HyperRAM interface

This document is a guide to understand the code implementation and decisions made as the development of the interface is in process.

Note that various signals and I/O ports have been modified in order to be able to debug as the code is being implemented. The final code will have the same ports as in [this link](https://infrequent-cormorant-741.notion.site/HyperRAM-interface-a996bf71213f4ebea602a78541cd65c9).

The information for the configuration of the devices was taken from Winbond’s product datasheet W958D8NBYA5I.pdf.

**Files**

Three vhd files are presented.

clk\_div\_2.vhd is a simple clock divider which is presented in a separate file for more clarity.

hyperram.vhd is the file where the behavior of the interface is being implemented.

hyperram\_tb.vhd is the testbench for the project.

**Implementation**

**Initial setup**

In order to get the requested behavior, the program starts requesting the devices ID’s. This is done via a read request considering some particular requirements that the datasheet informs. Note that the bidirectional signals have been separated in order to be able to perform the simulations since ModelSim’s tool does not allow to edit bidirectional signals. All the debug signals and ports are identified with a \_dbg suffix to be removed and replaced by the definitive signals and ports.

The transaction then starts by lowering CS with the RAM’s clock in 0 as the datasheet requests.

Six bytes have to be sent to request each one of the identification registers. The devices respond with two bytes which are repeated, so only the first two are considered for each register. The latency at this point should be 7 since it has not yet been configured but the code just waits for a raising edge in RWDS so that is not considered.

The command addresses that are sent are “E0 00 00 00 00 00” for the first register and “E0 00 00 00 00 01” for the second and the data that is expected to be read is “0E86” and “0001”.

If everything is correct, then the configuration of the devices is performed. If not, an error flag is set and the interface will be halted until reset.

For the configuration, no latency is needed, so the command and data are sent consecutively.

The addressing is “60 00 01 00 00 00” for command register 0 and “60 00 01 00 00 01” for command register 1. The data that will be sent is “DFEC” and “FF81” respectively. This configures the devices as requested (burst length 128, Fixed latency, minimum latency 3,27 Ohm drive strength). Then the interface is put in idle state waiting for read/write requests from the user with ready = ‘1’.

**Write cycle**

Once the flag ready is set, the user can request a write cycle by setting write\_start and writing start\_adress. The ready\_flag is then cleared and the writing cycle starts. CS\_n is cleared and the Command Address bytes are sent in DQ. The bytes sent are “20” to request a write to memory in a linear burst in the first three bits, the rest are reserved and set to 0. Then three 0 bits are sent followed by the memory address bits A23..A3. Another 13 0 bits are sent and finally A2..A0. Note that the address was 32 bit long but the first 8 bits are unusable for the memory’s density and are set to 0. So if the user writes those bits they will be ignored. Then the latency time is waited, counting 6 ram’s clock cycles (fixed latency 2 times 3) from the moment that the 4th byte is read. So in the 16th clock count the burst begins. Data is read from the FWFT FIFO as write\_fifo\_data\_rd\_enable is set and the byte count increments until 128 bytes are read from the FIFO and written to the memory. All this happens if the FIFO is not empty and during the falling edge of the system’s clock in order to have the data centered in the memory’s edges. This is the only moment that RWDS is driven by the controller and it is set to 0 to indicate to the memories that the bytes should be written (it can be used as a mask if set to 1). After the 128 bytes are written both the FIFO and memory are released and the controller becomes ready to start another transaction.

**Read cycle**

Once the flag ready is set, the user can request a read cycle by setting read\_start and writing start\_adress. The ready\_flag is then cleared and the reading cycle starts. CS\_n is cleared and the Command Address bytes are sent in DQ. The bytes sent are “A0” to request a read from memory in a linear burst in the first three bits, the rest are reserved and set to 0. Then three 0 bits are sent followed by the memory address bits A23..A3. Another 13 0 bits are sent and finally A2..A0. Note that the address was 32 bit long but the first 8 bits are unusable for the memory’s density and are set to 0. So if the user writes those bits they will be ignored. Then the latency time is waited, counting 6 ram’s clock cycles (fixed latency 2 times 3) from the moment that the 4th byte is read. So in the 16th clock count the burst begins.

As long as read\_fifo\_full is 0, read\_fifo\_data\_wr\_enable is set and data is read from the memories and written directly into the FIFO. The byte count increments until 128 bytes are read from the memory and written to the FIFO. RWDS is driven by the memory and it is used as trigger to read from the memory and write to the FIFO since it is DDR it doesn’t require additional logic. After the 128 bytes are read both the FIFO and memory are released and the controller becomes ready to start another transaction.

**Simulations**

To verify the project’s performance, a testbench was implemented in the file hyperram\_tb.vhd. There signals were created to emulate the behavior of both the memories and FIFO for setup, read and write cycles. All three parts were verified.