Address: $0xf0004800 + 0x0 = 0xf0004800$

Field	Name	Description
[0]	SOC_RST	Write 1 to this register to reset the full SoC (Pulse Reset)
[1]	CPU_RST	Write 1 to this register to reset the CPU(s) of the SoC (Hold Reset)

CTRL_SCRATCH

Address: $0xf0004800 + 0x4 = 0xf0004804$

Use this register as a scratch space to verify that software read/write accesses to the Wishbone/CSR bus are working correctly. The initial reset value of 0x1234578 can be used to verify endianness.

CTRL_BUS_ERRORS

Address: $0xf0004800 + 0x8 = 0xf0004808$

Total number of Wishbone bus errors (timeouts) since start.

15.2.11 ETPHY

Register Listing for ETPHY

Register	Address
ETPHY_CRG_RESET	<i>0xf0005000</i>
ETPHY_MDIO_W	<i>0xf0005004</i>
ETPHY_MDIO_R	<i>0xf0005008</i>

ETPHY_CRG_RESET

Address: $0xf0005000 + 0x0 = 0xf0005000$

ETHPHY_MDIO_W

Address: $0xf0005000 + 0x4 = 0xf0005004$

Field	Name	Description

ETHPHY_MDIO_R

Address: $0xf0005000 + 0x8 = 0xf0005008$

Field	Name	Description

15.2.12 IDENTIFIER_MEM

Register Listing for IDENTIFIER_MEM

Register	Address
<i>IDENTIFIER_MEM</i>	<i>0xf0005800</i>

IDENTIFIER_MEM

Address: $0xf0005800 + 0x0 = 0xf0005800$

8 x 108-bit memory

15.2.13 SDRAM

Register Listing for SDRAM

Register	Address
<i>SDRAM_DFII_CONTROL</i>	<i>0xf0006000</i>
<i>SDRAM_DFII_PIO_COMMAND</i>	<i>0xf0006004</i>
<i>SDRAM_DFII_PIO_COMMAND_ISSUE</i>	<i>0xf0006008</i>
<i>SDRAM_DFII_PIO_ADDRESS</i>	<i>0xf000600c</i>

continues on next page

Table 15.2 – continued from previous page

Register	Address
<i>SDRAM_DFII_PIO_BADDRESS</i>	0xf0006010
<i>SDRAM_DFII_PIO_WRDATA</i>	0xf0006014
<i>SDRAM_DFII_PIO_RDDATA</i>	0xf0006018
<i>SDRAM_DFII_PI1_COMMAND</i>	0xf000601c
<i>SDRAM_DFII_PI1_COMMAND_ISSUE</i>	0xf0006020
<i>SDRAM_DFII_PI1_ADDRESS</i>	0xf0006024
<i>SDRAM_DFII_PI1_BADDRESS</i>	0xf0006028
<i>SDRAM_DFII_PI1_WRDATA</i>	0xf000602c
<i>SDRAM_DFII_PI1_RDDATA</i>	0xf0006030
<i>SDRAM_DFII_PI2_COMMAND</i>	0xf0006034
<i>SDRAM_DFII_PI2_COMMAND_ISSUE</i>	0xf0006038
<i>SDRAM_DFII_PI2_ADDRESS</i>	0xf000603c
<i>SDRAM_DFII_PI2_BADDRESS</i>	0xf0006040
<i>SDRAM_DFII_PI2_WRDATA</i>	0xf0006044
<i>SDRAM_DFII_PI2_RDDATA</i>	0xf0006048
<i>SDRAM_DFII_PI3_COMMAND</i>	0xf000604c
<i>SDRAM_DFII_PI3_COMMAND_ISSUE</i>	0xf0006050
<i>SDRAM_DFII_PI3_ADDRESS</i>	0xf0006054
<i>SDRAM_DFII_PI3_BADDRESS</i>	0xf0006058
<i>SDRAM_DFII_PI3_WRDATA</i>	0xf000605c
<i>SDRAM_DFII_PI3_RDDATA</i>	0xf0006060
<i>SDRAM_DFII_PI4_COMMAND</i>	0xf0006064
<i>SDRAM_DFII_PI4_COMMAND_ISSUE</i>	0xf0006068
<i>SDRAM_DFII_PI4_ADDRESS</i>	0xf000606c
<i>SDRAM_DFII_PI4_BADDRESS</i>	0xf0006070
<i>SDRAM_DFII_PI4_WRDATA</i>	0xf0006074
<i>SDRAM_DFII_PI4_RDDATA</i>	0xf0006078
<i>SDRAM_DFII_PI5_COMMAND</i>	0xf000607c
<i>SDRAM_DFII_PI5_COMMAND_ISSUE</i>	0xf0006080
<i>SDRAM_DFII_PI5_ADDRESS</i>	0xf0006084
<i>SDRAM_DFII_PI5_BADDRESS</i>	0xf0006088
<i>SDRAM_DFII_PI5_WRDATA</i>	0xf000608c
<i>SDRAM_DFII_PI5_RDDATA</i>	0xf0006090
<i>SDRAM_DFII_PI6_COMMAND</i>	0xf0006094
<i>SDRAM_DFII_PI6_COMMAND_ISSUE</i>	0xf0006098
<i>SDRAM_DFII_PI6_ADDRESS</i>	0xf000609c
<i>SDRAM_DFII_PI6_BADDRESS</i>	0xf00060a0
<i>SDRAM_DFII_PI6_WRDATA</i>	0xf00060a4
<i>SDRAM_DFII_PI6_RDDATA</i>	0xf00060a8
<i>SDRAM_DFII_PI7_COMMAND</i>	0xf00060ac
<i>SDRAM_DFII_PI7_COMMAND_ISSUE</i>	0xf00060b0
<i>SDRAM_DFII_PI7_ADDRESS</i>	0xf00060b4
<i>SDRAM_DFII_PI7_BADDRESS</i>	0xf00060b8
<i>SDRAM_DFII_PI7_WRDATA</i>	0xf00060bc
<i>SDRAM_DFII_PI7_RDDATA</i>	0xf00060c0
<i>SDRAM_CONTROLLER_TRP</i>	0xf00060c4
<i>SDRAM_CONTROLLER_TRCD</i>	0xf00060c8
<i>SDRAM_CONTROLLER_TWR</i>	0xf00060cc

continues on next page

Table 15.2 – continued from previous page

Register	Address
<i>SDRAM_CONTROLLER_TWTR</i>	0xf00060d0
<i>SDRAM_CONTROLLER_TREFI</i>	0xf00060d4
<i>SDRAM_CONTROLLER_TRFC</i>	0xf00060d8
<i>SDRAM_CONTROLLER_TFAW</i>	0xf00060dc
<i>SDRAM_CONTROLLER_TCCD</i>	0xf00060e0
<i>SDRAM_CONTROLLER_TCCD_WR</i>	0xf00060e4
<i>SDRAM_CONTROLLER_TRTP</i>	0xf00060e8
<i>SDRAM_CONTROLLER_TRRD</i>	0xf00060ec
<i>SDRAM_CONTROLLER_TRC</i>	0xf00060f0
<i>SDRAM_CONTROLLER_TRAS</i>	0xf00060f4
<i>SDRAM_CONTROLLER_LAST_ADDR_0</i>	0xf00060f8
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_0</i>	0xf00060fc
<i>SDRAM_CONTROLLER_LAST_ADDR_1</i>	0xf0006100
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_1</i>	0xf0006104
<i>SDRAM_CONTROLLER_LAST_ADDR_2</i>	0xf0006108
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_2</i>	0xf000610c
<i>SDRAM_CONTROLLER_LAST_ADDR_3</i>	0xf0006110
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_3</i>	0xf0006114
<i>SDRAM_CONTROLLER_LAST_ADDR_4</i>	0xf0006118
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_4</i>	0xf000611c
<i>SDRAM_CONTROLLER_LAST_ADDR_5</i>	0xf0006120
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_5</i>	0xf0006124
<i>SDRAM_CONTROLLER_LAST_ADDR_6</i>	0xf0006128
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_6</i>	0xf000612c
<i>SDRAM_CONTROLLER_LAST_ADDR_7</i>	0xf0006130
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_7</i>	0xf0006134

SDRAM_DFII_CONTROL

Address: 0xf0006000 + 0x0 = 0xf0006000

Control DFI signals common to all phases

Field	Name	Description						
[0]	SEL	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>0b0</td><td>Software control. (CPU)</td></tr> <tr> <td>0b1</td><td>Hardware control (default).</td></tr> </tbody> </table>	Value	Description	0b0	Software control. (CPU)	0b1	Hardware control (default).
Value	Description							
0b0	Software control. (CPU)							
0b1	Hardware control (default).							
[1]	CKE	DFI clock enable bus						
[2]	ODT	DFI on-die termination bus						
[3]	RESET_N	DFI clock reset bus						

SDRAM_DFIIPIO_COMMAND

Address: $0xf0006000 + 0x4 = 0xf0006004$

Control DFI signals on a single phase

Field	Name	Description
[0]	CS	DFI chip select bus
[1]	WE	DFI write enable bus
[2]	CAS	DFI column address strobe bus
[3]	RAS	DFI row address strobe bus
[4]	WREN	DFI write data enable bus
[5]	RDEN	DFI read data enable bus

SDRAM_DFIIPIO_COMMAND_ISSUE

Address: $0xf0006000 + 0x8 = 0xf0006008$

SDRAM_DFIIPIO_ADDRESS

Address: $0xf0006000 + 0xc = 0xf000600c$

DFI address bus

SDRAM_DFIIPIO_BADDRESS

Address: $0xf0006000 + 0x10 = 0xf0006010$

DFI bank address bus

SDRAM_DFIIPIO_WRDATA

Address: $0xf0006000 + 0x14 = 0xf0006014$

DFI write data bus

SDRAM_DFIIPIO_RDDATA

Address: $0xf0006000 + 0x18 = 0xf0006018$

DFI read data bus

SDRAM_DFII_PI1_COMMAND

Address: $0xf0006000 + 0x1c = 0xf000601c$

Control DFI signals on a single phase

Field	Name	Description
[0]	CS	DFI chip select bus
[1]	WE	DFI write enable bus
[2]	CAS	DFI column address strobe bus
[3]	RAS	DFI row address strobe bus
[4]	WREN	DFI write data enable bus
[5]	RDEN	DFI read data enable bus

SDRAM_DFII_PI1_COMMAND_ISSUE

Address: $0xf0006000 + 0x20 = 0xf0006020$

SDRAM_DFII_PI1_ADDRESS

Address: $0xf0006000 + 0x24 = 0xf0006024$

DFI address bus

SDRAM_DFII_PI1_BADDRESS

Address: $0xf0006000 + 0x28 = 0xf0006028$

DFI bank address bus

SDRAM_DFII_PI1_WRDATA

Address: $0xf0006000 + 0x2c = 0xf000602c$

DFI write data bus

SDRAM_DFII_PI1_RDDATA

Address: $0xf0006000 + 0x30 = 0xf0006030$

DFI read data bus

SDRAM_DFII_PI2_COMMAND

Address: $0xf0006000 + 0x34 = 0xf0006034$

Control DFI signals on a single phase

Field	Name	Description
[0]	CS	DFI chip select bus
[1]	WE	DFI write enable bus
[2]	CAS	DFI column address strobe bus
[3]	RAS	DFI row address strobe bus
[4]	WREN	DFI write data enable bus
[5]	RDEN	DFI read data enable bus

SDRAM_DFII_PI2_COMMAND_ISSUE

Address: $0xf0006000 + 0x38 = 0xf0006038$

SDRAM_DFII_PI2_ADDRESS

Address: $0xf0006000 + 0x3c = 0xf000603c$

DFI address bus

SDRAM_DFII_PI2_BADDRESS

Address: $0xf0006000 + 0x40 = 0xf0006040$

DFI bank address bus

SDRAM_DFII_PI2_WRDATA

Address: $0xf0006000 + 0x44 = 0xf0006044$

DFI write data bus

SDRAM_DFII_PI2_RDDATA

Address: $0xf0006000 + 0x48 = 0xf0006048$

DFI read data bus

SDRAM_DFII_PI3_COMMAND

Address: $0xf0006000 + 0x4c = 0xf000604c$

Control DFI signals on a single phase

Field	Name	Description
[0]	CS	DFI chip select bus
[1]	WE	DFI write enable bus
[2]	CAS	DFI column address strobe bus
[3]	RAS	DFI row address strobe bus
[4]	WREN	DFI write data enable bus
[5]	RDEN	DFI read data enable bus

SDRAM_DFII_PI3_COMMAND_ISSUE

Address: $0xf0006000 + 0x50 = 0xf0006050$

SDRAM_DFII_PI3_ADDRESS

Address: $0xf0006000 + 0x54 = 0xf0006054$

DFI address bus

SDRAM_DFII_PI3_BADDRESS

Address: $0xf0006000 + 0x58 = 0xf0006058$

DFI bank address bus

SDRAM_DFII_PI3_WRDATA

Address: $0xf0006000 + 0x5c = 0xf000605c$

DFI write data bus

SDRAM_DFII_PI3_RDDATA

Address: $0xf0006000 + 0x60 = 0xf0006060$

DFI read data bus

SDRAM_DFII_PI4_COMMAND

Address: $0xf0006000 + 0x64 = 0xf0006064$

Control DFI signals on a single phase

Field	Name	Description
[0]	CS	DFI chip select bus
[1]	WE	DFI write enable bus
[2]	CAS	DFI column address strobe bus
[3]	RAS	DFI row address strobe bus
[4]	WREN	DFI write data enable bus
[5]	RDEN	DFI read data enable bus

SDRAM_DFII_PI4_COMMAND_ISSUE

Address: $0xf0006000 + 0x68 = 0xf0006068$

SDRAM_DFII_PI4_ADDRESS

Address: $0xf0006000 + 0x6c = 0xf000606c$

DFI address bus

SDRAM_DFII_PI4_BADDRESS

Address: $0xf0006000 + 0x70 = 0xf0006070$

DFI bank address bus

SDRAM_DFII_PI4_WRDATA

Address: $0xf0006000 + 0x74 = 0xf0006074$

DFI write data bus

SDRAM_DFII_PI4_RDDATA

Address: $0xf0006000 + 0x78 = 0xf0006078$

DFI read data bus

SDRAM_DFII_PI5_COMMAND

Address: $0xf0006000 + 0x7c = 0xf000607c$

Control DFI signals on a single phase

Field	Name	Description
[0]	CS	DFI chip select bus
[1]	WE	DFI write enable bus
[2]	CAS	DFI column address strobe bus
[3]	RAS	DFI row address strobe bus
[4]	WREN	DFI write data enable bus
[5]	RDEN	DFI read data enable bus

SDRAM_DFII_PI5_COMMAND_ISSUE

Address: $0xf0006000 + 0x80 = 0xf0006080$

SDRAM_DFII_PI5_ADDRESS

Address: $0xf0006000 + 0x84 = 0xf0006084$

DFI address bus

SDRAM_DFII_PI5_BADDRESS

Address: $0xf0006000 + 0x88 = 0xf0006088$

DFI bank address bus

SDRAM_DFII_PI5_WRDATA

Address: $0xf0006000 + 0x8c = 0xf000608c$

DFI write data bus

SDRAM_DFII_PI5_RDDATA

Address: $0xf0006000 + 0x90 = 0xf0006090$

DFI read data bus

SDRAM_DFII_PI6_COMMAND

Address: $0xf0006000 + 0x94 = 0xf0006094$

Control DFI signals on a single phase

Field	Name	Description
[0]	CS	DFI chip select bus
[1]	WE	DFI write enable bus
[2]	CAS	DFI column address strobe bus
[3]	RAS	DFI row address strobe bus
[4]	WREN	DFI write data enable bus
[5]	RDEN	DFI read data enable bus

SDRAM_DFII_PI6_COMMAND_ISSUE

Address: $0xf0006000 + 0x98 = 0xf0006098$

SDRAM_DFII_PI6_ADDRESS

Address: $0xf0006000 + 0x9c = 0xf000609c$

DFI address bus

SDRAM_DFII_PI6_BADDRESS

Address: $0xf0006000 + 0xa0 = 0xf00060a0$

DFI bank address bus

SDRAM_DFII_PI6_WRDATA

Address: $0xf0006000 + 0xa4 = 0xf00060a4$

DFI write data bus

SDRAM_DFII_PI6_RDDATA

Address: $0xf0006000 + 0xa8 = 0xf00060a8$

DFI read data bus

SDRAM_DFII_PI7_COMMAND

Address: $0xf0006000 + 0xac = 0xf00060ac$

Control DFI signals on a single phase

Field	Name	Description
[0]	CS	DFI chip select bus
[1]	WE	DFI write enable bus
[2]	CAS	DFI column address strobe bus
[3]	RAS	DFI row address strobe bus
[4]	WREN	DFI write data enable bus
[5]	RDEN	DFI read data enable bus

SDRAM_DFII_PI7_COMMAND_ISSUE

Address: $0xf0006000 + 0xb0 = 0xf00060b0$

SDRAM_DFII_PI7_ADDRESS

Address: $0xf0006000 + 0xb4 = 0xf00060b4$

DFI address bus

SDRAM_DFII_PI7_BADDRESS

Address: $0xf0006000 + 0xb8 = 0xf00060b8$

DFI bank address bus

SDRAM_DFII_PI7_WRDATA

Address: $0xf0006000 + 0xbc = 0xf00060bc$

DFI write data bus

SDRAM_DFII_PI7_RDDATA

Address: $0xf0006000 + 0xc0 = 0xf00060c0$

DFI read data bus

SDRAM_CONTROLLER_TRP

Address: 0xf0006000 + 0xc4 = 0xf00060c4

SDRAM_CONTROLLER_TRCD

Address: 0xf0006000 + 0xc8 = 0xf00060c8

SDRAM_CONTROLLER_TWR

Address: 0xf0006000 + 0xcc = 0xf00060cc

SDRAM_CONTROLLER_TWTR

Address: 0xf0006000 + 0xd0 = 0xf00060d0

SDRAM_CONTROLLER_TREFI

Address: 0xf0006000 + 0xd4 = 0xf00060d4

SDRAM_CONTROLLER_TRFC

Address: 0xf0006000 + 0xd8 = 0xf00060d8

SDRAM_CONTROLLER_TFAW

Address: 0xf0006000 + 0xdc = 0xf00060dc

SDRAM_CONTROLLER_TCCD

Address: 0xf0006000 + 0xe0 = 0xf00060e0

SDRAM_CONTROLLER_TCCD_WR

Address: 0xf0006000 + 0xe4 = 0xf00060e4

SDRAM_CONTROLLER_TRTP

Address: 0xf0006000 + 0xe8 = 0xf00060e8

SDRAM_CONTROLLER_TRRD

Address: 0xf0006000 + 0xec = 0xf00060ec

SDRAM_CONTROLLER_TRC

Address: 0xf0006000 + 0xf0 = 0xf00060f0

SDRAM_CONTROLLER_TRAS

Address: 0xf0006000 + 0xf4 = 0xf00060f4

SDRAM_CONTROLLER_LAST_ADDR_0

Address: 0xf0006000 + 0xf8 = 0xf00060f8

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_0

Address: 0xf0006000 + 0xfc = 0xf00060fc

SDRAM_CONTROLLER_LAST_ADDR_1

Address: 0xf0006000 + 0x100 = 0xf0006100

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_1

Address: 0xf0006000 + 0x104 = 0xf0006104

SDRAM_CONTROLLER_LAST_ADDR_2

Address: 0xf0006000 + 0x108 = 0xf0006108

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_2

Address: 0xf0006000 + 0x10c = 0xf000610c

SDRAM_CONTROLLER_LAST_ADDR_3

Address: 0xf0006000 + 0x110 = 0xf0006110

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_3

Address: 0xf0006000 + 0x114 = 0xf0006114

SDRAM_CONTROLLER_LAST_ADDR_4

Address: 0xf0006000 + 0x118 = 0xf0006118

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_4

Address: 0xf0006000 + 0x11c = 0xf000611c

SDRAM_CONTROLLER_LAST_ADDR_5

Address: 0xf0006000 + 0x120 = 0xf0006120

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_5

Address: 0xf0006000 + 0x124 = 0xf0006124

SDRAM_CONTROLLER_LAST_ADDR_6

Address: 0xf0006000 + 0x128 = 0xf0006128

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_6

Address: 0xf0006000 + 0x12c = 0xf000612c

SDRAM_CONTROLLER_LAST_ADDR_7

Address: 0xf0006000 + 0x130 = 0xf0006130

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_7

Address: $0xf0006000 + 0x134 = 0xf0006134$

15.2.14 SDRAM_CHECKER

Register Listing for SDRAM_CHECKER

Register	Address
<i>SDRAM_CHECKER_RESET</i>	<i>0xf0006800</i>
<i>SDRAM_CHECKER_START</i>	<i>0xf0006804</i>
<i>SDRAM_CHECKER_DONE</i>	<i>0xf0006808</i>
<i>SDRAM_CHECKER_BASE</i>	<i>0xf000680c</i>
<i>SDRAM_CHECKER_END</i>	<i>0xf0006810</i>
<i>SDRAM_CHECKER_LENGTH</i>	<i>0xf0006814</i>
<i>SDRAM_CHECKER_RANDOM</i>	<i>0xf0006818</i>
<i>SDRAM_CHECKER_TICKS</i>	<i>0xf000681c</i>
<i>SDRAM_CHECKER_ERRORS</i>	<i>0xf0006820</i>

SDRAM_CHECKER_RESET

Address: $0xf0006800 + 0x0 = 0xf0006800$

SDRAM_CHECKER_START

Address: $0xf0006800 + 0x4 = 0xf0006804$

SDRAM_CHECKER_DONE

Address: $0xf0006800 + 0x8 = 0xf0006808$

SDRAM_CHECKER_BASE

Address: $0xf0006800 + 0xc = 0xf000680c$

SDRAM_CHECKER_END

Address: $0xf0006800 + 0x10 = 0xf0006810$

SDRAM_CHECKER_LENGTH

Address: $0xf0006800 + 0x14 = 0xf0006814$

SDRAM_CHECKER_RANDOM

Address: $0xf0006800 + 0x18 = 0xf0006818$

Field	Name	Description

SDRAM_CHECKER_TICKS

Address: $0xf0006800 + 0x1c = 0xf000681c$

SDRAM_CHECKER_ERRORS

Address: $0xf0006800 + 0x20 = 0xf0006820$

15.2.15 SDRAM_GENERATOR

Register Listing for SDRAM_GENERATOR

Register	Address
<i>SDRAM_GENERATOR_RESET</i>	0xf0007000
<i>SDRAM_GENERATOR_START</i>	0xf0007004
<i>SDRAM_GENERATOR_DONE</i>	0xf0007008
<i>SDRAM_GENERATOR_BASE</i>	0xf000700c
<i>SDRAM_GENERATOR_END</i>	0xf0007010
<i>SDRAM_GENERATOR_LENGTH</i>	0xf0007014
<i>SDRAM_GENERATOR_RANDOM</i>	0xf0007018
<i>SDRAM_GENERATOR_TICKS</i>	0xf000701c

SDRAM_GENERATOR_RESET

Address: $0xf0007000 + 0x0 = 0xf0007000$

SDRAM_GENERATOR_START

Address: $0xf0007000 + 0x4 = 0xf0007004$

SDRAM_GENERATOR_DONE

Address: $0xf0007000 + 0x8 = 0xf0007008$

SDRAM_GENERATOR_BASE

Address: $0xf0007000 + 0xc = 0xf000700c$

SDRAM_GENERATOR_END

Address: $0xf0007000 + 0x10 = 0xf0007010$

SDRAM_GENERATOR_LENGTH

Address: $0xf0007000 + 0x14 = 0xf0007014$

SDRAM_GENERATOR_RANDOM

Address: $0xf0007000 + 0x18 = 0xf0007018$

Field	Name	Description

SDRAM_GENERATOR_TICKS

Address: $0xf0007000 + 0x1c = 0xf000701c$

15.2.16 TIMERO

Timer

Provides a generic Timer core.

The Timer is implemented as a countdown timer that can be used in various modes:

- Polling : Returns current countdown value to software
- One-Shot: Loads itself and stops when value reaches 0
- Periodic: (Re-)Loads itself when value reaches 0

en register allows the user to enable/disable the Timer. When the Timer is enabled, it is automatically loaded with the value of *load* register.

When the Timer reaches 0, it is automatically reloaded with value of *reload* register.

The user can latch the current countdown value by writing to *update_value* register, it will update *value* register with current countdown value.

To use the Timer in One-Shot mode, the user needs to:

- Disable the timer

- Set the load register to the expected duration
- (Re-)Enable the Timer

To use the Timer in Periodic mode, the user needs to:

- Disable the Timer
- Set the load register to 0
- Set the reload register to the expected period
- Enable the Timer

For both modes, the CPU can be advertised by an IRQ that the duration/period has elapsed. (The CPU can also do software polling with `update_value` and `value` to know the elapsed duration)

Register Listing for TIMERO

Register	Address
<code>TIMERO_LOAD</code>	<code>0xf0007800</code>
<code>TIMERO_RELOAD</code>	<code>0xf0007804</code>
<code>TIMERO_EN</code>	<code>0xf0007808</code>
<code>TIMERO_UPDATE_VALUE</code>	<code>0xf000780c</code>
<code>TIMERO_VALUE</code>	<code>0xf0007810</code>
<code>TIMERO_EV_STATUS</code>	<code>0xf0007814</code>
<code>TIMERO_EV_PENDING</code>	<code>0xf0007818</code>
<code>TIMERO_EV_ENABLE</code>	<code>0xf000781c</code>

TIMERO_LOAD

Address: $0xf0007800 + 0x0 = 0xf0007800$

Load value when Timer is (re-)enabled. In One-Shot mode, the value written to this register specifies the Timer's duration in clock cycles.

TIMERO_RELOAD

Address: $0xf0007800 + 0x4 = 0xf0007804$

Reload value when Timer reaches 0. In Periodic mode, the value written to this register specify the Timer's period in clock cycles.

TIMER0_EN

Address: $0xf0007800 + 0x8 = 0xf0007808$

Enable flag of the Timer. Set this flag to 1 to enable/start the Timer. Set to 0 to disable the Timer.

TIMER0_UPDATE_VALUE

Address: $0xf0007800 + 0xc = 0xf000780c$

Update trigger for the current countdown value. A write to this register latches the current countdown value to value register.

TIMER0_VALUE

Address: $0xf0007800 + 0x10 = 0xf0007810$

Latched countdown value. This value is updated by writing to update_value.

TIMER0_EV_STATUS

Address: $0xf0007800 + 0x14 = 0xf0007814$

This register contains the current raw level of the zero event trigger. Writes to this register have no effect.

Field	Name	Description
[0]	ZERO	Level of the zero event

TIMER0_EV_PENDING

Address: $0xf0007800 + 0x18 = 0xf0007818$

When a zero event occurs, the corresponding bit will be set in this register. To clear the Event, set the corresponding bit in this register.

Field	Name	Description
[0]	ZERO	1 if a zero event occurred. This Event is triggered on a falling edge.

TIMERO_EV_ENABLE

Address: $0xf0007800 + 0x1c = 0xf000781c$

This register enables the corresponding zero events. Write a 0 to this register to disable individual events.

Field	Name	Description
[0]	ZERO	Write a 1 to enable the zero Event

15.2.17 UART

Register Listing for UART

Register	Address
<i>UART_RXTX</i>	<i>0xf0008000</i>
<i>UART_TXFULL</i>	<i>0xf0008004</i>
<i>UART_RXEMPTY</i>	<i>0xf0008008</i>
<i>UART_EV_STATUS</i>	<i>0xf000800c</i>
<i>UART_EV_PENDING</i>	<i>0xf0008010</i>
<i>UART_EV_ENABLE</i>	<i>0xf0008014</i>
<i>UART_TXEMPTY</i>	<i>0xf0008018</i>
<i>UART_RXFULL</i>	<i>0xf000801c</i>
<i>UART_XOVER_RXTX</i>	<i>0xf0008020</i>
<i>UART_XOVER_TXFULL</i>	<i>0xf0008024</i>
<i>UART_XOVER_RXEMPTY</i>	<i>0xf0008028</i>
<i>UART_XOVER_EV_STATUS</i>	<i>0xf000802c</i>
<i>UART_XOVER_EV_PENDING</i>	<i>0xf0008030</i>
<i>UART_XOVER_EV_ENABLE</i>	<i>0xf0008034</i>
<i>UART_XOVER_TXEMPTY</i>	<i>0xf0008038</i>
<i>UART_XOVER_RXFULL</i>	<i>0xf000803c</i>

UART_RXTX

Address: $0xf0008000 + 0x0 = 0xf0008000$

UART_TXFULL

Address: $0xf0008000 + 0x4 = 0xf0008004$

TX FIFO Full.

UART_RXEMPTY

Address: $0xf0008000 + 0x8 = 0xf0008008$

RX FIFO Empty.

UART_EV_STATUS

Address: $0xf0008000 + 0xc = 0xf000800c$

This register contains the current raw level of the rx event trigger. Writes to this register have no effect.

Field	Name	Description
[0]	TX	Level of the tx event
[1]	RX	Level of the rx event

UART_EV_PENDING

Address: $0xf0008000 + 0x10 = 0xf0008010$

When a rx event occurs, the corresponding bit will be set in this register. To clear the Event, set the corresponding bit in this register.

Field	Name	Description
[0]	TX	1 if a tx event occurred. This Event is triggered on a falling edge.
[1]	RX	1 if a rx event occurred. This Event is triggered on a falling edge.

UART_EV_ENABLE

Address: $0xf0008000 + 0x14 = 0xf0008014$

This register enables the corresponding rx events. Write a 0 to this register to disable individual events.

Field	Name	Description
[0]	TX	Write a 1 to enable the tx Event
[1]	RX	Write a 1 to enable the rx Event

UART_TXEMPTY

Address: $0xf0008000 + 0x18 = 0xf0008018$

TX FIFO Empty.

UART_RXFULL

Address: $0xf0008000 + 0x1c = 0xf000801c$

RX FIFO Full.

UART_XOVER_RXTX

Address: $0xf0008000 + 0x20 = 0xf0008020$

UART_XOVER_TXFULL

Address: $0xf0008000 + 0x24 = 0xf0008024$

TX FIFO Full.

UART_XOVER_RXEMPTY

Address: $0xf0008000 + 0x28 = 0xf0008028$

RX FIFO Empty.

UART_XOVER_EV_STATUS

Address: $0xf0008000 + 0x2c = 0xf000802c$

This register contains the current raw level of the rx event trigger. Writes to this register have no effect.

Field	Name	Description
[0]	TX	Level of the tx event
[1]	RX	Level of the rx event

UART_XOVER_EV_PENDING

Address: $0xf0008000 + 0x30 = 0xf0008030$

When a rx event occurs, the corresponding bit will be set in this register. To clear the Event, set the corresponding bit in this register.

Field	Name	Description
[0]	TX	1 if a tx event occurred. This Event is triggered on a falling edge.
[1]	RX	1 if a rx event occurred. This Event is triggered on a falling edge.

UART_XOVER_EV_ENABLE

Address: $0xf0008000 + 0x34 = 0xf0008034$

This register enables the corresponding rx events. Write a 0 to this register to disable individual events.

Field	Name	Description
[0]	TX	Write a 1 to enable the tx Event
[1]	RX	Write a 1 to enable the rx Event

UART_XOVER_TXEMPTY

Address: $0xf0008000 + 0x38 = 0xf0008038$

TX FIFO Empty.

UART_XOVER_RXFULL

Address: $0xf0008000 + 0x3c = 0xf000803c$

RX FIFO Full.

CHAPTER
SIXTEEN

DOCUMENTATION FOR ROW HAMMER TESTER DDR5 TEST BOARD

16.1 Modules

16.1.1 Interrupt Controller

This device has an EventManager-based interrupt system. Individual modules generate *events* which are wired into a central interrupt controller.

When an interrupt occurs, you should look the interrupt number up in the CPU- specific interrupt table and then call the relevant module.

Assigned Interrupts

The following interrupts are assigned on this system:

Interrupt	Module
1	<i>TIMERO</i>
0	<i>UART</i>

16.2 Register Groups

16.2.1 LEDS

Register Listing for LEDS

Register	Address
<i>LEDS_OUT</i>	<i>0xf0000000</i>

LEDS_OUT

Address: $0xf0000000 + 0x0 = 0xf0000000$

Led Output(s) Control.

16.2.2 DDRPHY

Register Listing for DDRPHY

Register	Address
<i>DDRPHY_CSRMODULE_ENABLE_FIFOS</i>	0xf0000800
<i>DDRPHY_CSRMODULE_RST</i>	0xf0000804
<i>DDRPHY_CSRMODULE_RDIMM_MODE</i>	0xf0000808
<i>DDRPHY_CSRMODULE_RDPHASE</i>	0xf000080c
<i>DDRPHY_CSRMODULE_WRPHASE</i>	0xf0000810
<i>DDRPHY_CSRMODULE_ALERT</i>	0xf0000814
<i>DDRPHY_CSRMODULE_ALERT_REDUCE</i>	0xf0000818
<i>DDRPHY_CSRMODULE_SAMPLE_ALERT</i>	0xf000081c
<i>DDRPHY_CSRMODULE_RESET_ALERT</i>	0xf0000820
<i>DDRPHY_CSRMODULE_CKDLY_RST</i>	0xf0000824
<i>DDRPHY_CSRMODULE_CKDLY_INC</i>	0xf0000828
<i>DDRPHY_CSRMODULE_PREAMBLE</i>	0xf000082c
<i>DDRPHY_CSRMODULE_WLEVEL_EN</i>	0xf0000830
<i>DDRPHY_CSRMODULE_PAR_ENABLE</i>	0xf0000834
<i>DDRPHY_CSRMODULE_PAR_VALUE</i>	0xf0000838
<i>DDRPHY_CSRMODULE_DISCARD_RD_FIFO</i>	0xf000083c
<i>DDRPHY_CSRMODULE_DL_Y_SEL</i>	0xf0000840
<i>DDRPHY_CSRMODULE_DQ_DQS_RATIO</i>	0xf0000844
<i>DDRPHY_CSRMODULE_CK_RDLY_INC</i>	0xf0000848
<i>DDRPHY_CSRMODULE_CK_RDLY_RST</i>	0xf000084c
<i>DDRPHY_CSRMODULE_CK_RDDLY</i>	0xf0000850
<i>DDRPHY_CSRMODULE_CK_RDDLY_PREAMBLE</i>	0xf0000854
<i>DDRPHY_CSRMODULE_CK_WDLY_INC</i>	0xf0000858
<i>DDRPHY_CSRMODULE_CK_WDLY_RST</i>	0xf000085c
<i>DDRPHY_CSRMODULE_CK_WDLY_DQS</i>	0xf0000860
<i>DDRPHY_CSRMODULE_CK_WDDLY_INC</i>	0xf0000864
<i>DDRPHY_CSRMODULE_CK_WDDLY_RST</i>	0xf0000868
<i>DDRPHY_CSRMODULE_CK_WDLY_DQ</i>	0xf000086c
<i>DDRPHY_CSRMODULE_DQ_DL_Y_SEL</i>	0xf0000870
<i>DDRPHY_CSRMODULE_CSDLY_RST</i>	0xf0000874
<i>DDRPHY_CSRMODULE_CSDLY_INC</i>	0xf0000878
<i>DDRPHY_CSRMODULE_CADLY_RST</i>	0xf000087c
<i>DDRPHY_CSRMODULE_CADLY_INC</i>	0xf0000880
<i>DDRPHY_CSRMODULE_PARDLY_RST</i>	0xf0000884
<i>DDRPHY_CSRMODULE_PARDLY_INC</i>	0xf0000888
<i>DDRPHY_CSRMODULE_CSDLY</i>	0xf000088c
<i>DDRPHY_CSRMODULE_CADLY</i>	0xf0000890

continues on next page

Table 16.1 – continued from previous page

Register	Address
<i>DDRPHY_CSRMODULE_RDLY_DQ_RST</i>	0xf0000894
<i>DDRPHY_CSRMODULE_RDLY_DQ_INC</i>	0xf0000898
<i>DDRPHY_CSRMODULE_RDLY_DQS_RST</i>	0xf000089c
<i>DDRPHY_CSRMODULE_RDLY_DQS_INC</i>	0xf00008a0
<i>DDRPHY_CSRMODULE_RDLY_DQS</i>	0xf00008a4
<i>DDRPHY_CSRMODULE_RDLY_DQ</i>	0xf00008a8
<i>DDRPHY_CSRMODULE_WDLY_DQ_RST</i>	0xf00008ac
<i>DDRPHY_CSRMODULE_WDLY_DQ_INC</i>	0xf00008b0
<i>DDRPHY_CSRMODULE_WDLY_DM_RST</i>	0xf00008b4
<i>DDRPHY_CSRMODULE_WDLY_DM_INC</i>	0xf00008b8
<i>DDRPHY_CSRMODULE_WDLY_DQS_RST</i>	0xf00008bc
<i>DDRPHY_CSRMODULE_WDLY_DQS_INC</i>	0xf00008c0
<i>DDRPHY_CSRMODULE_WDLY_DQS</i>	0xf00008c4
<i>DDRPHY_CSRMODULE_WDLY_DQ</i>	0xf00008c8
<i>DDRPHY_CSRMODULE_WDLY_DM</i>	0xf00008cc

DDRPHY_CSRMODULE_ENABLE_FIFOS

Address: 0xf0000800 + 0x0 = 0xf0000800

DDRPHY_CSRMODULE_RST

Address: 0xf0000800 + 0x4 = 0xf0000804

DDRPHY_CSRMODULE_RDIMM_MODE

Address: 0xf0000800 + 0x8 = 0xf0000808

DDRPHY_CSRMODULE_RDPHASE

Address: 0xf0000800 + 0xc = 0xf000080c

DDRPHY_CSRMODULE_WRPAGE

Address: $0xf0000800 + 0x10 = 0xf0000810$

DDRPHY_CSRMODULE_ALERT

Address: $0xf0000800 + 0x14 = 0xf0000814$

DDRPHY_CSRMODULE_ALERT_REDUCE

Address: $0xf0000800 + 0x18 = 0xf0000818$

Field	Name	Description
[0]	INITIAL_STATE	Initial value of all bits
[1]	OPERATION	0 - or (default), 1 -and

DDRPHY_CSRMODULE_SAMPLE_ALERT

Address: $0xf0000800 + 0x1c = 0xf000081c$

DDRPHY_CSRMODULE_RESET_ALERT

Address: $0xf0000800 + 0x20 = 0xf0000820$

DDRPHY_CSRMODULE_CKDLY_RST

Address: $0xf0000800 + 0x24 = 0xf0000824$

DDRPHY_CSRMODULE_CKDLY_INC

Address: $0xf0000800 + 0x28 = 0xf0000828$

DDRPHY_CSRMODULE_PREAMBLE

Address: $0xf0000800 + 0x2c = 0xf000082c$

DDRPHY_CSRMODULE_WLEVEL_EN

Address: $0xf0000800 + 0x30 = 0xf0000830$

DDRPHY_CSRMODULE_PAR_ENABLE

Address: $0xf0000800 + 0x34 = 0xf0000834$

DDRPHY_CSRMODULE_PAR_VALUE

Address: $0xf0000800 + 0x38 = 0xf0000838$

DDRPHY_CSRMODULE_DISCARD_RD_FIFO

Address: $0xf0000800 + 0x3c = 0xf000083c$

DDRPHY_CSRMODULE_DLY_SEL

Address: $0xf0000800 + 0x40 = 0xf0000840$

DDRPHY_CSRMODULE_DQ_DQS_RATIO

Address: 0xf0000800 + 0x44 = 0xf0000844

DDRPHY_CSRMODULE_CK_RDLY_INC

Address: 0xf0000800 + 0x48 = 0xf0000848

DDRPHY_CSRMODULE_CK_RDLY_RST

Address: 0xf0000800 + 0x4c = 0xf000084c

DDRPHY_CSRMODULE_CK_RDDLY

Address: 0xf0000800 + 0x50 = 0xf0000850

DDRPHY_CSRMODULE_CK_RDDLY_PREAMBLE

Address: 0xf0000800 + 0x54 = 0xf0000854

DDRPHY_CSRMODULE_CK_WDLY_INC

Address: 0xf0000800 + 0x58 = 0xf0000858

DDRPHY_CSRMODULE_CK_WDLY_RST

Address: 0xf0000800 + 0x5c = 0xf000085c

DDRPHY_CSRMODULE_CK_WDLY_DQS

Address: 0xf0000800 + 0x60 = 0xf0000860

DDRPHY_CSRMODULE_CK_WDDLY_INC

Address: 0xf0000800 + 0x64 = 0xf0000864

DDRPHY_CSRMODULE_CK_WDDLY_RST

Address: 0xf0000800 + 0x68 = 0xf0000868

DDRPHY_CSRMODULE_CK_WDLY_DQ

Address: 0xf0000800 + 0x6c = 0xf000086c

DDRPHY_CSRMODULE_DQ_DLY_SEL

Address: 0xf0000800 + 0x70 = 0xf0000870

DDRPHY_CSRMODULE_CSDLY_RST

Address: 0xf0000800 + 0x74 = 0xf0000874

DDRPHY_CSRMODULE_CSDLY_INC

Address: 0xf0000800 + 0x78 = 0xf0000878

DDRPHY_CSRMODULE_CADLY_RST

Address: 0xf0000800 + 0x7c = 0xf000087c

DDRPHY_CSRMODULE_CADLY_INC

Address: 0xf0000800 + 0x80 = 0xf0000880

DDRPHY_CSRMODULE_PARDLY_RST

Address: 0xf0000800 + 0x84 = 0xf0000884

DDRPHY_CSRMODULE_PARDLY_INC

Address: 0xf0000800 + 0x88 = 0xf0000888

DDRPHY_CSRMODULE_CSDLY

Address: 0xf0000800 + 0x8c = 0xf000088c

DDRPHY_CSRMODULE_CADLY

Address: 0xf0000800 + 0x90 = 0xf0000890

DDRPHY_CSRMODULE_RDLY_DQ_RST

Address: 0xf0000800 + 0x94 = 0xf0000894

DDRPHY_CSRMODULE_RDLY_DQ_INC

Address: 0xf0000800 + 0x98 = 0xf0000898

DDRPHY_CSRMODULE_RDLY_DQS_RST

Address: 0xf0000800 + 0x9c = 0xf000089c

DDRPHY_CSRMODULE_RDLY_DQS_INC

Address: 0xf0000800 + 0xa0 = 0xf00008a0

DDRPHY_CSRMODULE_RDLY_DQS

Address: 0xf0000800 + 0xa4 = 0xf00008a4

DDRPHY_CSRMODULE_RDLY_DQ

Address: 0xf0000800 + 0xa8 = 0xf00008a8

DDRPHY_CSRMODULE_WDLY_DQ_RST

Address: 0xf0000800 + 0xac = 0xf00008ac

DDRPHY_CSRMODULE_WDLY_DQ_INC

Address: 0xf0000800 + 0xb0 = 0xf00008b0

DDRPHY_CSRMODULE_WDLY_DM_RST

Address: 0xf0000800 + 0xb4 = 0xf00008b4

DDRPHY_CSRMODULE_WDLY_DM_INC

Address: 0xf0000800 + 0xb8 = 0xf00008b8

DDRPHY_CSRMODULE_WDLY_DQS_RST

Address: 0xf0000800 + 0xbc = 0xf00008bc

DDRPHY_CSRMODULE_WDLY_DQS_INC

Address: 0xf0000800 + 0xc0 = 0xf00008c0

DDRPHY_CSRMODULE_WDLY_DQS

Address: 0xf0000800 + 0xc4 = 0xf00008c4

DDRPHY_CSRMODULE_WDLY_DQ

Address: 0xf0000800 + 0xc8 = 0xf00008c8

DDRPHY_CSRMODULE_WDLY_DM

Address: 0xf0000800 + 0xcc = 0xf00008cc

16.2.3 CONTROLLER_SETTINGS

Allows to change LiteDRAMController behaviour at runtime

Register Listing for CONTROLLER_SETTINGS

Register	Address
<i>CONTROLLER_SETTINGS_REFRESH</i>	<i>0xf0001000</i>

CONTROLLER_SETTINGS_REFRESH

Address: $0xf0001000 + 0x0 = 0xf0001000$

Enable/disable Refresh commands sending

16.2.4 DDRCTRL

Register Listing for DDRCTRL

Register	Address
<i>DDRCTRL_INIT_DONE</i>	<i>0xf0001800</i>
<i>DDRCTRL_INIT_ERROR</i>	<i>0xf0001804</i>

DDRCTRL_INIT_DONE

Address: $0xf0001800 + 0x0 = 0xf0001800$

DDRCTRL_INIT_ERROR

Address: $0xf0001800 + 0x4 = 0xf0001804$

16.2.5 ROWHAMMER

Row Hammer DMA attacker

This module allows to perform a Row Hammer attack by configuring it with two addresses that map to different rows of a single bank. When enabled, it will perform alternating DMA reads from the given locations, which will result in the DRAM controller having to repeatedly open/close rows at each read access.

Register Listing for ROWHAMMER

Register	Address
<code>ROWHAMMER_ENABLED</code>	<code>0xf0002000</code>
<code>ROWHAMMER_ADDRESS1</code>	<code>0xf0002004</code>
<code>ROWHAMMER_ADDRESS2</code>	<code>0xf0002008</code>
<code>ROWHAMMER_COUNT</code>	<code>0xf000200c</code>

`ROWHAMMER_ENABLED`

Address: $0xf0002000 + 0x0 = 0xf0002000$

Used to start/stop the operation of the module

`ROWHAMMER_ADDRESS1`

Address: $0xf0002000 + 0x4 = 0xf0002004$

First attacked address

`ROWHAMMER_ADDRESS2`

Address: $0xf0002000 + 0x8 = 0xf0002008$

Second attacked address

`ROWHAMMER_COUNT`

Address: $0xf0002000 + 0xc = 0xf000200c$

This is the number of DMA accesses performed. When the module is enabled, the value can be freely read. When the module is disabled, the register is clear-on-write and has to be read before the next attack.

16.2.6 WRITER

DMA DRAM writer.

Allows to fill DRAM with a predefined pattern using DMA.

Pattern

Provides access to RAM to store access pattern: *mem_addr* and *mem_data*. The pattern address space can be limited using the *data_mask*.

For example, having *mem_addr* filled with [0x04, 0x02, 0x03, ...] and *mem_data* filled with [0xff, 0xaa, 0x55, ...] and setting *data_mask* = 0b01, the pattern [(address, data), ...] written will be: [(0x04, 0xff), (0x02, 0xaa), (0x04, 0xff), ...] (wraps due to masking).

DRAM memory range that is being accessed can be configured using *mem_mask*.

To use this module, make sure that *ready* is 1, then write the desired number of transfers to *count*. Writing to the *start* CSR will initialize the operation. When the operation is ongoing *ready* will be 0.

Register Listing for WRITER

Register	Address
WRITER_START	0xf0002800
WRITER_READY	0xf0002804
WRITER_MODULO	0xf0002808
WRITER_COUNT	0xf000280c
WRITER_DONE	0xf0002810
WRITER_MEM_MASK	0xf0002814
WRITER_DATA_MASK	0xf0002818
WRITER_DATA_DIV	0xf000281c
WRITER_INVERTER_DIVISOR_MASK	0xf0002820
WRITER_INVERTER_SELECTION_MASK	0xf0002824
WRITER_LAST_ADDRESS	0xf0002828

WRITER_START

Address: 0xf0002800 + 0x0 = 0xf0002800

Write to the register starts the transfer (if ready=1)

WRITER_READY

Address: 0xf0002800 + 0x4 = 0xf0002804

Indicates that the transfer is not ongoing

WRITER_MODULO

Address: $0xf0002800 + 0x8 = 0xf0002808$

When set use modulo to calculate DMA transfers address rather than bit masking

WRITER_COUNT

Address: $0xf0002800 + 0xc = 0xf000280c$

Desired number of DMA transfers

WRITER_DONE

Address: $0xf0002800 + 0x10 = 0xf0002810$

Number of completed DMA transfers

WRITER_MEM_MASK

Address: $0xf0002800 + 0x14 = 0xf0002814$

DRAM address mask for DMA transfers

WRITER_DATA_MASK

Address: $0xf0002800 + 0x18 = 0xf0002818$

Pattern memory address mask

WRITER_DATA_DIV

Address: $0xf0002800 + 0x1c = 0xf000281c$

Pattern memory address divisor-1

WRITER_INVERTER_DIVISOR_MASK

Address: $0xf0002800 + 0x20 = 0xf0002820$

Divisor mask for selecting rows for which pattern data gets inverted

WRITER_INVERTER_SELECTION_MASK

Address: $0xf0002800 + 0x24 = 0xf0002824$

Selection mask for selecting rows for which pattern data gets inverted

WRITER_LAST_ADDRESS

Address: $0xf0002800 + 0x28 = 0xf0002828$

Number of completed DMA transfers

16.2.7 READER

DMA DRAM reader.

Allows to check DRAM contents against a predefined pattern using DMA.

Pattern

Provides access to RAM to store access pattern: *mem_addr* and *mem_data*. The pattern address space can be limited using the *data_mask*.

For example, having *mem_addr* filled with [0x04, 0x02, 0x03, ...] and *mem_data* filled with [0xff, 0xaa, 0x55, ...] and setting *data_mask* = 0b01, the pattern [(address, data), ...] written will be: [(0x04, 0xff), (0x02, 0xaa), (0x04, 0xff), ...] (wraps due to masking).

DRAM memory range that is being accessed can be configured using *mem_mask*.

To use this module, make sure that *ready* is 1, then write the desired number of transfers to *count*. Writing to the *start* CSR will initialize the operation. When the operation is ongoing *ready* will be 0.

Reading errors

This module allows to check the locations of errors in the memory. It scans the configured memory area and compares the values read to the predefined pattern. If *skip_fifo* is 0, this module will stop after each error encountered, so that it can be examined. Wait until the *error_ready* CSR is 1. Then use the CSRs *error_offset*, *error_data* and *error_expected* to examine the errors in the current transfer. To continue reading, write 1 to *error_continue* CSR. Setting *skip_fifo* to 1 will disable this behaviour entirely.

The final number of errors can be read from *error_count*. NOTE: This value represents the number of erroneous DMA transfers.

The current progress can be read from the *done* CSR.

Register Listing for READER

Register	Address
<i>READER_START</i>	0xf0003000
<i>READER_READY</i>	0xf0003004
<i>READER_MODULO</i>	0xf0003008
<i>READER_COUNT</i>	0xf000300c
<i>READER_DONE</i>	0xf0003010
<i>READER_MEM_MASK</i>	0xf0003014
<i>READER_DATA_MASK</i>	0xf0003018
<i>READER_DATA_DIV</i>	0xf000301c
<i>READER_INVERTER_DIVISOR_MASK</i>	0xf0003020
<i>READER_INVERTER_SELECTION_MASK</i>	0xf0003024
<i>READER_ERROR_COUNT</i>	0xf0003028
<i>READER_SKIP_FIFO</i>	0xf000302c
<i>READER_ERROR_OFFSET</i>	0xf0003030
<i>READER_ERROR_DATA1</i>	0xf0003034
<i>READER_ERROR_DATA0</i>	0xf0003038
<i>READER_ERROR_EXPECTED1</i>	0xf000303c
<i>READER_ERROR_EXPECTED0</i>	0xf0003040
<i>READER_ERROR_READY</i>	0xf0003044
<i>READER_ERROR_CONTINUE</i>	0xf0003048

READER_START

Address: 0xf0003000 + 0x0 = 0xf0003000

Write to the register starts the transfer (if ready=1)

READER_READY

Address: 0xf0003000 + 0x4 = 0xf0003004

Indicates that the transfer is not ongoing

READER_MODULO

Address: 0xf0003000 + 0x8 = 0xf0003008

When set use modulo to calculate DMA transfers address rather than bit masking

READER_COUNT

Address: $0xf0003000 + 0xc = 0xf000300c$

Desired number of DMA transfers

READER_DONE

Address: $0xf0003000 + 0x10 = 0xf0003010$

Number of completed DMA transfers

READER_MEM_MASK

Address: $0xf0003000 + 0x14 = 0xf0003014$

DRAM address mask for DMA transfers

READER_DATA_MASK

Address: $0xf0003000 + 0x18 = 0xf0003018$

Pattern memory address mask

READER_DATA_DIV

Address: $0xf0003000 + 0x1c = 0xf000301c$

Pattern memory address divisor-1

READER_INVERTER_DIVISOR_MASK

Address: $0xf0003000 + 0x20 = 0xf0003020$

Divisor mask for selecting rows for which pattern data gets inverted

READER_INVERTER_SELECTION_MASK

Address: $0xf0003000 + 0x24 = 0xf0003024$

Selection mask for selecting rows for which pattern data gets inverted

READER_ERROR_COUNT

Address: $0xf0003000 + 0x28 = 0xf0003028$

Number of errors detected

READER_SKIP_FIFO

Address: $0xf0003000 + 0x2c = 0xf000302c$

Skip waiting for user to read the errors FIFO

READER_ERROR_OFFSET

Address: $0xf0003000 + 0x30 = 0xf0003030$

Current offset of the error

READER_ERROR_DATA1

Address: $0xf0003000 + 0x34 = 0xf0003034$

Bits 32-63 of *READER_ERROR_DATA*. Erroneous value read from DRAM memory

READER_ERROR_DATA0

Address: $0xf0003000 + 0x38 = 0xf0003038$

Bits 0-31 of *READER_ERROR_DATA*.

READER_ERROR_EXPECTED1

Address: $0xf0003000 + 0x3c = 0xf000303c$

Bits 32-63 of *READER_ERROR_EXPECTED*. Value expected to be read from DRAM memory

READER_ERROR_EXPECTED0

Address: $0xf0003000 + 0x40 = 0xf0003040$

Bits 0-31 of *READER_ERROR_EXPECTED*.

READER_ERROR_READY

Address: $0xf0003000 + 0x44 = 0xf0003044$

Error detected and ready to read

READER_ERROR_CONTINUE

Address: $0xf0003000 + 0x48 = 0xf0003048$

Continue reading until the next error

16.2.8 DFI_SWITCH

Register Listing for DFI_SWITCH

Register	Address
<i>DFI_SWITCH_REFRESH_COUNT</i>	<i>0xf0003800</i>
<i>DFI_SWITCH_AT_REFRESH</i>	<i>0xf0003804</i>
<i>DFI_SWITCH_REFRESH_UPDATE</i>	<i>0xf0003808</i>

DFI_SWITCH_REFRESH_COUNT

Address: $0xf0003800 + 0x0 = 0xf0003800$

Count of all refresh commands issued (both by Memory Controller and Payload Executor). Value is latched from internal counter on mode transition: MC -> PE or by writing to the *refresh_update* CSR.

DFI_SWITCH_AT_REFRESH

Address: $0xf0003800 + 0x4 = 0xf0003804$

If set to a value different than 0 the mode transition MC -> PE will be performed only when the value of this register matches the current refresh commands count.

DFI_SWITCH_REFRESH_UPDATE

Address: $0xf0003800 + 0x8 = 0xf0003808$

Force an update of the *refresh_count* CSR.

16.2.9 PAYLOAD_EXECUTOR

Executes the DRAM payload from memory

Instruction decoder

All instructions are 32-bit. The format of most instructions is the same, except for the LOOP instruction, which has a constant TIMESLICE of 1.

NOOP with a TIMESLICE of 0 is a special case which is interpreted as STOP instruction. When this instruction is encountered execution gets finished immediately.

NOTE: TIMESLICE is the number of cycles the instruction will take. This means that instructions other than NOOP that use TIMESLICE=0 are illegal (although will silently be executed as having TIMESLICE=1).

NOTE2: LOOP instruction will *jump* COUNT times, meaning that the “code” inside the loop will effectively be executed COUNT+1 times.

Op codes:

Op	Value
NOOP	0b000
LOOP	0b111
ACT	0b100
PRE	0b101
REF	0b110
ZQC	0b001
READ	0b010

Instruction format:

LSB	MSB
dfi: OP_CODE TIMESLICE ADDRESS	
noop: OP_CODE TIMESLICE_NOOP	
loop: OP_CODE COUNT JUMP	
stop: <NOOP> 0	

Where ADDRESS depends on the DFI command and is one of:

LSB	MSB
RANK BANK COLUMN	
RANK BANK ROW	

Register Listing for PAYLOAD_EXECUTOR

Register	Address
PAYLOAD_EXECUTOR_START	0xf0004000
PAYLOAD_EXECUTOR_STATUS	0xf0004004
PAYLOAD_EXECUTOR_READ_COUNT	0xf0004008

PAYLOAD_EXECUTOR_START

Address: $0xf0004000 + 0x0 = 0xf0004000$

Writing to this register initializes payload execution

PAYLOAD_EXECUTOR_STATUS

Address: $0xf0004000 + 0x4 = 0xf0004004$

Payload executor status register

Field	Name	Description
[0]	READY	Indicates that the executor is not running
[1]	OVERFLOW	Indicates the scratchpad memory address counter has overflowed due to the number of READ commands sent during execution

PAYLOAD_EXECUTOR_READ_COUNT

Address: $0xf0004000 + 0x8 = 0xf0004008$

Number of data from READ commands that is stored in the scratchpad memory

16.2.10 CTRL

Register Listing for CTRL

Register	Address
CTRL_RESET	0xf0004800
CTRL_SCRATCH	0xf0004804
CTRL_BUS_ERRORS	0xf0004808

CTRL__RESET

Address: $0xf0004800 + 0x0 = 0xf0004800$

Field	Name	Description
[0]	SOC_RST	Write 1 to this register to reset the full SoC (Pulse Reset)
[1]	CPU_RST	Write 1 to this register to reset the CPU(s) of the SoC (Hold Reset)

CTRL_SCRATCH

Address: $0xf0004800 + 0x4 = 0xf0004804$

Use this register as a scratch space to verify that software read/write accesses to the Wishbone/CSR bus are working correctly. The initial reset value of 0x1234578 can be used to verify endianness.

CTRL_BUS_ERRORS

Address: $0xf0004800 + 0x8 = 0xf0004808$

Total number of Wishbone bus errors (timeouts) since start.

16.2.11 ETPHY

Register Listing for ETPHY

Register	Address
ETPHY_CRG_RESET	<i>0xf0005000</i>
ETPHY_MDIO_W	<i>0xf0005004</i>
ETPHY_MDIO_R	<i>0xf0005008</i>

ETPHY_CRG_RESET

Address: $0xf0005000 + 0x0 = 0xf0005000$

ETHPHY_MDIO_W

Address: $0xf0005000 + 0x4 = 0xf0005004$

Field	Name	Description

ETHPHY_MDIO_R

Address: $0xf0005000 + 0x8 = 0xf0005008$

Field	Name	Description

16.2.12 IDENTIFIER_MEM

Register Listing for IDENTIFIER_MEM

Register	Address
<i>IDENTIFIER_MEM</i>	<i>0xf0005800</i>

IDENTIFIER_MEM

Address: $0xf0005800 + 0x0 = 0xf0005800$

8 x 108-bit memory

16.2.13 MAIN

Register Listing for MAIN

Register	Address
<i>MAIN_DQ_REMAPPING1</i>	<i>0xf0006000</i>
<i>MAIN_DQ_REMAPPING0</i>	<i>0xf0006004</i>

MAIN_DQ_REMAPPING1

Address: $0xf0006000 + 0x0 = 0xf0006000$

Bits 32-63 of *MAIN_DQ_REMAPPING*.

MAIN_DQ_REMAPPING0

Address: $0xf0006000 + 0x4 = 0xf0006004$

Bits 0-31 of *MAIN_DQ_REMAPPING*.

16.2.14 SDRAM

Register Listing for SDRAM

Register	Address
<i>SDRAM_DFII_CONTROL</i>	<i>0xf0006800</i>
<i>SDRAM_DFII_FORCE_ISSUE</i>	<i>0xf0006804</i>
<i>SDRAM_DFII_CMDINJECTOR_COMMAND_STORAGE</i>	<i>0xf0006808</i>
<i>SDRAM_DFII_CMDINJECTOR_COMMAND_STORAGE_WR_MASK</i>	<i>0xf000680c</i>
<i>SDRAM_DFII_CMDINJECTOR_PHASE_ADDR</i>	<i>0xf0006810</i>
<i>SDRAM_DFII_CMDINJECTOR_STORE_CONTINUOUS_CMD</i>	<i>0xf0006814</i>
<i>SDRAM_DFII_CMDINJECTOR_STORE_SINGLESHT_CMD</i>	<i>0xf0006818</i>
<i>SDRAM_DFII_CMDINJECTOR_SINGLE_SHOT</i>	<i>0xf000681c</i>
<i>SDRAM_DFII_CMDINJECTOR_ISSUE_COMMAND</i>	<i>0xf0006820</i>
<i>SDRAM_DFII_CMDINJECTOR_WRDATASELECT</i>	<i>0xf0006824</i>
<i>SDRAM_DFII_CMDINJECTOR_WRDATA</i>	<i>0xf0006828</i>
<i>SDRAM_DFII_CMDINJECTOR_WRDATAS</i>	<i>0xf000682c</i>
<i>SDRAM_DFII_CMDINJECTOR_WRDATASTORE</i>	<i>0xf0006830</i>
<i>SDRAM_DFII_CMDINJECTOR_SETUP</i>	<i>0xf0006834</i>
<i>SDRAM_DFII_CMDINJECTOR_SAMPLE</i>	<i>0xf0006838</i>
<i>SDRAM_DFII_CMDINJECTOR_RESULT_ARRAY</i>	<i>0xf000683c</i>
<i>SDRAM_DFII_CMDINJECTOR_RESET</i>	<i>0xf0006840</i>
<i>SDRAM_DFII_CMDINJECTOR_RDDATASELECT</i>	<i>0xf0006844</i>
<i>SDRAM_DFII_CMDINJECTOR_RDDATACAPTURE_CNT</i>	<i>0xf0006848</i>
<i>SDRAM_DFII_CMDINJECTOR_RDDATA</i>	<i>0xf000684c</i>
<i>SDRAM_CONTROLLER_TRP</i>	<i>0xf0006850</i>
<i>SDRAM_CONTROLLER_TRCD</i>	<i>0xf0006854</i>
<i>SDRAM_CONTROLLER_TWR</i>	<i>0xf0006858</i>
<i>SDRAM_CONTROLLER_TWTR</i>	<i>0xf000685c</i>
<i>SDRAM_CONTROLLER_TREFI</i>	<i>0xf0006860</i>
<i>SDRAM_CONTROLLER_TRFC</i>	<i>0xf0006864</i>
<i>SDRAM_CONTROLLER_TFAW</i>	<i>0xf0006868</i>
<i>SDRAM_CONTROLLER_TCCD</i>	<i>0xf000686c</i>
<i>SDRAM_CONTROLLER_TCCD_WR</i>	<i>0xf0006870</i>
<i>SDRAM_CONTROLLER_TRTP</i>	<i>0xf0006874</i>
<i>SDRAM_CONTROLLER_TRRD</i>	<i>0xf0006878</i>

continues on next page

Table 16.2 – continued from previous page

Register	Address
<i>SDRAM_CONTROLLER_TRC</i>	0xf000687c
<i>SDRAM_CONTROLLER_TRAS</i>	0xf0006880
<i>SDRAM_CONTROLLER_LAST_ADDR_0</i>	0xf0006884
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_0</i>	0xf0006888
<i>SDRAM_CONTROLLER_LAST_ADDR_1</i>	0xf000688c
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_1</i>	0xf0006890
<i>SDRAM_CONTROLLER_LAST_ADDR_2</i>	0xf0006894
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_2</i>	0xf0006898
<i>SDRAM_CONTROLLER_LAST_ADDR_3</i>	0xf000689c
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_3</i>	0xf00068a0
<i>SDRAM_CONTROLLER_LAST_ADDR_4</i>	0xf00068a4
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_4</i>	0xf00068a8
<i>SDRAM_CONTROLLER_LAST_ADDR_5</i>	0xf00068ac
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_5</i>	0xf00068b0
<i>SDRAM_CONTROLLER_LAST_ADDR_6</i>	0xf00068b4
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_6</i>	0xf00068b8
<i>SDRAM_CONTROLLER_LAST_ADDR_7</i>	0xf00068bc
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_7</i>	0xf00068c0
<i>SDRAM_CONTROLLER_LAST_ADDR_8</i>	0xf00068c4
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_8</i>	0xf00068c8
<i>SDRAM_CONTROLLER_LAST_ADDR_9</i>	0xf00068cc
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_9</i>	0xf00068d0
<i>SDRAM_CONTROLLER_LAST_ADDR_10</i>	0xf00068d4
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_10</i>	0xf00068d8
<i>SDRAM_CONTROLLER_LAST_ADDR_11</i>	0xf00068dc
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_11</i>	0xf00068e0
<i>SDRAM_CONTROLLER_LAST_ADDR_12</i>	0xf00068e4
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_12</i>	0xf00068e8
<i>SDRAM_CONTROLLER_LAST_ADDR_13</i>	0xf00068ec
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_13</i>	0xf00068f0
<i>SDRAM_CONTROLLER_LAST_ADDR_14</i>	0xf00068f4
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_14</i>	0xf00068f8
<i>SDRAM_CONTROLLER_LAST_ADDR_15</i>	0xf00068fc
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_15</i>	0xf0006900
<i>SDRAM_CONTROLLER_LAST_ADDR_16</i>	0xf0006904
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_16</i>	0xf0006908
<i>SDRAM_CONTROLLER_LAST_ADDR_17</i>	0xf000690c
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_17</i>	0xf0006910
<i>SDRAM_CONTROLLER_LAST_ADDR_18</i>	0xf0006914
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_18</i>	0xf0006918
<i>SDRAM_CONTROLLER_LAST_ADDR_19</i>	0xf000691c
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_19</i>	0xf0006920
<i>SDRAM_CONTROLLER_LAST_ADDR_20</i>	0xf0006924
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_20</i>	0xf0006928
<i>SDRAM_CONTROLLER_LAST_ADDR_21</i>	0xf000692c
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_21</i>	0xf0006930
<i>SDRAM_CONTROLLER_LAST_ADDR_22</i>	0xf0006934
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_22</i>	0xf0006938

continues on next page

Table 16.2 – continued from previous page

Register	Address
<i>SDRAM_CONTROLLER_LAST_ADDR_23</i>	0xf000693c
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_23</i>	0xf0006940
<i>SDRAM_CONTROLLER_LAST_ADDR_24</i>	0xf0006944
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_24</i>	0xf0006948
<i>SDRAM_CONTROLLER_LAST_ADDR_25</i>	0xf000694c
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_25</i>	0xf0006950
<i>SDRAM_CONTROLLER_LAST_ADDR_26</i>	0xf0006954
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_26</i>	0xf0006958
<i>SDRAM_CONTROLLER_LAST_ADDR_27</i>	0xf000695c
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_27</i>	0xf0006960
<i>SDRAM_CONTROLLER_LAST_ADDR_28</i>	0xf0006964
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_28</i>	0xf0006968
<i>SDRAM_CONTROLLER_LAST_ADDR_29</i>	0xf000696c
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_29</i>	0xf0006970
<i>SDRAM_CONTROLLER_LAST_ADDR_30</i>	0xf0006974
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_30</i>	0xf0006978
<i>SDRAM_CONTROLLER_LAST_ADDR_31</i>	0xf000697c
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_31</i>	0xf0006980

SDRAM_DFII_CONTROL

Address: 0xf0006800 + 0x0 = 0xf0006800

Control DFI signals common to all phases

Field	Name	Description						
[0]	SEL	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>0b0</td><td>Software (CPU) control.</td></tr> <tr> <td>0b1</td><td>Hardware control (default).</td></tr> </tbody> </table>	Value	Description	0b0	Software (CPU) control.	0b1	Hardware control (default).
Value	Description							
0b0	Software (CPU) control.							
0b1	Hardware control (default).							
[1]	CKE	DFI clock enable bus						
[2]	ODT	DFI on-die termination bus						
[3]	RESET_N	DFI clock reset bus						
[4]	MODE_2N	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>0b0</td><td>In 1N mode</td></tr> <tr> <td>0b1</td><td>In 2N mode (Default)</td></tr> </tbody> </table>	Value	Description	0b0	In 1N mode	0b1	In 2N mode (Default)
Value	Description							
0b0	In 1N mode							
0b1	In 2N mode (Default)							
[5]	CONTROL	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>0b1</td><td>Cmd Injector</td></tr> </tbody> </table>	Value	Description	0b1	Cmd Injector		
Value	Description							
0b1	Cmd Injector							

SDRAM_DFII_FORCE_ISSUE

Address: $0xf0006800 + 0x4 = 0xf0006804$

SDRAM_DFII_CMDINJECTOR_COMMAND_STORAGE

Address: $0xf0006800 + 0x8 = 0xf0006808$

DDR5 command and control signals

Field	Name	Description
[13:0]	CA	Command/Address bus
[14]	CS	DFI chip select bus

SDRAM_DFII_CMDINJECTOR_COMMAND_STORAGE_WR_MASK

Address: 0xf0006800 + 0xc = 0xf000680c

DDR5 wrdata mask control signals

Field	Name	Description

SDRAM_DFII_CMDINJECTOR_PHASE_ADDR

Address: 0xf0006800 + 0x10 = 0xf0006810

SDRAM_DFII_CMDINJECTOR_STORE_CONTINUOUS_CMD

Address: 0xf0006800 + 0x14 = 0xf0006814

SDRAM_DFII_CMDINJECTOR_STORE_SINGLESHTOT_CMD

Address: 0xf0006800 + 0x18 = 0xf0006818

SDRAM_DFII_CMDINJECTOR_SINGLE_SHOT

Address: 0xf0006800 + 0x1c = 0xf000681c

SDRAM_DFII_CMDINJECTOR_ISSUE_COMMAND

Address: 0xf0006800 + 0x20 = 0xf0006820

SDRAM_DFII_CMDINJECTOR_WRDATA_SELECT

Address: 0xf0006800 + 0x24 = 0xf0006824

SDRAM_DFII_CMDINJECTOR_WRDATA

Address: 0xf0006800 + 0x28 = 0xf0006828

SDRAM_DFII_CMDINJECTOR_WRDATA_S

Address: 0xf0006800 + 0x2c = 0xf000682c

SDRAM_DFII_CMDINJECTOR_WRDATA_STORE

Address: 0xf0006800 + 0x30 = 0xf0006830

SDRAM_DFII_CMDINJECTOR_SETUP

Address: 0xf0006800 + 0x34 = 0xf0006834

Field	Name	Description
[0]	INITIAL_STATE	Initial value of all bits
[1]	OPERATION	0 - or (default), 1 -and

SDRAM_DFII_CMDINJECTOR_SAMPLE

Address: 0xf0006800 + 0x38 = 0xf0006838

SDRAM_DFII_CMDINJECTOR_RESULT_ARRAY

Address: 0xf0006800 + 0x3c = 0xf000683c

SDRAM_DFII_CMDINJECTOR_RESET

Address: 0xf0006800 + 0x40 = 0xf0006840

SDRAM_DFII_CMDINJECTOR_RDDATA_SELECT

Address: 0xf0006800 + 0x44 = 0xf0006844

SDRAM_DFII_CMDINJECTOR_RDDATA_CAPTURE_CNT

Address: 0xf0006800 + 0x48 = 0xf0006848

SDRAM_DFII_CMDINJECTOR_RDDATA

Address: 0xf0006800 + 0x4c = 0xf000684c

SDRAM_CONTROLLER_TRP

Address: 0xf0006800 + 0x50 = 0xf0006850

SDRAM_CONTROLLER_TRCD

Address: 0xf0006800 + 0x54 = 0xf0006854

SDRAM_CONTROLLER_TWR

Address: 0xf0006800 + 0x58 = 0xf0006858

SDRAM_CONTROLLER_TWTR

Address: 0xf0006800 + 0x5c = 0xf000685c

SDRAM_CONTROLLER_TREFI

Address: 0xf0006800 + 0x60 = 0xf0006860

SDRAM_CONTROLLER_TRFC

Address: 0xf0006800 + 0x64 = 0xf0006864

SDRAM_CONTROLLER_TFAW

Address: 0xf0006800 + 0x68 = 0xf0006868

SDRAM_CONTROLLER_TCCD

Address: 0xf0006800 + 0x6c = 0xf000686c

SDRAM_CONTROLLER_TCCD_WR

Address: 0xf0006800 + 0x70 = 0xf0006870

SDRAM_CONTROLLER_TRTP

Address: $0xf0006800 + 0x74 = 0xf0006874$

SDRAM_CONTROLLER_TRRD

Address: $0xf0006800 + 0x78 = 0xf0006878$

SDRAM_CONTROLLER_TRC

Address: $0xf0006800 + 0x7c = 0xf000687c$

SDRAM_CONTROLLER_TRAS

Address: $0xf0006800 + 0x80 = 0xf0006880$

SDRAM_CONTROLLER_LAST_ADDR_0

Address: $0xf0006800 + 0x84 = 0xf0006884$

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_0

Address: $0xf0006800 + 0x88 = 0xf0006888$

SDRAM_CONTROLLER_LAST_ADDR_1

Address: $0xf0006800 + 0x8c = 0xf000688c$

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_1

Address: 0xf0006800 + 0x90 = 0xf0006890

SDRAM_CONTROLLER_LAST_ADDR_2

Address: 0xf0006800 + 0x94 = 0xf0006894

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_2

Address: 0xf0006800 + 0x98 = 0xf0006898

SDRAM_CONTROLLER_LAST_ADDR_3

Address: 0xf0006800 + 0x9c = 0xf000689c

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_3

Address: 0xf0006800 + 0xa0 = 0xf00068a0

SDRAM_CONTROLLER_LAST_ADDR_4

Address: 0xf0006800 + 0xa4 = 0xf00068a4

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_4

Address: 0xf0006800 + 0xa8 = 0xf00068a8

SDRAM_CONTROLLER_LAST_ADDR_5

Address: 0xf0006800 + 0xac = 0xf00068ac

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_5

Address: 0xf0006800 + 0xb0 = 0xf00068b0

SDRAM_CONTROLLER_LAST_ADDR_6

Address: 0xf0006800 + 0xb4 = 0xf00068b4

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_6

Address: 0xf0006800 + 0xb8 = 0xf00068b8

SDRAM_CONTROLLER_LAST_ADDR_7

Address: 0xf0006800 + 0xbc = 0xf00068bc

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_7

Address: 0xf0006800 + 0xc0 = 0xf00068c0

SDRAM_CONTROLLER_LAST_ADDR_8

Address: 0xf0006800 + 0xc4 = 0xf00068c4

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_8

Address: 0xf0006800 + 0xc8 = 0xf00068c8

SDRAM_CONTROLLER_LAST_ADDR_9

Address: 0xf0006800 + 0xcc = 0xf00068cc

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_9

Address: 0xf0006800 + 0xd0 = 0xf00068d0

SDRAM_CONTROLLER_LAST_ADDR_10

Address: 0xf0006800 + 0xd4 = 0xf00068d4

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_10

Address: 0xf0006800 + 0xd8 = 0xf00068d8

SDRAM_CONTROLLER_LAST_ADDR_11

Address: 0xf0006800 + 0xdc = 0xf00068dc

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_11

Address: 0xf0006800 + 0xe0 = 0xf00068e0

SDRAM_CONTROLLER_LAST_ADDR_12

Address: 0xf0006800 + 0xe4 = 0xf00068e4

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_12

Address: 0xf0006800 + 0xe8 = 0xf00068e8

SDRAM_CONTROLLER_LAST_ADDR_13

Address: 0xf0006800 + 0xec = 0xf00068ec

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_13

Address: 0xf0006800 + 0xf0 = 0xf00068f0

SDRAM_CONTROLLER_LAST_ADDR_14

Address: 0xf0006800 + 0xf4 = 0xf00068f4

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_14

Address: 0xf0006800 + 0xf8 = 0xf00068f8

SDRAM_CONTROLLER_LAST_ADDR_15

Address: 0xf0006800 + 0xfc = 0xf00068fc

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_15

Address: 0xf0006800 + 0x100 = 0xf0006900

SDRAM_CONTROLLER_LAST_ADDR_16

Address: 0xf0006800 + 0x104 = 0xf0006904

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_16

Address: 0xf0006800 + 0x108 = 0xf0006908

SDRAM_CONTROLLER_LAST_ADDR_17

Address: 0xf0006800 + 0x10c = 0xf000690c

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_17

Address: 0xf0006800 + 0x110 = 0xf0006910

SDRAM_CONTROLLER_LAST_ADDR_18

Address: 0xf0006800 + 0x114 = 0xf0006914

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_18

Address: 0xf0006800 + 0x118 = 0xf0006918

SDRAM_CONTROLLER_LAST_ADDR_19

Address: 0xf0006800 + 0x11c = 0xf000691c

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_19

Address: 0xf0006800 + 0x120 = 0xf0006920

SDRAM_CONTROLLER_LAST_ADDR_20

Address: 0xf0006800 + 0x124 = 0xf0006924

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_20

Address: 0xf0006800 + 0x128 = 0xf0006928

SDRAM_CONTROLLER_LAST_ADDR_21

Address: 0xf0006800 + 0x12c = 0xf000692c

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_21

Address: 0xf0006800 + 0x130 = 0xf0006930

SDRAM_CONTROLLER_LAST_ADDR_22

Address: 0xf0006800 + 0x134 = 0xf0006934

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_22

Address: 0xf0006800 + 0x138 = 0xf0006938

SDRAM_CONTROLLER_LAST_ADDR_23

Address: 0xf0006800 + 0x13c = 0xf000693c

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_23

Address: 0xf0006800 + 0x140 = 0xf0006940

SDRAM_CONTROLLER_LAST_ADDR_24

Address: 0xf0006800 + 0x144 = 0xf0006944

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_24

Address: 0xf0006800 + 0x148 = 0xf0006948

SDRAM_CONTROLLER_LAST_ADDR_25

Address: 0xf0006800 + 0x14c = 0xf000694c

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_25

Address: 0xf0006800 + 0x150 = 0xf0006950

SDRAM_CONTROLLER_LAST_ADDR_26

Address: 0xf0006800 + 0x154 = 0xf0006954

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_26

Address: 0xf0006800 + 0x158 = 0xf0006958

SDRAM_CONTROLLER_LAST_ADDR_27

Address: 0xf0006800 + 0x15c = 0xf000695c

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_27

Address: 0xf0006800 + 0x160 = 0xf0006960

SDRAM_CONTROLLER_LAST_ADDR_28

Address: 0xf0006800 + 0x164 = 0xf0006964

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_28

Address: 0xf0006800 + 0x168 = 0xf0006968

SDRAM_CONTROLLER_LAST_ADDR_29

Address: 0xf0006800 + 0x16c = 0xf000696c

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_29

Address: $0xf0006800 + 0x170 = 0xf0006970$

SDRAM_CONTROLLER_LAST_ADDR_30

Address: $0xf0006800 + 0x174 = 0xf0006974$

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_30

Address: $0xf0006800 + 0x178 = 0xf0006978$

SDRAM_CONTROLLER_LAST_ADDR_31

Address: $0xf0006800 + 0x17c = 0xf000697c$

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_31

Address: $0xf0006800 + 0x180 = 0xf0006980$

16.2.15 SDRAM_CHECKER

Register Listing for SDRAM_CHECKER

Register	Address
<i>SDRAM_CHECKER_RESET</i>	<i>0xf0007000</i>
<i>SDRAM_CHECKER_START</i>	<i>0xf0007004</i>
<i>SDRAM_CHECKER_DONE</i>	<i>0xf0007008</i>
<i>SDRAM_CHECKER_BASE</i>	<i>0xf000700c</i>
<i>SDRAM_CHECKER_END</i>	<i>0xf0007010</i>
<i>SDRAM_CHECKER_LENGTH</i>	<i>0xf0007014</i>
<i>SDRAM_CHECKER_RANDOM</i>	<i>0xf0007018</i>
<i>SDRAM_CHECKER_TICKS</i>	<i>0xf000701c</i>
<i>SDRAM_CHECKER_ERRORS</i>	<i>0xf0007020</i>

SDRAM_CHECKER_RESET

Address: $0xf0007000 + 0x0 = 0xf0007000$

SDRAM_CHECKER_START

Address: $0xf0007000 + 0x4 = 0xf0007004$

SDRAM_CHECKER_DONE

Address: $0xf0007000 + 0x8 = 0xf0007008$

SDRAM_CHECKER_BASE

Address: $0xf0007000 + 0xc = 0xf000700c$

SDRAM_CHECKER_END

Address: $0xf0007000 + 0x10 = 0xf0007010$

SDRAM_CHECKER_LENGTH

Address: $0xf0007000 + 0x14 = 0xf0007014$

SDRAM_CHECKER_RANDOM

Address: $0xf0007000 + 0x18 = 0xf0007018$

Field	Name	Description

SDRAM_CHECKER_TICKS

Address: $0xf0007000 + 0x1c = 0xf000701c$

SDRAM_CHECKER_ERRORS

Address: $0xf0007000 + 0x20 = 0xf0007020$

16.2.16 SDRAM_GENERATOR

Register Listing for SDRAM_GENERATOR

Register	Address
<i>SDRAM_GENERATOR_RESET</i>	<i>0xf0007800</i>
<i>SDRAM_GENERATOR_START</i>	<i>0xf0007804</i>
<i>SDRAM_GENERATOR_DONE</i>	<i>0xf0007808</i>
<i>SDRAM_GENERATOR_BASE</i>	<i>0xf000780c</i>
<i>SDRAM_GENERATOR_END</i>	<i>0xf0007810</i>
<i>SDRAM_GENERATOR_LENGTH</i>	<i>0xf0007814</i>
<i>SDRAM_GENERATOR_RANDOM</i>	<i>0xf0007818</i>
<i>SDRAM_GENERATOR_TICKS</i>	<i>0xf000781c</i>

SDRAM_GENERATOR_RESET

Address: $0xf0007800 + 0x0 = 0xf0007800$

SDRAM_GENERATOR_START

Address: $0xf0007800 + 0x4 = 0xf0007804$

SDRAM_GENERATOR_DONE

Address: $0xf0007800 + 0x8 = 0xf0007808$

SDRAM_GENERATOR_BASE

Address: $0xf0007800 + 0xc = 0xf000780c$

SDRAM_GENERATOR_END

Address: $0xf0007800 + 0x10 = 0xf0007810$

SDRAM_GENERATOR_LENGTH

Address: $0xf0007800 + 0x14 = 0xf0007814$

SDRAM_GENERATOR_RANDOM

Address: $0xf0007800 + 0x18 = 0xf0007818$

Field	Name	Description

SDRAM_GENERATOR_TICKS

Address: $0xf0007800 + 0x1c = 0xf000781c$

16.2.17 TIMERO

Timer

Provides a generic Timer core.

The Timer is implemented as a countdown timer that can be used in various modes:

- Polling : Returns current countdown value to software
- One-Shot: Loads itself and stops when value reaches 0
- Periodic: (Re-)Loads itself when value reaches 0

en register allows the user to enable/disable the Timer. When the Timer is enabled, it is automatically loaded with the value of *load* register.

When the Timer reaches 0, it is automatically reloaded with value of *reload* register.

The user can latch the current countdown value by writing to *update_value* register, it will update *value* register with current countdown value.

To use the Timer in One-Shot mode, the user needs to:

- Disable the timer
- Set the load register to the expected duration
- (Re-)Enable the Timer

To use the Timer in Periodic mode, the user needs to:

- Disable the Timer
- Set the load register to 0
- Set the reload register to the expected period
- Enable the Timer

For both modes, the CPU can be advertised by an IRQ that the duration/period has elapsed. (The CPU can also do software polling with *update_value* and *value* to know the elapsed duration)

Register Listing for TIMERO

Register	Address
<i>TIMERO_LOAD</i>	0xf0008000
<i>TIMERO_RELOAD</i>	0xf0008004
<i>TIMERO_EN</i>	0xf0008008
<i>TIMERO_UPDATE_VALUE</i>	0xf000800c
<i>TIMERO_VALUE</i>	0xf0008010
<i>TIMERO_EV_STATUS</i>	0xf0008014
<i>TIMERO_EV_PENDING</i>	0xf0008018
<i>TIMERO_EV_ENABLE</i>	0xf000801c

TIMER0_LOAD

Address: $0xf0008000 + 0x0 = 0xf0008000$

Load value when Timer is (re-)enabled. In One-Shot mode, the value written to this register specifies the Timer's duration in clock cycles.

TIMER0_RELOAD

Address: $0xf0008000 + 0x4 = 0xf0008004$

Reload value when Timer reaches 0. In Periodic mode, the value written to this register specify the Timer's period in clock cycles.

TIMER0_EN

Address: $0xf0008000 + 0x8 = 0xf0008008$

Enable flag of the Timer. Set this flag to 1 to enable/start the Timer. Set to 0 to disable the Timer.

TIMER0_UPDATE_VALUE

Address: $0xf0008000 + 0xc = 0xf000800c$

Update trigger for the current countdown value. A write to this register latches the current countdown value to value register.

TIMER0_VALUE

Address: $0xf0008000 + 0x10 = 0xf0008010$

Latched countdown value. This value is updated by writing to update_value.

TIMER0_EV_STATUS

Address: $0xf0008000 + 0x14 = 0xf0008014$

This register contains the current raw level of the zero event trigger. Writes to this register have no effect.

Field	Name	Description
[0]	ZERO	Level of the zero event

TIMER0_EV_PENDING

Address: $0xf0008000 + 0x18 = 0xf0008018$

When a zero event occurs, the corresponding bit will be set in this register. To clear the Event, set the corresponding bit in this register.

Field	Name	Description
[0]	ZERO	1 if a zero event occurred. This Event is triggered on a falling edge.

TIMER0_EV_ENABLE

Address: $0xf0008000 + 0x1c = 0xf000801c$

This register enables the corresponding zero events. Write a 0 to this register to disable individual events.

Field	Name	Description
[0]	ZERO	Write a 1 to enable the zero Event

16.2.18 UART

Register Listing for UART

Register	Address
UART_RXTX	0xf0008800
UART_TXFULL	0xf0008804
UART_RXEMPTY	0xf0008808
UART_EV_STATUS	0xf000880c
UART_EV_PENDING	0xf0008810
UART_EV_ENABLE	0xf0008814
UART_TXEMPTY	0xf0008818
UART_RXFULL	0xf000881c
UART_XOVER_RXTX	0xf0008820
UART_XOVER_TXFULL	0xf0008824
UART_XOVER_RXEMPTY	0xf0008828
UART_XOVER_EV_STATUS	0xf000882c
UART_XOVER_EV_PENDING	0xf0008830
UART_XOVER_EV_ENABLE	0xf0008834
UART_XOVER_TXEMPTY	0xf0008838
UART_XOVER_RXFULL	0xf000883c

UART_RXTX

Address: $0xf0008800 + 0x0 = 0xf0008800$

UART_TXFULL

Address: $0xf0008800 + 0x4 = 0xf0008804$

TX FIFO Full.

UART_RXEMPTY

Address: $0xf0008800 + 0x8 = 0xf0008808$

RX FIFO Empty.

UART_EV_STATUS

Address: $0xf0008800 + 0xc = 0xf000880c$

This register contains the current raw level of the rx event trigger. Writes to this register have no effect.

Field	Name	Description
[0]	TX	Level of the tx event
[1]	RX	Level of the rx event

UART_EV_PENDING

Address: $0xf0008800 + 0x10 = 0xf0008810$

When a rx event occurs, the corresponding bit will be set in this register. To clear the Event, set the corresponding bit in this register.

Field	Name	Description
[0]	TX	1 if a tx event occurred. This Event is triggered on a falling edge.
[1]	RX	1 if a rx event occurred. This Event is triggered on a falling edge.

UART_EV_ENABLE

Address: $0xf0008800 + 0x14 = 0xf0008814$

This register enables the corresponding rx events. Write a 0 to this register to disable individual events.

Field	Name	Description
[0]	TX	Write a 1 to enable the tx Event
[1]	RX	Write a 1 to enable the rx Event

UART_TXEMPTY

Address: $0xf0008800 + 0x18 = 0xf0008818$

TX FIFO Empty.

UART_RXFULL

Address: $0xf0008800 + 0x1c = 0xf000881c$

RX FIFO Full.

UART_XOVER_RXTX

Address: $0xf0008800 + 0x20 = 0xf0008820$

UART_XOVER_TXFULL

Address: $0xf0008800 + 0x24 = 0xf0008824$

TX FIFO Full.

UART_XOVER_RXEMPTY

Address: $0xf0008800 + 0x28 = 0xf0008828$

RX FIFO Empty.

UART_XOVER_EV_STATUS

Address: $0xf0008800 + 0x2c = 0xf000882c$

This register contains the current raw level of the rx event trigger. Writes to this register have no effect.

Field	Name	Description
[0]	TX	Level of the tx event
[1]	RX	Level of the rx event

UART_XOVER_EV_PENDING

Address: $0xf0008800 + 0x30 = 0xf0008830$

When a rx event occurs, the corresponding bit will be set in this register. To clear the Event, set the corresponding bit in this register.

Field	Name	Description
[0]	TX	1 if a tx event occurred. This Event is triggered on a falling edge.
[1]	RX	1 if a rx event occurred. This Event is triggered on a falling edge.

UART_XOVER_EV_ENABLE

Address: $0xf0008800 + 0x34 = 0xf0008834$

This register enables the corresponding rx events. Write a 0 to this register to disable individual events.

Field	Name	Description
[0]	TX	Write a 1 to enable the tx Event
[1]	RX	Write a 1 to enable the rx Event

UART_XOVER_TXEMPTY

Address: $0xf0008800 + 0x38 = 0xf0008838$

TX FIFO Empty.

UART_XOVER_RXFULL

Address: 0xf0008800 + 0x3c = 0xf000883c

RX FIFO Full.

CHAPTER
SEVENTEEN

DOCUMENTATION FOR ROW HAMMER TESTER DDR5 TESTER

17.1 Modules

17.1.1 Interrupt Controller

This device has an EventManager-based interrupt system. Individual modules generate *events* which are wired into a central interrupt controller.

When an interrupt occurs, you should look the interrupt number up in the CPU- specific interrupt table and then call the relevant module.

Assigned Interrupts

The following interrupts are assigned on this system:

Interrupt	Module
1	<i>TIMER0</i>
0	<i>UART</i>

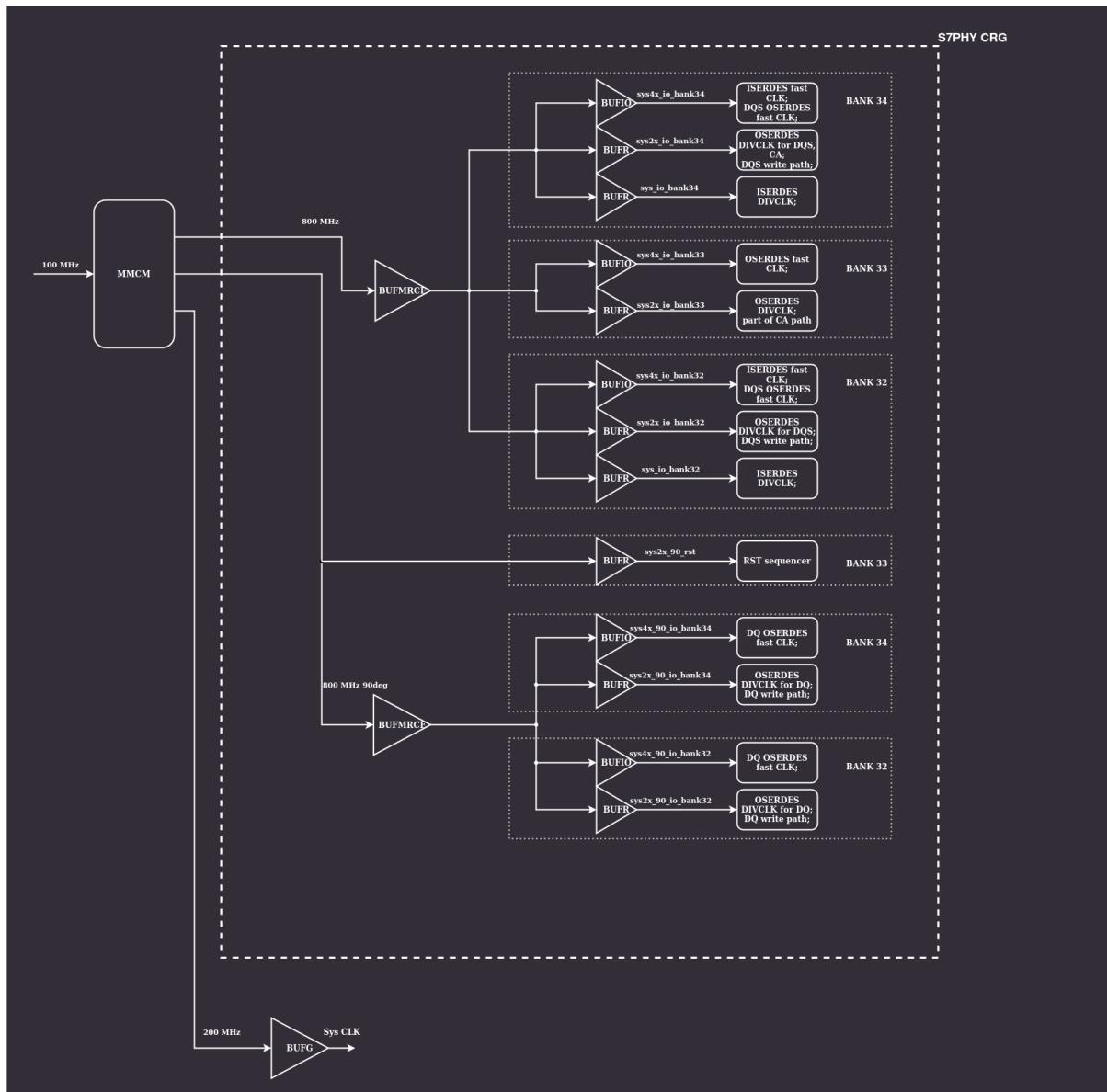
17.1.2 PHYCRG

S7CRGPHY

This module contains 7 series specific clock and reset generation for S7DDR5 PHY. It adds:

- BUFMRCE to control multi region PHYs (UDIMMs and/or RDIMMs),
- BUFMRCE/BUFRs reset sequence,
- ISERDES reset sequence correct with Xilinx documentation and design advisories,
- OSERDES reset sequence.

DDR5 Tester Clock tree



17.2 Register Groups

17.2.1 LEDs

Register Listing for LEDs

Register	Address
<i>LEDS_OUT</i>	<i>0xf0000000</i>

LEDS_OUT

Address: $0xf0000000 + 0x0 = 0xf0000000$

Led Output(s) Control.

17.2.2 DDRPHY

Register Listing for DDRPHY

Register	Address
<i>DDRPHY_CSRMODULE_ENABLE_FIFOS</i>	0xf0000800
<i>DDRPHY_CSRMODULE_RST</i>	0xf0000804
<i>DDRPHY_CSRMODULE_RDIMM_MODE</i>	0xf0000808
<i>DDRPHY_CSRMODULE_RDPHASE</i>	0xf000080c
<i>DDRPHY_CSRMODULE_WRPHASE</i>	0xf0000810
<i>DDRPHY_CSRMODULE_ALERT</i>	0xf0000814
<i>DDRPHY_CSRMODULE_ALERT_REDUCE</i>	0xf0000818
<i>DDRPHY_CSRMODULE_SAMPLE_ALERT</i>	0xf000081c
<i>DDRPHY_CSRMODULE_RESET_ALERT</i>	0xf0000820
<i>DDRPHY_CSRMODULE_CKDLY_RST</i>	0xf0000824
<i>DDRPHY_CSRMODULE_CKDLY_INC</i>	0xf0000828
<i>DDRPHY_CSRMODULE_A_PREAMBLE</i>	0xf000082c
<i>DDRPHY_CSRMODULE_A_WLEVEL_EN</i>	0xf0000830
<i>DDRPHY_CSRMODULE_A_PAR_ENABLE</i>	0xf0000834
<i>DDRPHY_CSRMODULE_A_PAR_VALUE</i>	0xf0000838
<i>DDRPHY_CSRMODULE_A_DISCARD_RD_FIFO</i>	0xf000083c
<i>DDRPHY_CSRMODULE_A_DLY_SEL</i>	0xf0000840
<i>DDRPHY_CSRMODULE_A_DQ_DQS_RATIO</i>	0xf0000844
<i>DDRPHY_CSRMODULE_A_CK_RDLY_INC</i>	0xf0000848
<i>DDRPHY_CSRMODULE_A_CK_RDLY_RST</i>	0xf000084c
<i>DDRPHY_CSRMODULE_A_CK_RDDLY</i>	0xf0000850
<i>DDRPHY_CSRMODULE_A_CK_RDDLY_PREAMBLE</i>	0xf0000854
<i>DDRPHY_CSRMODULE_A_CK_WDLY_INC</i>	0xf0000858
<i>DDRPHY_CSRMODULE_A_CK_WDLY_RST</i>	0xf000085c
<i>DDRPHY_CSRMODULE_A_CK_WDLY_DQS</i>	0xf0000860
<i>DDRPHY_CSRMODULE_A_CK_WDDLY_INC</i>	0xf0000864
<i>DDRPHY_CSRMODULE_A_CK_WDDLY_RST</i>	0xf0000868
<i>DDRPHY_CSRMODULE_A_CK_WDLY_DQ</i>	0xf000086c
<i>DDRPHY_CSRMODULE_A_DQ_DLY_SEL</i>	0xf0000870
<i>DDRPHY_CSRMODULE_A_CSDLY_RST</i>	0xf0000874
<i>DDRPHY_CSRMODULE_A_CSDLY_INC</i>	0xf0000878
<i>DDRPHY_CSRMODULE_A_CADLY_RST</i>	0xf000087c
<i>DDRPHY_CSRMODULE_A_CADLY_INC</i>	0xf0000880
<i>DDRPHY_CSRMODULE_A_PARDLY_RST</i>	0xf0000884
<i>DDRPHY_CSRMODULE_A_PARDLY_INC</i>	0xf0000888
<i>DDRPHY_CSRMODULE_A_CSDLY</i>	0xf000088c
<i>DDRPHY_CSRMODULE_A_CADLY</i>	0xf0000890

continues on next page

Table 17.1 – continued from previous page

Register	Address
<i>DDRPHY_CSRMODULE_A_RDLY_DQ_RST</i>	0xf0000894
<i>DDRPHY_CSRMODULE_A_RDLY_DQ_INC</i>	0xf0000898
<i>DDRPHY_CSRMODULE_A_RDLY_DQS_RST</i>	0xf000089c
<i>DDRPHY_CSRMODULE_A_RDLY_DQS_INC</i>	0xf00008a0
<i>DDRPHY_CSRMODULE_A_RDLY_DQS</i>	0xf00008a4
<i>DDRPHY_CSRMODULE_A_RDLY_DQ</i>	0xf00008a8
<i>DDRPHY_CSRMODULE_A_WDLY_DQ_RST</i>	0xf00008ac
<i>DDRPHY_CSRMODULE_A_WDLY_DQ_INC</i>	0xf00008b0
<i>DDRPHY_CSRMODULE_A_WDLY_DM_RST</i>	0xf00008b4
<i>DDRPHY_CSRMODULE_A_WDLY_DM_INC</i>	0xf00008b8
<i>DDRPHY_CSRMODULE_A_WDLY_DQS_RST</i>	0xf00008bc
<i>DDRPHY_CSRMODULE_A_WDLY_DQS_INC</i>	0xf00008c0
<i>DDRPHY_CSRMODULE_A_WDLY_DQS</i>	0xf00008c4
<i>DDRPHY_CSRMODULE_A_WDLY_DQ</i>	0xf00008c8
<i>DDRPHY_CSRMODULE_A_WDLY_DM</i>	0xf00008cc
<i>DDRPHY_CSRMODULE_B_PREAMBLE</i>	0xf00008d0
<i>DDRPHY_CSRMODULE_B_WLEVEL_EN</i>	0xf00008d4
<i>DDRPHY_CSRMODULE_B_PAR_ENABLE</i>	0xf00008d8
<i>DDRPHY_CSRMODULE_B_PAR_VALUE</i>	0xf00008dc
<i>DDRPHY_CSRMODULE_B_DISCARD_RD_FIFO</i>	0xf00008e0
<i>DDRPHY_CSRMODULE_B_DLY_SEL</i>	0xf00008e4
<i>DDRPHY_CSRMODULE_B_DQ_DQS_RATIO</i>	0xf00008e8
<i>DDRPHY_CSRMODULE_B_CK_RDLY_INC</i>	0xf00008ec
<i>DDRPHY_CSRMODULE_B_CK_RDLY_RST</i>	0xf00008f0
<i>DDRPHY_CSRMODULE_B_CK_RDDLY</i>	0xf00008f4
<i>DDRPHY_CSRMODULE_B_CK_RDDLY_PREAMBLE</i>	0xf00008f8
<i>DDRPHY_CSRMODULE_B_CK_WDLY_INC</i>	0xf00008fc
<i>DDRPHY_CSRMODULE_B_CK_WDLY_RST</i>	0xf0000900
<i>DDRPHY_CSRMODULE_B_CK_WDLY_DQS</i>	0xf0000904
<i>DDRPHY_CSRMODULE_B_CK_WDDLY_INC</i>	0xf0000908
<i>DDRPHY_CSRMODULE_B_CK_WDDLY_RST</i>	0xf000090c
<i>DDRPHY_CSRMODULE_B_CK_WDLY_DQ</i>	0xf0000910
<i>DDRPHY_CSRMODULE_B_DQ_DLY_SEL</i>	0xf0000914
<i>DDRPHY_CSRMODULE_B_CSDLY_RST</i>	0xf0000918
<i>DDRPHY_CSRMODULE_B_CSDLY_INC</i>	0xf000091c
<i>DDRPHY_CSRMODULE_B_CADLY_RST</i>	0xf0000920
<i>DDRPHY_CSRMODULE_B_CADLY_INC</i>	0xf0000924
<i>DDRPHY_CSRMODULE_B_PARDLY_RST</i>	0xf0000928
<i>DDRPHY_CSRMODULE_B_PARDLY_INC</i>	0xf000092c
<i>DDRPHY_CSRMODULE_B_CSDLY</i>	0xf0000930
<i>DDRPHY_CSRMODULE_B_CADLY</i>	0xf0000934
<i>DDRPHY_CSRMODULE_B_RDLY_DQ_RST</i>	0xf0000938
<i>DDRPHY_CSRMODULE_B_RDLY_DQ_INC</i>	0xf000093c
<i>DDRPHY_CSRMODULE_B_RDLY_DQS_RST</i>	0xf0000940
<i>DDRPHY_CSRMODULE_B_RDLY_DQS_INC</i>	0xf0000944
<i>DDRPHY_CSRMODULE_B_RDLY_DQS</i>	0xf0000948
<i>DDRPHY_CSRMODULE_B_RDLY_DQ</i>	0xf000094c
<i>DDRPHY_CSRMODULE_B_WDLY_DQ_RST</i>	0xf0000950

continues on next page

Table 17.1 – continued from previous page

Register	Address
<i>DDRPHY_CSRMODULE_B_WDLY_DQ_INC</i>	0xf0000954
<i>DDRPHY_CSRMODULE_B_WDLY_DM_RST</i>	0xf0000958
<i>DDRPHY_CSRMODULE_B_WDLY_DM_INC</i>	0xf000095c
<i>DDRPHY_CSRMODULE_B_WDLY_DQS_RST</i>	0xf0000960
<i>DDRPHY_CSRMODULE_B_WDLY_DQS_INC</i>	0xf0000964
<i>DDRPHY_CSRMODULE_B_WDLY_DQS</i>	0xf0000968
<i>DDRPHY_CSRMODULE_B_WDLY_DQ</i>	0xf000096c
<i>DDRPHY_CSRMODULE_B_WDLY_DM</i>	0xf0000970

DDRPHY_CSRMODULE_ENABLE_FIFOS

Address: 0xf0000800 + 0x0 = 0xf0000800

DDRPHY_CSRMODULE_RST

Address: 0xf0000800 + 0x4 = 0xf0000804

DDRPHY_CSRMODULE_RDIMM_MODE

Address: 0xf0000800 + 0x8 = 0xf0000808

DDRPHY_CSRMODULE_RDPHASE

Address: 0xf0000800 + 0xc = 0xf000080c

DDRPHY_CSRMODULE_WRPHASE

Address: 0xf0000800 + 0x10 = 0xf0000810

DDRPHY_CSRMODULE_ALERT

Address: $0xf0000800 + 0x14 = 0xf0000814$

DDRPHY_CSRMODULE_ALERT_REDUCE

Address: $0xf0000800 + 0x18 = 0xf0000818$

Field	Name	Description
[0]	INITIAL_STATE	Initial value of all bits
[1]	OPERATION	0 - or (default), 1 -and

DDRPHY_CSRMODULE_SAMPLE_ALERT

Address: $0xf0000800 + 0x1c = 0xf000081c$

DDRPHY_CSRMODULE_RESET_ALERT

Address: $0xf0000800 + 0x20 = 0xf0000820$

DDRPHY_CSRMODULE_CKDLY_RST

Address: $0xf0000800 + 0x24 = 0xf0000824$

DDRPHY_CSRMODULE_CKDLY_INC

Address: $0xf0000800 + 0x28 = 0xf0000828$

DDRPHY_CSRMODULE_A_PREAMBLE

Address: 0xf0000800 + 0x2c = 0xf000082c

DDRPHY_CSRMODULE_A_WLEVEL_EN

Address: 0xf0000800 + 0x30 = 0xf0000830

DDRPHY_CSRMODULE_A_PAR_ENABLE

Address: 0xf0000800 + 0x34 = 0xf0000834

DDRPHY_CSRMODULE_A_PAR_VALUE

Address: 0xf0000800 + 0x38 = 0xf0000838

DDRPHY_CSRMODULE_A_DISCARD_RD_FIFO

Address: 0xf0000800 + 0x3c = 0xf000083c

DDRPHY_CSRMODULE_A_DLY_SEL

Address: 0xf0000800 + 0x40 = 0xf0000840

DDRPHY_CSRMODULE_A_DQ_DQS_RATIO

Address: 0xf0000800 + 0x44 = 0xf0000844

DDRPHY_CSRMODULE_A_CK_RDLY_INC

Address: 0xf0000800 + 0x48 = 0xf0000848

DDRPHY_CSRMODULE_A_CK_RDLY_RST

Address: 0xf0000800 + 0x4c = 0xf000084c

DDRPHY_CSRMODULE_A_CK_RDDLY

Address: 0xf0000800 + 0x50 = 0xf0000850

DDRPHY_CSRMODULE_A_CK_RDDLY_PREAMBLE

Address: 0xf0000800 + 0x54 = 0xf0000854

DDRPHY_CSRMODULE_A_CK_WDLY_INC

Address: 0xf0000800 + 0x58 = 0xf0000858

DDRPHY_CSRMODULE_A_CK_WDLY_RST

Address: 0xf0000800 + 0x5c = 0xf000085c

DDRPHY_CSRMODULE_A_CK_WDLY_DQS

Address: 0xf0000800 + 0x60 = 0xf0000860

DDRPHY_CSRMODULE_A_CK_WDDLY_INC

Address: 0xf0000800 + 0x64 = 0xf0000864

DDRPHY_CSRMODULE_A_CK_WDDLY_RST

Address: 0xf0000800 + 0x68 = 0xf0000868

DDRPHY_CSRMODULE_A_CK_WDLY_DQ

Address: 0xf0000800 + 0x6c = 0xf000086c

DDRPHY_CSRMODULE_A_DQ_DLY_SEL

Address: 0xf0000800 + 0x70 = 0xf0000870

DDRPHY_CSRMODULE_A_CSDLY_RST

Address: 0xf0000800 + 0x74 = 0xf0000874

DDRPHY_CSRMODULE_A_CSDLY_INC

Address: 0xf0000800 + 0x78 = 0xf0000878

DDRPHY_CSRMODULE_A_CADLY_RST

Address: 0xf0000800 + 0x7c = 0xf000087c

DDRPHY_CSRMODULE_A_CADLY_INC

Address: 0xf0000800 + 0x80 = 0xf0000880

DDRPHY_CSRMODULE_A_PARDLY_RST

Address: 0xf0000800 + 0x84 = 0xf0000884

DDRPHY_CSRMODULE_A_PARDLY_INC

Address: 0xf0000800 + 0x88 = 0xf0000888

DDRPHY_CSRMODULE_A_CSDLY

Address: 0xf0000800 + 0x8c = 0xf000088c

DDRPHY_CSRMODULE_A_CADLY

Address: 0xf0000800 + 0x90 = 0xf0000890

DDRPHY_CSRMODULE_A_RDLY_DQ_RST

Address: 0xf0000800 + 0x94 = 0xf0000894

DDRPHY_CSRMODULE_A_RDLY_DQ_INC

Address: 0xf0000800 + 0x98 = 0xf0000898

DDRPHY_CSRMODULE_A_RDLY_DQS_RST

Address: 0xf0000800 + 0x9c = 0xf000089c

DDRPHY_CSRMODULE_A_RDLY_DQS_INC

Address: 0xf0000800 + 0xa0 = 0xf00008a0

DDRPHY_CSRMODULE_A_RDLY_DQS

Address: 0xf0000800 + 0xa4 = 0xf00008a4

DDRPHY_CSRMODULE_A_RDLY_DQ

Address: 0xf0000800 + 0xa8 = 0xf00008a8

DDRPHY_CSRMODULE_A_WDLY_DQ_RST

Address: 0xf0000800 + 0xac = 0xf00008ac

DDRPHY_CSRMODULE_A_WDLY_DQ_INC

Address: 0xf0000800 + 0xb0 = 0xf00008b0

DDRPHY_CSRMODULE_A_WDLY_DM_RST

Address: 0xf0000800 + 0xb4 = 0xf00008b4

DDRPHY_CSRMODULE_A_WDLY_DM_INC

Address: 0xf0000800 + 0xb8 = 0xf00008b8

DDRPHY_CSRMODULE_A_WDLY_DQS_RST

Address: 0xf0000800 + 0xbc = 0xf00008bc

DDRPHY_CSRMODULE_A_WDLY_DQS_INC

Address: 0xf0000800 + 0xc0 = 0xf00008c0

DDRPHY_CSRMODULE_A_WDLY_DQS

Address: 0xf0000800 + 0xc4 = 0xf00008c4

DDRPHY_CSRMODULE_A_WDLY_DQ

Address: 0xf0000800 + 0xc8 = 0xf00008c8

DDRPHY_CSRMODULE_A_WDLY_DM

Address: 0xf0000800 + 0xcc = 0xf00008cc

DDRPHY_CSRMODULE_B_PREAMBLE

Address: 0xf0000800 + 0xd0 = 0xf00008d0

DDRPHY_CSRMODULE_B_WLEVEL_EN

Address: 0xf0000800 + 0xd4 = 0xf00008d4

DDRPHY_CSRMODULE_B_PAR_ENABLE

Address: 0xf0000800 + 0xd8 = 0xf00008d8

DDRPHY_CSRMODULE_B_PAR_VALUE

Address: 0xf0000800 + 0xdc = 0xf00008dc

DDRPHY_CSRMODULE_B_DISCARD_RD_FIFO

Address: 0xf0000800 + 0xe0 = 0xf00008e0

DDRPHY_CSRMODULE_B_DLY_SEL

Address: 0xf0000800 + 0xe4 = 0xf00008e4

DDRPHY_CSRMODULE_B_DQ_DQS_RATIO

Address: 0xf0000800 + 0xe8 = 0xf00008e8

DDRPHY_CSRMODULE_B_CK_RDLY_INC

Address: 0xf0000800 + 0xec = 0xf00008ec

DDRPHY_CSRMODULE_B_CK_RDLY_RST

Address: 0xf0000800 + 0xf0 = 0xf00008f0

DDRPHY_CSRMODULE_B_CK_RDDLY

Address: 0xf0000800 + 0xf4 = 0xf00008f4

DDRPHY_CSRMODULE_B_CK_RDDLY_PREAMBLE

Address: 0xf0000800 + 0xf8 = 0xf00008f8

DDRPHY_CSRMODULE_B_CK_WDLY_INC

Address: 0xf0000800 + 0xfc = 0xf00008fc

DDRPHY_CSRMODULE_B_CK_WDLY_RST

Address: 0xf0000800 + 0x100 = 0xf0000900

DDRPHY_CSRMODULE_B_CK_WDLY_DQS

Address: 0xf0000800 + 0x104 = 0xf0000904

DDRPHY_CSRMODULE_B_CK_WDDLY_INC

Address: 0xf0000800 + 0x108 = 0xf0000908

DDRPHY_CSRMODULE_B_CK_WDDLY_RST

Address: 0xf0000800 + 0x10c = 0xf000090c

DDRPHY_CSRMODULE_B_CK_WDLY_DQ

Address: 0xf0000800 + 0x110 = 0xf0000910

DDRPHY_CSRMODULE_B_DQ_DLY_SEL

Address: 0xf0000800 + 0x114 = 0xf0000914

DDRPHY_CSRMODULE_B_CSDLY_RST

Address: 0xf0000800 + 0x118 = 0xf0000918

DDRPHY_CSRMODULE_B_CSDLY_INC

Address: 0xf0000800 + 0x11c = 0xf000091c

DDRPHY_CSRMODULE_B_CADLY_RST

Address: 0xf0000800 + 0x120 = 0xf0000920

DDRPHY_CSRMODULE_B_CADLY_INC

Address: 0xf0000800 + 0x124 = 0xf0000924

DDRPHY_CSRMODULE_B_PARDLY_RST

Address: 0xf0000800 + 0x128 = 0xf0000928

DDRPHY_CSRMODULE_B_PARDLY_INC

Address: 0xf0000800 + 0x12c = 0xf000092c

DDRPHY_CSRMODULE_B_CSDLY

Address: 0xf0000800 + 0x130 = 0xf0000930

DDRPHY_CSRMODULE_B_CADLY

Address: 0xf0000800 + 0x134 = 0xf0000934

DDRPHY_CSRMODULE_B_RDLY_DQ_RST

Address: 0xf0000800 + 0x138 = 0xf0000938

DDRPHY_CSRMODULE_B_RDLY_DQ_INC

Address: 0xf0000800 + 0x13c = 0xf000093c

DDRPHY_CSRMODULE_B_RDLY_DQS_RST

Address: 0xf0000800 + 0x140 = 0xf0000940

DDRPHY_CSRMODULE_B_RDLY_DQS_INC

Address: 0xf0000800 + 0x144 = 0xf0000944

DDRPHY_CSRMODULE_B_RDLY_DQS

Address: 0xf0000800 + 0x148 = 0xf0000948

DDRPHY_CSRMODULE_B_RDLY_DQ

Address: 0xf0000800 + 0x14c = 0xf000094c

DDRPHY_CSRMODULE_B_WDLY_DQ_RST

Address: 0xf0000800 + 0x150 = 0xf0000950

DDRPHY_CSRMODULE_B_WDLY_DQ_INC

Address: 0xf0000800 + 0x154 = 0xf0000954

DDRPHY_CSRMODULE_B_WDLY_DM_RST

Address: 0xf0000800 + 0x158 = 0xf0000958

DDRPHY_CSRMODULE_B_WDLY_DM_INC

Address: 0xf0000800 + 0x15c = 0xf000095c

DDRPHY_CSRMODULE_B_WDLY_DQS_RST

Address: 0xf0000800 + 0x160 = 0xf0000960

DDRPHY_CSRMODULE_B_WDLY_DQS_INC

Address: 0xf0000800 + 0x164 = 0xf0000964

DDRPHY_CSRMODULE_B_WDLY_DQS

Address: 0xf0000800 + 0x168 = 0xf0000968

DDRPHY_CSRMODULE_B_WDLY_DQ

Address: 0xf0000800 + 0x16c = 0xf000096c

DDRPHY_CSRMODULE_B_WDLY_DM

Address: 0xf0000800 + 0x170 = 0xf0000970

17.2.3 CONTROLLER_SETTINGS

Allows to change LiteDRAMController behaviour at runtime

Register Listing for CONTROLLER_SETTINGS

Register	Address
<i>CONTROLLER_SETTINGS_REFRESH</i>	<i>0xf0001000</i>

CONTROLLER_SETTINGS_REFRESH

Address: 0xf0001000 + 0x0 = 0xf0001000

Enable/disable Refresh commands sending

17.2.4 DDRCTRL

Register Listing for DDRCTRL

Register	Address
<i>DDRCTRL_INIT_DONE</i>	<i>0xf0001800</i>
<i>DDRCTRL_INIT_ERROR</i>	<i>0xf0001804</i>

DDRCTRL_INIT_DONE

Address: 0xf0001800 + 0x0 = 0xf0001800

DDRCTRL_INIT_ERROR

Address: 0xf0001800 + 0x4 = 0xf0001804

17.2.5 ROWHAMMER

Row Hammer DMA attacker

This module allows to perform a Row Hammer attack by configuring it with two addresses that map to different rows of a single bank. When enabled, it will perform alternating DMA reads from the given locations, which will result in the DRAM controller having to repeatedly open/close rows at each read access.

Register Listing for ROWHAMMER

Register	Address
<i>ROWHAMMER_ENABLED</i>	<i>0xf0002000</i>
<i>ROWHAMMER_ADDRESS1</i>	<i>0xf0002004</i>
<i>ROWHAMMER_ADDRESS2</i>	<i>0xf0002008</i>
<i>ROWHAMMER_COUNT</i>	<i>0xf000200c</i>

ROWHAMMER_ENABLED

Address: $0xf0002000 + 0x0 = 0xf0002000$

Used to start/stop the operation of the module

ROWHAMMER_ADDRESS1

Address: $0xf0002000 + 0x4 = 0xf0002004$

First attacked address

ROWHAMMER_ADDRESS2

Address: $0xf0002000 + 0x8 = 0xf0002008$

Second attacked address

ROWHAMMER_COUNT

Address: $0xf0002000 + 0xc = 0xf000200c$

This is the number of DMA accesses performed. When the module is enabled, the value can be freely read. When the module is disabled, the register is clear-on-write and has to be read before the next attack.

17.2.6 WRITER

DMA DRAM writer.

Allows to fill DRAM with a predefined pattern using DMA.

Pattern

Provides access to RAM to store access pattern: *mem_addr* and *mem_data*. The pattern address space can be limited using the *data_mask*.

For example, having *mem_addr* filled with $[0x04, 0x02, 0x03, \dots]$ and *mem_data* filled with $[0xff, 0xaa, 0x55, \dots]$ and setting *data_mask* = *0b01*, the pattern $[(address, data), \dots]$ written will be: $[(0x04, 0xff), (0x02, 0xaa), (0x04, 0xff), \dots]$ (wraps due to masking).

DRAM memory range that is being accessed can be configured using *mem_mask*.

To use this module, make sure that *ready* is 1, then write the desired number of transfers to *count*. Writing to the *start* CSR will initialize the operation. When the operation is ongoing *ready* will be 0.

Register Listing for WRITER

Register	Address
WRITER_START	0xf0002800
WRITER_READY	0xf0002804
WRITER_MODULO	0xf0002808
WRITER_COUNT	0xf000280c
WRITER_DONE	0xf0002810
WRITER_MEM_MASK	0xf0002814
WRITER_DATA_MASK	0xf0002818
WRITER_DATA_DIV	0xf000281c
WRITER_INVERTER_DIVISOR_MASK	0xf0002820
WRITER_INVERTER_SELECTION_MASK	0xf0002824
WRITER_LAST_ADDRESS	0xf0002828

WRITER_START

Address: $0xf0002800 + 0x0 = 0xf0002800$

Write to the register starts the transfer (if ready=1)

WRITER_READY

Address: $0xf0002800 + 0x4 = 0xf0002804$

Indicates that the transfer is not ongoing

WRITER_MODULO

Address: $0xf0002800 + 0x8 = 0xf0002808$

When set use modulo to calculate DMA transfers address rather than bit masking

WRITER_COUNT

Address: $0xf0002800 + 0xc = 0xf000280c$

Desired number of DMA transfers

WRITER_DONE

Address: $0xf0002800 + 0x10 = 0xf0002810$

Number of completed DMA transfers

WRITER_MEM_MASK

Address: $0xf0002800 + 0x14 = 0xf0002814$

DRAM address mask for DMA transfers

WRITER_DATA_MASK

Address: $0xf0002800 + 0x18 = 0xf0002818$

Pattern memory address mask

WRITER_DATA_DIV

Address: $0xf0002800 + 0x1c = 0xf000281c$

Pattern memory address divisor-1

WRITER_INVERTER_DIVISOR_MASK

Address: $0xf0002800 + 0x20 = 0xf0002820$

Divisor mask for selecting rows for which pattern data gets inverted

WRITER_INVERTER_SELECTION_MASK

Address: $0xf0002800 + 0x24 = 0xf0002824$

Selection mask for selecting rows for which pattern data gets inverted

WRITER_LAST_ADDRESS

Address: $0xf0002800 + 0x28 = 0xf0002828$

Number of completed DMA transfers

17.2.7 READER

DMA DRAM reader.

Allows to check DRAM contents against a predefined pattern using DMA.

Pattern

Provides access to RAM to store access pattern: *mem_addr* and *mem_data*. The pattern address space can be limited using the *data_mask*.

For example, having *mem_addr* filled with [0x04, 0x02, 0x03, ...] and *mem_data* filled with [0xff, 0xaa, 0x55, ...] and setting *data_mask* = 0b01, the pattern [(address, data), ...] written will be: [(0x04, 0xff), (0x02, 0xaa), (0x04, 0xff), ...] (wraps due to masking).

DRAM memory range that is being accessed can be configured using *mem_mask*.

To use this module, make sure that *ready* is 1, then write the desired number of transfers to *count*. Writing to the *start* CSR will initialize the operation. When the operation is ongoing *ready* will be 0.

Reading errors

This module allows to check the locations of errors in the memory. It scans the configured memory area and compares the values read to the predefined pattern. If *skip_fifo* is 0, this module will stop after each error encountered, so that it can be examined. Wait until the *error_ready* CSR is 1. Then use the CSRs *error_offset*, *error_data* and *error_expected* to examine the errors in the current transfer. To continue reading, write 1 to *error_continue* CSR. Setting *skip_fifo* to 1 will disable this behaviour entirely.

The final number of errors can be read from *error_count*. NOTE: This value represents the number of erroneous DMA transfers.

The current progress can be read from the *done* CSR.

Register Listing for READER

Register	Address
<i>READER_START</i>	0xf0003000
<i>READER_READY</i>	0xf0003004
<i>READER_MODULO</i>	0xf0003008
<i>READER_COUNT</i>	0xf000300c
<i>READER_DONE</i>	0xf0003010
<i>READER_MEM_MASK</i>	0xf0003014
<i>READER_DATA_MASK</i>	0xf0003018
<i>READER_DATA_DIV</i>	0xf000301c
<i>READER_INVERTER_DIVISOR_MASK</i>	0xf0003020
<i>READER_INVERTER_SELECTION_MASK</i>	0xf0003024
<i>READER_ERROR_COUNT</i>	0xf0003028

continues on next page

Table 17.2 – continued from previous page

Register	Address
<i>READER_SKIP_FIFO</i>	0xf000302c
<i>READER_ERROR_OFFSET</i>	0xf0003030
<i>READER_ERROR_DATA7</i>	0xf0003034
<i>READER_ERROR_DATA6</i>	0xf0003038
<i>READER_ERROR_DATA5</i>	0xf000303c
<i>READER_ERROR_DATA4</i>	0xf0003040
<i>READER_ERROR_DATA3</i>	0xf0003044
<i>READER_ERROR_DATA2</i>	0xf0003048
<i>READER_ERROR_DATA1</i>	0xf000304c
<i>READER_ERROR_DATA0</i>	0xf0003050
<i>READER_ERROR_EXPECTED7</i>	0xf0003054
<i>READER_ERROR_EXPECTED6</i>	0xf0003058
<i>READER_ERROR_EXPECTED5</i>	0xf000305c
<i>READER_ERROR_EXPECTED4</i>	0xf0003060
<i>READER_ERROR_EXPECTED3</i>	0xf0003064
<i>READER_ERROR_EXPECTED2</i>	0xf0003068
<i>READER_ERROR_EXPECTED1</i>	0xf000306c
<i>READER_ERROR_EXPECTED0</i>	0xf0003070
<i>READER_ERROR_READY</i>	0xf0003074
<i>READER_ERROR_CONTINUE</i>	0xf0003078

READER_START

Address: 0xf0003000 + 0x0 = 0xf0003000

Write to the register starts the transfer (if ready=1)

READER_READY

Address: 0xf0003000 + 0x4 = 0xf0003004

Indicates that the transfer is not ongoing

READER_MODULO

Address: 0xf0003000 + 0x8 = 0xf0003008

When set use modulo to calculate DMA transfers address rather than bit masking

READER_COUNT

Address: $0xf0003000 + 0xc = 0xf000300c$

Desired number of DMA transfers

READER_DONE

Address: $0xf0003000 + 0x10 = 0xf0003010$

Number of completed DMA transfers

READER_MEM_MASK

Address: $0xf0003000 + 0x14 = 0xf0003014$

DRAM address mask for DMA transfers

READER_DATA_MASK

Address: $0xf0003000 + 0x18 = 0xf0003018$

Pattern memory address mask

READER_DATA_DIV

Address: $0xf0003000 + 0x1c = 0xf000301c$

Pattern memory address divisor-1

READER_INVERTER_DIVISOR_MASK

Address: $0xf0003000 + 0x20 = 0xf0003020$

Divisor mask for selecting rows for which pattern data gets inverted

READER_INVERTER_SELECTION_MASK

Address: $0xf0003000 + 0x24 = 0xf0003024$

Selection mask for selecting rows for which pattern data gets inverted

READER_ERROR_COUNT

Address: $0xf0003000 + 0x28 = 0xf0003028$

Number of errors detected

READER_SKIP_FIFO

Address: $0xf0003000 + 0x2c = 0xf000302c$

Skip waiting for user to read the errors FIFO

READER_ERROR_OFFSET

Address: $0xf0003000 + 0x30 = 0xf0003030$

Current offset of the error

READER_ERROR_DATA7

Address: $0xf0003000 + 0x34 = 0xf0003034$

Bits 224-255 of *READER_ERROR_DATA*. Erroneous value read from DRAM memory

READER_ERROR_DATA6

Address: $0xf0003000 + 0x38 = 0xf0003038$

Bits 192-223 of *READER_ERROR_DATA*.

READER_ERROR_DATA5

Address: $0xf0003000 + 0x3c = 0xf000303c$

Bits 160-191 of *READER_ERROR_DATA*.

READER_ERROR_DATA4

Address: $0xf0003000 + 0x40 = 0xf0003040$

Bits 128-159 of *READER_ERROR_DATA*.

READER_ERROR_DATA3

Address: $0xf0003000 + 0x44 = 0xf0003044$

Bits 96-127 of *READER_ERROR_DATA*.

READER_ERROR_DATA2

Address: $0xf0003000 + 0x48 = 0xf0003048$

Bits 64-95 of *READER_ERROR_DATA*.

READER_ERROR_DATA1

Address: $0xf0003000 + 0x4c = 0xf000304c$

Bits 32-63 of *READER_ERROR_DATA*.

READER_ERROR_DATA0

Address: $0xf0003000 + 0x50 = 0xf0003050$

Bits 0-31 of *READER_ERROR_DATA*.

READER_ERROR_EXPECTED7

Address: $0xf0003000 + 0x54 = 0xf0003054$

Bits 224-255 of *READER_ERROR_EXPECTED*. Value expected to be read from DRAM memory

READER_ERROR_EXPECTED6

Address: $0xf0003000 + 0x58 = 0xf0003058$

Bits 192-223 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED5

Address: $0xf0003000 + 0x5c = 0xf000305c$

Bits 160-191 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED4

Address: $0xf0003000 + 0x60 = 0xf0003060$

Bits 128-159 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED3

Address: $0xf0003000 + 0x64 = 0xf0003064$

Bits 96-127 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED2

Address: $0xf0003000 + 0x68 = 0xf0003068$

Bits 64-95 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED1

Address: $0xf0003000 + 0x6c = 0xf000306c$

Bits 32-63 of *READER_ERROR_EXPECTED*.

READER_ERROR_EXPECTED0

Address: $0xf0003000 + 0x70 = 0xf0003070$

Bits 0-31 of *READER_ERROR_EXPECTED*.

READER_ERROR_READY

Address: $0xf0003000 + 0x74 = 0xf0003074$

Error detected and ready to read

READER_ERROR_CONTINUE

Address: $0xf0003000 + 0x78 = 0xf0003078$

Continue reading until the next error

17.2.8 DFI_SWITCH

Register Listing for DFI_SWITCH

Register	Address
<i>DFI_SWITCH_REFRESH_COUNT</i>	<i>0xf0003800</i>
<i>DFI_SWITCH_AT_REFRESH</i>	<i>0xf0003804</i>
<i>DFI_SWITCH_REFRESH_UPDATE</i>	<i>0xf0003808</i>

DFI_SWITCH_REFRESH_COUNT

Address: $0xf0003800 + 0x0 = 0xf0003800$

Count of all refresh commands issued (both by Memory Controller and Payload Executor). Value is latched from internal counter on mode transition: MC -> PE or by writing to the *refresh_update* CSR.

DFI_SWITCH_AT_REFRESH

Address: $0xf0003800 + 0x4 = 0xf0003804$

If set to a value different than 0 the mode transition MC -> PE will be performed only when the value of this register matches the current refresh commands count.

DFI_SWITCH_REFRESH_UPDATE

Address: $0xf0003800 + 0x8 = 0xf0003808$

Force an update of the *refresh_count* CSR.

17.2.9 PAYLOAD_EXECUTOR

Executes the DRAM payload from memory

Instruction decoder

All instructions are 32-bit. The format of most instructions is the same, except for the LOOP instruction, which has a constant TIMESLICE of 1.

NOOP with a TIMESLICE of 0 is a special case which is interpreted as STOP instruction. When this instruction is encountered execution gets finished immediately.

NOTE: TIMESLICE is the number of cycles the instruction will take. This means that instructions other than NOOP that use TIMESLICE=0 are illegal (although will silently be executed as having TIMESLICE=1).

NOTE2: LOOP instruction will *jump* COUNT times, meaning that the “code” inside the loop will effectively be executed COUNT+1 times.

Op codes:

Op	Value
NOOP	0b000
LOOP	0b111
ACT	0b100
PRE	0b101
REF	0b110
ZQC	0b001
READ	0b010

Instruction format:

LSB	MSB
dfi:	OP_CODE TIMESLICE ADDRESS
noop:	OP_CODE TIMESLICE_NOOP
loop:	OP_CODE COUNT JUMP
stop:	<NOOP> 0

Where ADDRESS depends on the DFI command and is one of:

LSB	MSB
RANK BANK COLUMN	
RANK BANK ROW	

Register Listing for PAYLOAD_EXECUTOR

Register	Address
PAYLOAD_EXECUTOR_START	0xf0004000
PAYLOAD_EXECUTOR_STATUS	0xf0004004
PAYLOAD_EXECUTOR_READ_COUNT	0xf0004008

PAYLOAD_EXECUTOR_START

Address: $0xf0004000 + 0x0 = 0xf0004000$

Writing to this register initializes payload execution

PAYLOAD_EXECUTOR_STATUS

Address: $0xf0004000 + 0x4 = 0xf0004004$

Payload executor status register

Field	Name	Description
[0]	READY	Indicates that the executor is not running
[1]	OVER-FLOW	Indicates the scratchpad memory address counter has overflowed due to the number of READ commands sent during execution

PAYLOAD_EXECUTOR_READ_COUNT

Address: $0xf0004000 + 0x8 = 0xf0004008$

Number of data from READ commands that is stored in the scratchpad memory

17.2.10 I2C

Register Listing for I2C

Register	Address
I2C_W	0xf0004800
I2C_R	0xf0004804

I2C_W

Address: $0xf0004800 + 0x0 = 0xf0004800$

Field	Name	Description

I2C_R

Address: $0xf0004800 + 0x4 = 0xf0004804$

Field	Name	Description

17.2.11 CTRL

Register Listing for CTRL

Register	Address
<i>CTRL_RESET</i>	<i>0xf0005000</i>
<i>CTRL_SCRATCH</i>	<i>0xf0005004</i>
<i>CTRL_BUS_ERRORS</i>	<i>0xf0005008</i>

CTRL__RESET

Address: $0xf0005000 + 0x0 = 0xf0005000$

Field	Name	Description
[0]	SOC_RST	Write 1 to this register to reset the full SoC (Pulse Reset)
[1]	CPU_RST	Write 1 to this register to reset the CPU(s) of the SoC (Hold Reset)

CTRL_SCRATCH

Address: $0xf0005000 + 0x4 = 0xf0005004$

Use this register as a scratch space to verify that software read/write accesses to the Wishbone/CSR bus are working correctly. The initial reset value of 0x1234578 can be used to verify endianness.

CTRL_BUS_ERRORS

Address: $0xf0005000 + 0x8 = 0xf0005008$

Total number of Wishbone bus errors (timeouts) since start.

17.2.12 ETPHY

Register Listing for ETPHY

Register	Address
<i>ETPHY_CRG_RESET</i>	<i>0xf0005800</i>
<i>ETPHY_MDIO_W</i>	<i>0xf0005804</i>
<i>ETPHY_MDIO_R</i>	<i>0xf0005808</i>

ETPHY_CRG_RESET

Address: $0xf0005800 + 0x0 = 0xf0005800$

ETPHY_MDIO_W

Address: $0xf0005800 + 0x4 = 0xf0005804$

Field	Name	Description

ETPHY_MDIO_R

Address: $0xf0005800 + 0x8 = 0xf0005808$

Field	Name	Description

17.2.13 IDENTIFIER_MEM

Register Listing for IDENTIFIER_MEM

Register	Address
<i>IDENTIFIER_MEM</i>	<i>0xf0006000</i>

IDENTIFIER_MEM

Address: $0xf0006000 + 0x0 = 0xf0006000$

8 x 109-bit memory

17.2.14 MAIN

Register Listing for MAIN

Register	Address
<i>MAIN_A_DQ_REMAPPING3</i>	<i>0xf0006800</i>
<i>MAIN_A_DQ_REMAPPING2</i>	<i>0xf0006804</i>
<i>MAIN_A_DQ_REMAPPING1</i>	<i>0xf0006808</i>
<i>MAIN_A_DQ_REMAPPING0</i>	<i>0xf000680c</i>
<i>MAIN_B_DQ_REMAPPING3</i>	<i>0xf0006810</i>
<i>MAIN_B_DQ_REMAPPING2</i>	<i>0xf0006814</i>
<i>MAIN_B_DQ_REMAPPING1</i>	<i>0xf0006818</i>
<i>MAIN_B_DQ_REMAPPING0</i>	<i>0xf000681c</i>

MAIN_A_DQ_REMAPPING3

Address: $0xf0006800 + 0x0 = 0xf0006800$

Bits 96-127 of *MAIN_A_DQ_REMAPPING*.

MAIN_A_DQ_REMAPPING2

Address: $0xf0006800 + 0x4 = 0xf0006804$

Bits 64-95 of *MAIN_A_DQ_REMAPPING*.

MAIN_A_DQ_REMAPPING1

Address: $0xf0006800 + 0x8 = 0xf0006808$

Bits 32-63 of *MAIN_A_DQ_REMAPPING*.

MAIN_A_DQ_REMAPPING0

Address: $0xf0006800 + 0xc = 0xf000680c$

Bits 0-31 of *MAIN_A_DQ_REMAPPING*.

MAIN_B_DQ_REMAPPING3

Address: $0xf0006800 + 0x10 = 0xf0006810$

Bits 96-127 of *MAIN_B_DQ_REMAPPING*.

MAIN_B_DQ_REMAPPING2

Address: $0xf0006800 + 0x14 = 0xf0006814$

Bits 64-95 of *MAIN_B_DQ_REMAPPING*.

MAIN_B_DQ_REMAPPING1

Address: $0xf0006800 + 0x18 = 0xf0006818$

Bits 32-63 of *MAIN_B_DQ_REMAPPING*.

MAIN_B_DQ_REMAPPING0

Address: $0xf0006800 + 0x1c = 0xf000681c$

Bits 0-31 of *MAIN_B_DQ_REMAPPING*.

17.2.15 SDRAM

Register Listing for SDRAM

Register	Address
<i>SDRAM_DFII_CONTROL</i>	<i>0xf0007000</i>
<i>SDRAM_DFII_FORCE_ISSUE</i>	<i>0xf0007004</i>
<i>SDRAM_DFII_A_CMDINJECTOR_COMMAND_STORAGE</i>	<i>0xf0007008</i>
<i>SDRAM_DFII_A_CMDINJECTOR_COMMAND_STORAGE_WR_MASK</i>	<i>0xf000700c</i>
<i>SDRAM_DFII_A_CMDINJECTOR_PHASE_ADDR</i>	<i>0xf0007010</i>

continues on next page

Table 17.3 – continued from previous page

Register	Address
<i>SDRAM_DFII_A_CMDINJECTOR_STORE_CONTINUOUS_CMD</i>	0xf0007014
<i>SDRAM_DFII_A_CMDINJECTOR_STORE_SINGLESHTO_CMD</i>	0xf0007018
<i>SDRAM_DFII_A_CMDINJECTOR_SINGLE_SHOT</i>	0xf000701c
<i>SDRAM_DFII_A_CMDINJECTOR_ISSUE_COMMAND</i>	0xf0007020
<i>SDRAM_DFII_A_CMDINJECTOR_WRDATA_SELECT</i>	0xf0007024
<i>SDRAM_DFII_A_CMDINJECTOR_WRDATA</i>	0xf0007028
<i>SDRAM_DFII_A_CMDINJECTOR_WRDATA_S</i>	0xf000702c
<i>SDRAM_DFII_A_CMDINJECTOR_WRDATA_STORE</i>	0xf0007030
<i>SDRAM_DFII_A_CMDINJECTOR_SETUP</i>	0xf0007034
<i>SDRAM_DFII_A_CMDINJECTOR_SAMPLE</i>	0xf0007038
<i>SDRAM_DFII_A_CMDINJECTOR_RESULT_ARRAY</i>	0xf000703c
<i>SDRAM_DFII_A_CMDINJECTOR_RESET</i>	0xf0007040
<i>SDRAM_DFII_A_CMDINJECTOR_RDDATA_SELECT</i>	0xf0007044
<i>SDRAM_DFII_A_CMDINJECTOR_RDDATA_CAPTURE_CNT</i>	0xf0007048
<i>SDRAM_DFII_A_CMDINJECTOR_RDDATA</i>	0xf000704c
<i>SDRAM_DFII_B_CMDINJECTOR_COMMAND_STORAGE</i>	0xf0007050
<i>SDRAM_DFII_B_CMDINJECTOR_COMMAND_STORAGE_WR_MASK</i>	0xf0007054
<i>SDRAM_DFII_B_CMDINJECTOR_PHASE_ADDR</i>	0xf0007058
<i>SDRAM_DFII_B_CMDINJECTOR_STORE_CONTINUOUS_CMD</i>	0xf000705c
<i>SDRAM_DFII_B_CMDINJECTOR_STORE_SINGLESHTO_CMD</i>	0xf0007060
<i>SDRAM_DFII_B_CMDINJECTOR_SINGLE_SHOT</i>	0xf0007064
<i>SDRAM_DFII_B_CMDINJECTOR_ISSUE_COMMAND</i>	0xf0007068
<i>SDRAM_DFII_B_CMDINJECTOR_WRDATA_SELECT</i>	0xf000706c
<i>SDRAM_DFII_B_CMDINJECTOR_WRDATA</i>	0xf0007070
<i>SDRAM_DFII_B_CMDINJECTOR_WRDATA_S</i>	0xf0007074
<i>SDRAM_DFII_B_CMDINJECTOR_WRDATA_STORE</i>	0xf0007078
<i>SDRAM_DFII_B_CMDINJECTOR_SETUP</i>	0xf000707c
<i>SDRAM_DFII_B_CMDINJECTOR_SAMPLE</i>	0xf0007080
<i>SDRAM_DFII_B_CMDINJECTOR_RESULT_ARRAY</i>	0xf0007084
<i>SDRAM_DFII_B_CMDINJECTOR_RESET</i>	0xf0007088
<i>SDRAM_DFII_B_CMDINJECTOR_RDDATA_SELECT</i>	0xf000708c
<i>SDRAM_DFII_B_CMDINJECTOR_RDDATA_CAPTURE_CNT</i>	0xf0007090
<i>SDRAM_DFII_B_CMDINJECTOR_RDDATA</i>	0xf0007094
<i>SDRAM_CONTROLLER_TRP</i>	0xf0007098
<i>SDRAM_CONTROLLER_TRCD</i>	0xf000709c
<i>SDRAM_CONTROLLER_TWR</i>	0xf00070a0
<i>SDRAM_CONTROLLER_TWTR</i>	0xf00070a4
<i>SDRAM_CONTROLLER_TREFI</i>	0xf00070a8
<i>SDRAM_CONTROLLER_TRFC</i>	0xf00070ac
<i>SDRAM_CONTROLLER_TFAW</i>	0xf00070b0
<i>SDRAM_CONTROLLER_TCCD</i>	0xf00070b4
<i>SDRAM_CONTROLLER_TCCD_WR</i>	0xf00070b8
<i>SDRAM_CONTROLLER_TRTP</i>	0xf00070bc
<i>SDRAM_CONTROLLER_TRRD</i>	0xf00070c0
<i>SDRAM_CONTROLLER_TRC</i>	0xf00070c4
<i>SDRAM_CONTROLLER_TRAS</i>	0xf00070c8
<i>SDRAM_CONTROLLER_LAST_ADDR_0</i>	0xf00070cc
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_0</i>	0xf00070d0

continues on next page

Table 17.3 – continued from previous page

Register	Address
<i>SDRAM_CONTROLLER_LAST_ADDR_1</i>	0xf00070d4
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_1</i>	0xf00070d8
<i>SDRAM_CONTROLLER_LAST_ADDR_2</i>	0xf00070dc
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_2</i>	0xf00070e0
<i>SDRAM_CONTROLLER_LAST_ADDR_3</i>	0xf00070e4
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_3</i>	0xf00070e8
<i>SDRAM_CONTROLLER_LAST_ADDR_4</i>	0xf00070ec
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_4</i>	0xf00070f0
<i>SDRAM_CONTROLLER_LAST_ADDR_5</i>	0xf00070f4
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_5</i>	0xf00070f8
<i>SDRAM_CONTROLLER_LAST_ADDR_6</i>	0xf00070fc
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_6</i>	0xf0007100
<i>SDRAM_CONTROLLER_LAST_ADDR_7</i>	0xf0007104
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_7</i>	0xf0007108
<i>SDRAM_CONTROLLER_LAST_ADDR_8</i>	0xf000710c
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_8</i>	0xf0007110
<i>SDRAM_CONTROLLER_LAST_ADDR_9</i>	0xf0007114
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_9</i>	0xf0007118
<i>SDRAM_CONTROLLER_LAST_ADDR_10</i>	0xf000711c
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_10</i>	0xf0007120
<i>SDRAM_CONTROLLER_LAST_ADDR_11</i>	0xf0007124
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_11</i>	0xf0007128
<i>SDRAM_CONTROLLER_LAST_ADDR_12</i>	0xf000712c
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_12</i>	0xf0007130
<i>SDRAM_CONTROLLER_LAST_ADDR_13</i>	0xf0007134
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_13</i>	0xf0007138
<i>SDRAM_CONTROLLER_LAST_ADDR_14</i>	0xf000713c
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_14</i>	0xf0007140
<i>SDRAM_CONTROLLER_LAST_ADDR_15</i>	0xf0007144
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_15</i>	0xf0007148
<i>SDRAM_CONTROLLER_LAST_ADDR_16</i>	0xf000714c
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_16</i>	0xf0007150
<i>SDRAM_CONTROLLER_LAST_ADDR_17</i>	0xf0007154
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_17</i>	0xf0007158
<i>SDRAM_CONTROLLER_LAST_ADDR_18</i>	0xf000715c
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_18</i>	0xf0007160
<i>SDRAM_CONTROLLER_LAST_ADDR_19</i>	0xf0007164
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_19</i>	0xf0007168
<i>SDRAM_CONTROLLER_LAST_ADDR_20</i>	0xf000716c
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_20</i>	0xf0007170
<i>SDRAM_CONTROLLER_LAST_ADDR_21</i>	0xf0007174
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_21</i>	0xf0007178
<i>SDRAM_CONTROLLER_LAST_ADDR_22</i>	0xf000717c
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_22</i>	0xf0007180
<i>SDRAM_CONTROLLER_LAST_ADDR_23</i>	0xf0007184
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_23</i>	0xf0007188
<i>SDRAM_CONTROLLER_LAST_ADDR_24</i>	0xf000718c
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_24</i>	0xf0007190

continues on next page

Table 17.3 – continued from previous page

Register	Address
<i>SDRAM_CONTROLLER_LAST_ADDR_25</i>	0xf0007194
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_25</i>	0xf0007198
<i>SDRAM_CONTROLLER_LAST_ADDR_26</i>	0xf000719c
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_26</i>	0xf00071a0
<i>SDRAM_CONTROLLER_LAST_ADDR_27</i>	0xf00071a4
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_27</i>	0xf00071a8
<i>SDRAM_CONTROLLER_LAST_ADDR_28</i>	0xf00071ac
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_28</i>	0xf00071b0
<i>SDRAM_CONTROLLER_LAST_ADDR_29</i>	0xf00071b4
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_29</i>	0xf00071b8
<i>SDRAM_CONTROLLER_LAST_ADDR_30</i>	0xf00071bc
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_30</i>	0xf00071c0
<i>SDRAM_CONTROLLER_LAST_ADDR_31</i>	0xf00071c4
<i>SDRAM_CONTROLLER_LAST_ACTIVE_ROW_31</i>	0xf00071c8

SDRAM_DFII_CONTROL

Address: 0xf0007000 + 0x0 = 0xf0007000

Control DFI signals common to all phases

Field	Name	Description						
[0]	SEL	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>0b0</td><td>Software (CPU) control.</td></tr> <tr> <td>0b1</td><td>Hardware control (default).</td></tr> </tbody> </table>	Value	Description	0b0	Software (CPU) control.	0b1	Hardware control (default).
Value	Description							
0b0	Software (CPU) control.							
0b1	Hardware control (default).							
[1]	CKE	DFI clock enable bus						
[2]	ODT	DFI on-die termination bus						
[3]	RESET_N	DFI clock reset bus						
[4]	MODE_2N	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>0b0</td><td>In 1N mode</td></tr> <tr> <td>0b1</td><td>In 2N mode (Default)</td></tr> </tbody> </table>	Value	Description	0b0	In 1N mode	0b1	In 2N mode (Default)
Value	Description							
0b0	In 1N mode							
0b1	In 2N mode (Default)							
[5]	A_CONTROL	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>0b1</td><td>A_Cmd Injector</td></tr> </tbody> </table>	Value	Description	0b1	A_Cmd Injector		
Value	Description							
0b1	A_Cmd Injector							
[6]	B_CONTROL	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>0b1</td><td>B_Cmd Injector</td></tr> </tbody> </table>	Value	Description	0b1	B_Cmd Injector		
Value	Description							
0b1	B_Cmd Injector							

SDRAM_DFII_FORCE_ISSUE

Address: $0xf0007000 + 0x4 = 0xf0007004$

SDRAM_DFII_A_CMDINJECTOR_COMMAND_STORAGE

Address: $0xf0007000 + 0x8 = 0xf0007008$

DDR5 command and control signals

Field	Name	Description
[13:0]	CA	Command/Address bus
[15:14]	CS	DFI chip select bus

SDRAM_DFII_A_CMDINJECTOR_COMMAND_STORAGE_WR_MASK

Address: $0xf0007000 + 0xc = 0xf000700c$

DDR5 wrdata mask control signals

Field	Name	Description

SDRAM_DFII_A_CMDINJECTOR_PHASE_ADDR

Address: $0xf0007000 + 0x10 = 0xf0007010$

SDRAM_DFII_A_CMDINJECTOR_STORE_CONTINUOUS_CMD

Address: $0xf0007000 + 0x14 = 0xf0007014$

SDRAM_DFII_A_CMDINJECTOR_STORE_SINGLESHOT_CMD

Address: $0xf0007000 + 0x18 = 0xf0007018$

SDRAM_DFII_A_CMDINJECTOR_SINGLE_SHOT

Address: $0xf0007000 + 0x1c = 0xf000701c$

SDRAM_DFII_A_CMDINJECTOR_ISSUE_COMMAND

Address: 0xf0007000 + 0x20 = 0xf0007020

SDRAM_DFII_A_CMDINJECTOR_WRDATA_SELECT

Address: 0xf0007000 + 0x24 = 0xf0007024

SDRAM_DFII_A_CMDINJECTOR_WRDATA

Address: 0xf0007000 + 0x28 = 0xf0007028

SDRAM_DFII_A_CMDINJECTOR_WRDATA_S

Address: 0xf0007000 + 0x2c = 0xf000702c

SDRAM_DFII_A_CMDINJECTOR_WRDATA_STORE

Address: 0xf0007000 + 0x30 = 0xf0007030

SDRAM_DFII_A_CMDINJECTOR_SETUP

Address: 0xf0007000 + 0x34 = 0xf0007034

Field	Name	Description
[0]	INITIAL_STATE	Initial value of all bits
[1]	OPERATION	0 - or (default), 1 -and

SDRAM_DFII_A_CMDINJECTOR_SAMPLE

Address: 0xf0007000 + 0x38 = 0xf0007038

SDRAM_DFII_A_CMDINJECTOR_RESULT_ARRAY

Address: 0xf0007000 + 0x3c = 0xf000703c

SDRAM_DFII_A_CMDINJECTOR_RESET

Address: 0xf0007000 + 0x40 = 0xf0007040

SDRAM_DFII_A_CMDINJECTOR_RDDATA_SELECT

Address: 0xf0007000 + 0x44 = 0xf0007044

SDRAM_DFII_A_CMDINJECTOR_RDDATA_CAPTURE_CNT

Address: 0xf0007000 + 0x48 = 0xf0007048

SDRAM_DFII_A_CMDINJECTOR_RDDATA

Address: 0xf0007000 + 0x4c = 0xf000704c

SDRAM_DFII_B_CMDINJECTOR_COMMAND_STORAGE

Address: 0xf0007000 + 0x50 = 0xf0007050

DDR5 command and control signals

Field	Name	Description
[13:0]	CA	Command/Address bus
[15:14]	CS	DFI chip select bus

SDRAM_DFII_B_CMDINJECTOR_COMMAND_STORAGE_WR_MASK

Address: 0xf0007000 + 0x54 = 0xf0007054

DDR5 wrdata mask control signals

Field	Name	Description

SDRAM_DFII_B_CMDINJECTOR_PHASE_ADDR

Address: 0xf0007000 + 0x58 = 0xf0007058

SDRAM_DFII_B_CMDINJECTOR_STORE_CONTINUOUS_CMD

Address: 0xf0007000 + 0x5c = 0xf000705c

SDRAM_DFII_B_CMDINJECTOR_STORE_SINGLESHOT_CMD

Address: 0xf0007000 + 0x60 = 0xf0007060

SDRAM_DFII_B_CMDINJECTOR_SINGLE_SHOT

Address: 0xf0007000 + 0x64 = 0xf0007064

SDRAM_DFII_B_CMDINJECTOR_ISSUE_COMMAND

Address: 0xf0007000 + 0x68 = 0xf0007068

SDRAM_DFII_B_CMDINJECTOR_WRDATA_SELECT

Address: 0xf0007000 + 0x6c = 0xf000706c

SDRAM_DFII_B_CMDINJECTOR_WRDATA

Address: 0xf0007000 + 0x70 = 0xf0007070

SDRAM_DFII_B_CMDINJECTOR_WRDATA_S

Address: 0xf0007000 + 0x74 = 0xf0007074

SDRAM_DFII_B_CMDINJECTOR_WRDATA_STORE

Address: 0xf0007000 + 0x78 = 0xf0007078

SDRAM_DFII_B_CMDINJECTOR_SETUP

Address: 0xf0007000 + 0x7c = 0xf000707c

Field	Name	Description
[0]	INITIAL_STATE	Initial value of all bits
[1]	OPERATION	0 - or (default), 1 -and

SDRAM_DFII_B_CMDINJECTOR_SAMPLE

Address: 0xf0007000 + 0x80 = 0xf0007080

SDRAM_DFII_B_CMDINJECTOR_RESULT_ARRAY

Address: 0xf0007000 + 0x84 = 0xf0007084

SDRAM_DFII_B_CMDINJECTOR_RESET

Address: 0xf0007000 + 0x88 = 0xf0007088

SDRAM_DFII_B_CMDINJECTOR_RDDATA_SELECT

Address: 0xf0007000 + 0x8c = 0xf000708c

SDRAM_DFII_B_CMDINJECTOR_RDDATA_CAPTURE_CNT

Address: 0xf0007000 + 0x90 = 0xf0007090

SDRAM_DFII_B_CMDINJECTOR_RDDATA

Address: 0xf0007000 + 0x94 = 0xf0007094

SDRAM_CONTROLLER_TRP

Address: 0xf0007000 + 0x98 = 0xf0007098

SDRAM_CONTROLLER_TRCD

Address: 0xf0007000 + 0x9c = 0xf000709c

SDRAM_CONTROLLER_TWR

Address: 0xf0007000 + 0xa0 = 0xf00070a0

SDRAM_CONTROLLER_TWTR

Address: 0xf0007000 + 0xa4 = 0xf00070a4

SDRAM_CONTROLLER_TREFI

Address: 0xf0007000 + 0xa8 = 0xf00070a8

SDRAM_CONTROLLER_TRFC

Address: 0xf0007000 + 0xac = 0xf00070ac

SDRAM_CONTROLLER_TFAW

Address: 0xf0007000 + 0xb0 = 0xf00070b0

SDRAM_CONTROLLER_TCCD

Address: 0xf0007000 + 0xb4 = 0xf00070b4

SDRAM_CONTROLLER_TCCD_WR

Address: 0xf0007000 + 0xb8 = 0xf00070b8

SDRAM_CONTROLLER_TRTP

Address: $0xf0007000 + 0xbc = 0xf00070bc$

SDRAM_CONTROLLER_TRRD

Address: $0xf0007000 + 0xc0 = 0xf00070c0$

SDRAM_CONTROLLER_TRC

Address: $0xf0007000 + 0xc4 = 0xf00070c4$

SDRAM_CONTROLLER_TRAS

Address: $0xf0007000 + 0xc8 = 0xf00070c8$

SDRAM_CONTROLLER_LAST_ADDR_0

Address: $0xf0007000 + 0xcc = 0xf00070cc$

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_0

Address: $0xf0007000 + 0xd0 = 0xf00070d0$

SDRAM_CONTROLLER_LAST_ADDR_1

Address: $0xf0007000 + 0xd4 = 0xf00070d4$

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_1

Address: 0xf0007000 + 0xd8 = 0xf00070d8

SDRAM_CONTROLLER_LAST_ADDR_2

Address: 0xf0007000 + 0xdc = 0xf00070dc

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_2

Address: 0xf0007000 + 0xe0 = 0xf00070e0

SDRAM_CONTROLLER_LAST_ADDR_3

Address: 0xf0007000 + 0xe4 = 0xf00070e4

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_3

Address: 0xf0007000 + 0xe8 = 0xf00070e8

SDRAM_CONTROLLER_LAST_ADDR_4

Address: 0xf0007000 + 0xec = 0xf00070ec

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_4

Address: 0xf0007000 + 0xf0 = 0xf00070f0

SDRAM_CONTROLLER_LAST_ADDR_5

Address: 0xf0007000 + 0xf4 = 0xf00070f4

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_5

Address: 0xf0007000 + 0xf8 = 0xf00070f8

SDRAM_CONTROLLER_LAST_ADDR_6

Address: 0xf0007000 + 0xfc = 0xf00070fc

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_6

Address: 0xf0007000 + 0x100 = 0xf0007100

SDRAM_CONTROLLER_LAST_ADDR_7

Address: 0xf0007000 + 0x104 = 0xf0007104

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_7

Address: 0xf0007000 + 0x108 = 0xf0007108

SDRAM_CONTROLLER_LAST_ADDR_8

Address: 0xf0007000 + 0x10c = 0xf000710c

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_8

Address: 0xf0007000 + 0x110 = 0xf0007110

SDRAM_CONTROLLER_LAST_ADDR_9

Address: 0xf0007000 + 0x114 = 0xf0007114

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_9

Address: 0xf0007000 + 0x118 = 0xf0007118

SDRAM_CONTROLLER_LAST_ADDR_10

Address: 0xf0007000 + 0x11c = 0xf000711c

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_10

Address: 0xf0007000 + 0x120 = 0xf0007120

SDRAM_CONTROLLER_LAST_ADDR_11

Address: 0xf0007000 + 0x124 = 0xf0007124

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_11

Address: 0xf0007000 + 0x128 = 0xf0007128

SDRAM_CONTROLLER_LAST_ADDR_12

Address: 0xf0007000 + 0x12c = 0xf000712c

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_12

Address: 0xf0007000 + 0x130 = 0xf0007130

SDRAM_CONTROLLER_LAST_ADDR_13

Address: 0xf0007000 + 0x134 = 0xf0007134

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_13

Address: 0xf0007000 + 0x138 = 0xf0007138

SDRAM_CONTROLLER_LAST_ADDR_14

Address: 0xf0007000 + 0x13c = 0xf000713c

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_14

Address: 0xf0007000 + 0x140 = 0xf0007140

SDRAM_CONTROLLER_LAST_ADDR_15

Address: 0xf0007000 + 0x144 = 0xf0007144

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_15

Address: 0xf0007000 + 0x148 = 0xf0007148

SDRAM_CONTROLLER_LAST_ADDR_16

Address: 0xf0007000 + 0x14c = 0xf000714c

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_16

Address: 0xf0007000 + 0x150 = 0xf0007150

SDRAM_CONTROLLER_LAST_ADDR_17

Address: 0xf0007000 + 0x154 = 0xf0007154

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_17

Address: 0xf0007000 + 0x158 = 0xf0007158

SDRAM_CONTROLLER_LAST_ADDR_18

Address: 0xf0007000 + 0x15c = 0xf000715c

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_18

Address: 0xf0007000 + 0x160 = 0xf0007160

SDRAM_CONTROLLER_LAST_ADDR_19

Address: 0xf0007000 + 0x164 = 0xf0007164

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_19

Address: 0xf0007000 + 0x168 = 0xf0007168

SDRAM_CONTROLLER_LAST_ADDR_20

Address: 0xf0007000 + 0x16c = 0xf000716c

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_20

Address: 0xf0007000 + 0x170 = 0xf0007170

SDRAM_CONTROLLER_LAST_ADDR_21

Address: 0xf0007000 + 0x174 = 0xf0007174

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_21

Address: 0xf0007000 + 0x178 = 0xf0007178

SDRAM_CONTROLLER_LAST_ADDR_22

Address: 0xf0007000 + 0x17c = 0xf000717c

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_22

Address: 0xf0007000 + 0x180 = 0xf0007180

SDRAM_CONTROLLER_LAST_ADDR_23

Address: 0xf0007000 + 0x184 = 0xf0007184

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_23

Address: 0xf0007000 + 0x188 = 0xf0007188

SDRAM_CONTROLLER_LAST_ADDR_24

Address: 0xf0007000 + 0x18c = 0xf000718c

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_24

Address: 0xf0007000 + 0x190 = 0xf0007190

SDRAM_CONTROLLER_LAST_ADDR_25

Address: 0xf0007000 + 0x194 = 0xf0007194

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_25

Address: 0xf0007000 + 0x198 = 0xf0007198

SDRAM_CONTROLLER_LAST_ADDR_26

Address: 0xf0007000 + 0x19c = 0xf000719c

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_26

Address: 0xf0007000 + 0x1a0 = 0xf00071a0

SDRAM_CONTROLLER_LAST_ADDR_27

Address: 0xf0007000 + 0x1a4 = 0xf00071a4

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_27

Address: 0xf0007000 + 0x1a8 = 0xf00071a8

SDRAM_CONTROLLER_LAST_ADDR_28

Address: 0xf0007000 + 0x1ac = 0xf00071ac

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_28

Address: 0xf0007000 + 0x1b0 = 0xf00071b0

SDRAM_CONTROLLER_LAST_ADDR_29

Address: 0xf0007000 + 0x1b4 = 0xf00071b4

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_29

Address: $0xf0007000 + 0x1b8 = 0xf00071b8$

SDRAM_CONTROLLER_LAST_ADDR_30

Address: $0xf0007000 + 0x1bc = 0xf00071bc$

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_30

Address: $0xf0007000 + 0x1c0 = 0xf00071c0$

SDRAM_CONTROLLER_LAST_ADDR_31

Address: $0xf0007000 + 0x1c4 = 0xf00071c4$

SDRAM_CONTROLLER_LAST_ACTIVE_ROW_31

Address: $0xf0007000 + 0x1c8 = 0xf00071c8$

17.2.16 SDRAM_CHECKER

Register Listing for SDRAM_CHECKER

Register	Address
<i>SDRAM_CHECKER_RESET</i>	<i>0xf0007800</i>
<i>SDRAM_CHECKER_START</i>	<i>0xf0007804</i>
<i>SDRAM_CHECKER_DONE</i>	<i>0xf0007808</i>
<i>SDRAM_CHECKER_BASE1</i>	<i>0xf000780c</i>
<i>SDRAM_CHECKER_BASE0</i>	<i>0xf0007810</i>
<i>SDRAM_CHECKER_END1</i>	<i>0xf0007814</i>
<i>SDRAM_CHECKER_END0</i>	<i>0xf0007818</i>
<i>SDRAM_CHECKER_LENGTH1</i>	<i>0xf000781c</i>
<i>SDRAM_CHECKER_LENGTH0</i>	<i>0xf0007820</i>
<i>SDRAM_CHECKER_RANDOM</i>	<i>0xf0007824</i>
<i>SDRAM_CHECKER_TICKS</i>	<i>0xf0007828</i>
<i>SDRAM_CHECKER_ERRORS</i>	<i>0xf000782c</i>

SDRAM_CHECKER_RESET

Address: 0xf0007800 + 0x0 = 0xf0007800

SDRAM_CHECKER_START

Address: 0xf0007800 + 0x4 = 0xf0007804

SDRAM_CHECKER_DONE

Address: 0xf0007800 + 0x8 = 0xf0007808

SDRAM_CHECKER_BASE1

Address: 0xf0007800 + 0xc = 0xf000780c

Bits 32-35 of *SDRAM_CHECKER_BASE*.

SDRAM_CHECKER_BASE0

Address: 0xf0007800 + 0x10 = 0xf0007810

Bits 0-31 of *SDRAM_CHECKER_BASE*.

SDRAM_CHECKER_END1

Address: 0xf0007800 + 0x14 = 0xf0007814

Bits 32-35 of *SDRAM_CHECKER_END*.

SDRAM_CHECKER_END0

Address: 0xf0007800 + 0x18 = 0xf0007818

Bits 0-31 of *SDRAM_CHECKER_END*.

SDRAM_CHECKER_LENGTH1

Address: $0xf0007800 + 0x1c = 0xf000781c$

Bits 32-35 of *SDRAM_CHECKER_LENGTH*.

SDRAM_CHECKER_LENGTH0

Address: $0xf0007800 + 0x20 = 0xf0007820$

Bits 0-31 of *SDRAM_CHECKER_LENGTH*.

SDRAM_CHECKER_RANDOM

Address: $0xf0007800 + 0x24 = 0xf0007824$

Field	Name	Description

SDRAM_CHECKER_TICKS

Address: $0xf0007800 + 0x28 = 0xf0007828$

SDRAM_CHECKER_ERRORS

Address: $0xf0007800 + 0x2c = 0xf000782c$

17.2.17 SDRAM_GENERATOR

Register Listing for SDRAM_GENERATOR

Register	Address
<i>SDRAM_GENERATOR_RESET</i>	0xf0008000
<i>SDRAM_GENERATOR_START</i>	0xf0008004
<i>SDRAM_GENERATOR_DONE</i>	0xf0008008
<i>SDRAM_GENERATOR_BASE1</i>	0xf000800c
<i>SDRAM_GENERATOR_BASE0</i>	0xf0008010
<i>SDRAM_GENERATOR_END1</i>	0xf0008014
<i>SDRAM_GENERATOR_END0</i>	0xf0008018
<i>SDRAM_GENERATOR_LENGTH1</i>	0xf000801c
<i>SDRAM_GENERATOR_LENGTH0</i>	0xf0008020
<i>SDRAM_GENERATOR_RANDOM</i>	0xf0008024
<i>SDRAM_GENERATOR_TICKS</i>	0xf0008028

SDRAM_GENERATOR_RESET

Address: $0xf0008000 + 0x0 = 0xf0008000$

SDRAM_GENERATOR_START

Address: $0xf0008000 + 0x4 = 0xf0008004$

SDRAM_GENERATOR_DONE

Address: $0xf0008000 + 0x8 = 0xf0008008$

SDRAM_GENERATOR_BASE1

Address: $0xf0008000 + 0xc = 0xf000800c$

Bits 32-35 of *SDRAM_GENERATOR_BASE*.

SDRAM_GENERATOR_BASE0

Address: $0xf0008000 + 0x10 = 0xf0008010$

Bits 0-31 of *SDRAM_GENERATOR_BASE*.

SDRAM_GENERATOR_END1

Address: $0xf0008000 + 0x14 = 0xf0008014$

Bits 32-35 of *SDRAM_GENERATOR_END*.

SDRAM_GENERATOR_END0

Address: $0xf0008000 + 0x18 = 0xf0008018$

Bits 0-31 of *SDRAM_GENERATOR_END*.

SDRAM_GENERATOR_LENGTH1

Address: $0xf0008000 + 0x1c = 0xf000801c$

Bits 32-35 of *SDRAM_GENERATOR_LENGTH*.

SDRAM_GENERATOR_LENGTH0

Address: $0xf0008000 + 0x20 = 0xf0008020$

Bits 0-31 of *SDRAM_GENERATOR_LENGTH*.

SDRAM_GENERATOR_RANDOM

Address: $0xf0008000 + 0x24 = 0xf0008024$

Field	Name	Description

SDRAM_GENERATOR_TICKS

Address: $0xf0008000 + 0x28 = 0xf0008028$

17.2.18 TIMERO

Timer

Provides a generic Timer core.

The Timer is implemented as a countdown timer that can be used in various modes:

- Polling : Returns current countdown value to software
- One-Shot: Loads itself and stops when value reaches 0
- Periodic: (Re-)Loads itself when value reaches 0

en register allows the user to enable/disable the Timer. When the Timer is enabled, it is automatically loaded with the value of *load* register.

When the Timer reaches 0, it is automatically reloaded with value of *reload* register.

The user can latch the current countdown value by writing to *update_value* register, it will update *value* register with current countdown value.

To use the Timer in One-Shot mode, the user needs to:

- Disable the timer
- Set the *load* register to the expected duration
- (Re-)Enable the Timer

To use the Timer in Periodic mode, the user needs to:

- Disable the Timer
- Set the *load* register to 0
- Set the *reload* register to the expected period
- Enable the Timer

For both modes, the CPU can be advertised by an IRQ that the duration/period has elapsed. (The CPU can also do software polling with *update_value* and *value* to know the elapsed duration)

Register Listing for TIMERO

Register	Address
<i>TIMERO_LOAD</i>	0xf0008800
<i>TIMERO_RELOAD</i>	0xf0008804
<i>TIMERO_EN</i>	0xf0008808
<i>TIMERO_UPDATE_VALUE</i>	0xf000880c
<i>TIMERO_VALUE</i>	0xf0008810
<i>TIMERO_EV_STATUS</i>	0xf0008814
<i>TIMERO_EV_PENDING</i>	0xf0008818
<i>TIMERO_EV_ENABLE</i>	0xf000881c

TIMERO_LOAD

Address: $0xf0008800 + 0x0 = 0xf0008800$

Load value when Timer is (re-)enabled. In One-Shot mode, the value written to this register specifies the Timer's duration in clock cycles.

TIMERO_RELOAD

Address: $0xf0008800 + 0x4 = 0xf0008804$

Reload value when Timer reaches 0. In Periodic mode, the value written to this register specify the Timer's period in clock cycles.

TIMERO_EN

Address: $0xf0008800 + 0x8 = 0xf0008808$

Enable flag of the Timer. Set this flag to 1 to enable/start the Timer. Set to 0 to disable the Timer.

TIMERO_UPDATE_VALUE

Address: $0xf0008800 + 0xc = 0xf000880c$

Update trigger for the current countdown value. A write to this register latches the current countdown value to value register.

TIMER0_VALUE

Address: $0xf0008800 + 0x10 = 0xf0008810$

Latched countdown value. This value is updated by writing to update_value.

TIMER0_EV_STATUS

Address: $0xf0008800 + 0x14 = 0xf0008814$

This register contains the current raw level of the zero event trigger. Writes to this register have no effect.

Field	Name	Description
[0]	ZERO	Level of the zero event

TIMER0_EV_PENDING

Address: $0xf0008800 + 0x18 = 0xf0008818$

When a zero event occurs, the corresponding bit will be set in this register. To clear the Event, set the corresponding bit in this register.

Field	Name	Description
[0]	ZERO	1 if a zero event occurred. This Event is triggered on a falling edge.

TIMER0_EV_ENABLE

Address: $0xf0008800 + 0x1c = 0xf000881c$

This register enables the corresponding zero events. Write a 0 to this register to disable individual events.

Field	Name	Description
[0]	ZERO	Write a 1 to enable the zero Event

17.2.19 UART

Register Listing for UART

Register	Address
<code>UART_RXTX</code>	<code>0xf0009000</code>
<code>UART_TXFULL</code>	<code>0xf0009004</code>
<code>UART_RXEMPTY</code>	<code>0xf0009008</code>
<code>UART_EV_STATUS</code>	<code>0xf000900c</code>
<code>UART_EV_PENDING</code>	<code>0xf0009010</code>
<code>UART_EV_ENABLE</code>	<code>0xf0009014</code>
<code>UART_TXEMPTY</code>	<code>0xf0009018</code>
<code>UART_RXFULL</code>	<code>0xf000901c</code>
<code>UART_XOVER_RXTX</code>	<code>0xf0009020</code>
<code>UART_XOVER_TXFULL</code>	<code>0xf0009024</code>
<code>UART_XOVER_RXEMPTY</code>	<code>0xf0009028</code>
<code>UART_XOVER_EV_STATUS</code>	<code>0xf000902c</code>
<code>UART_XOVER_EV_PENDING</code>	<code>0xf0009030</code>
<code>UART_XOVER_EV_ENABLE</code>	<code>0xf0009034</code>
<code>UART_XOVER_TXEMPTY</code>	<code>0xf0009038</code>
<code>UART_XOVER_RXFULL</code>	<code>0xf000903c</code>

UART_RXTX

Address: $0xf0009000 + 0x0 = 0xf0009000$

UART_TXFULL

Address: $0xf0009000 + 0x4 = 0xf0009004$

TX FIFO Full.

UART_RXEMPTY

Address: $0xf0009000 + 0x8 = 0xf0009008$

RX FIFO Empty.

UART_EV_STATUS

Address: $0xf0009000 + 0xc = 0xf000900c$

This register contains the current raw level of the rx event trigger. Writes to this register have no effect.

Field	Name	Description
[0]	TX	Level of the tx event
[1]	RX	Level of the rx event

UART_EV_PENDING

Address: $0xf0009000 + 0x10 = 0xf0009010$

When a rx event occurs, the corresponding bit will be set in this register. To clear the Event, set the corresponding bit in this register.

Field	Name	Description
[0]	TX	1 if a tx event occurred. This Event is triggered on a falling edge.
[1]	RX	1 if a rx event occurred. This Event is triggered on a falling edge.

UART_EV_ENABLE

Address: $0xf0009000 + 0x14 = 0xf0009014$

This register enables the corresponding rx events. Write a 0 to this register to disable individual events.

Field	Name	Description
[0]	TX	Write a 1 to enable the tx Event
[1]	RX	Write a 1 to enable the rx Event

UART_TXEMPTY

Address: $0xf0009000 + 0x18 = 0xf0009018$

TX FIFO Empty.

UART_RXFULL

Address: $0xf0009000 + 0x1c = 0xf000901c$

RX FIFO Full.

UART_XOVER_RXTX

Address: $0xf0009000 + 0x20 = 0xf0009020$

UART_XOVER_TXFULL

Address: $0xf0009000 + 0x24 = 0xf0009024$

TX FIFO Full.

UART_XOVER_RXEMPTY

Address: $0xf0009000 + 0x28 = 0xf0009028$

RX FIFO Empty.

UART_XOVER_EV_STATUS

Address: $0xf0009000 + 0x2c = 0xf000902c$

This register contains the current raw level of the rx event trigger. Writes to this register have no effect.

Field	Name	Description
[0]	TX	Level of the tx event
[1]	RX	Level of the rx event

UART_XOVER_EV_PENDING

Address: $0xf0009000 + 0x30 = 0xf0009030$

When a rx event occurs, the corresponding bit will be set in this register. To clear the Event, set the corresponding bit in this register.

Field	Name	Description
[0]	TX	1 if a tx event occurred. This Event is triggered on a falling edge.
[1]	RX	1 if a rx event occurred. This Event is triggered on a falling edge.

UART_XOVER_EV_ENABLE

Address: $0xf0009000 + 0x34 = 0xf0009034$

This register enables the corresponding rx events. Write a 0 to this register to disable individual events.

Field	Name	Description
[0]	TX	Write a 1 to enable the tx Event
[1]	RX	Write a 1 to enable the rx Event

UART_XOVER_TXEMPTY

Address: $0xf0009000 + 0x38 = 0xf0009038$

TX FIFO Empty.

UART_XOVER_RXFULL

Address: $0xf0009000 + 0x3c = 0xf000903c$

RX FIFO Full.