Dram Read Implementation

1) Overall design:
   The entity Dram Read is a DMA interface which is used to deal with the communication between external Dram and user application. Through this entity, the complex transaction of reading Dram can be simplified. User application designer does not need to know how the DRAM works and pay extra attention to design corresponding circuits. My design has one FIFO ip and four major blocks. They are address generator, handshake synchronizer, done generator and parity check.

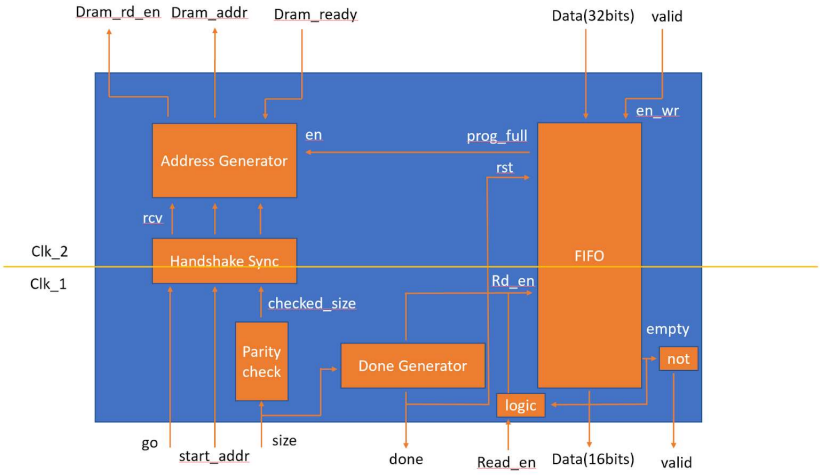

Fig. 1 overall design of DRAM_RD

2) Parity check for input size signal:
   First, due to different bits of input data and output data, we should convert the input size to output size. Input data has 32 bits but output data has 16 bits, which means one input data will be splitted into two output data in FIFO. Therefore, just one memory access can get two data. However, if the size is an odd number, for example size equals to 3 which cannot be divided by 2, we need 2 memory accesses to get 3 data. Because of this, we design an entity called parity check to deal with this problem. In our design, we determine whether the size number is an odd number or even number by checking the least bit of the binary size number. If the least bit is '0', the number is an even number. It will simply right shift 1 bit. If the least bit is '1', it means the number is an odd number. It will right shift 1 bit first and then plus 1. The testbench simulation result is shown in figure 2. When input size equals 1 or 2, we just need one memory access, so the output size is one. The following size signal is converted correctly as we expected.

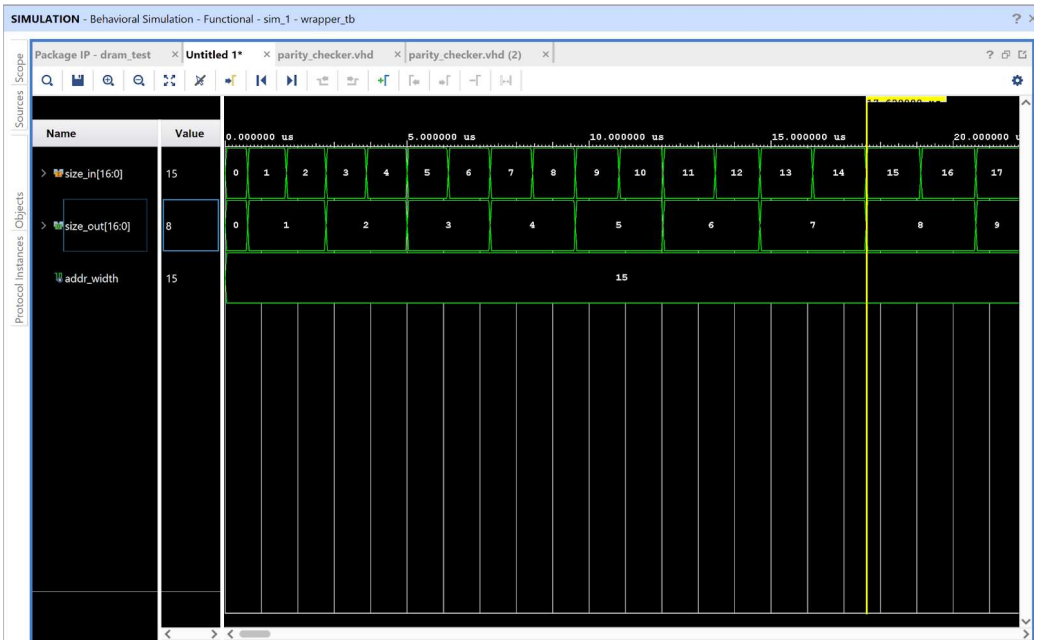

Fig. 2 Parity check simulation results

3) Handshake synchronizer:

After converting size signals, checked_size, go and start address are sent to the handshake synchronizer because the destination of these signals is in the different clock-domain. Handshake synchronizer is quite useful to synchronize single multi-bits signals(non-stream data). When the go signal is asserted, the source FSM starts to work. It sends a send signal to tell the destination FSM it is ready and sends an enable signal to the source registers to transfer data. After synchronizing the send signal by dual Flip flop, the send signal is received by the destination_FSM. The destination_FSM begins to generate an enable signal to the destination registers to receive data and a rcv signal to tell outside blocks it has received the data. At same time, the destination_FSM sends a ack signal to source_FSM to tell it to stop transfer. Finally, Source_FSM stops to generate the enable signal and asserts '0' to send signal to stop destination_FSM.

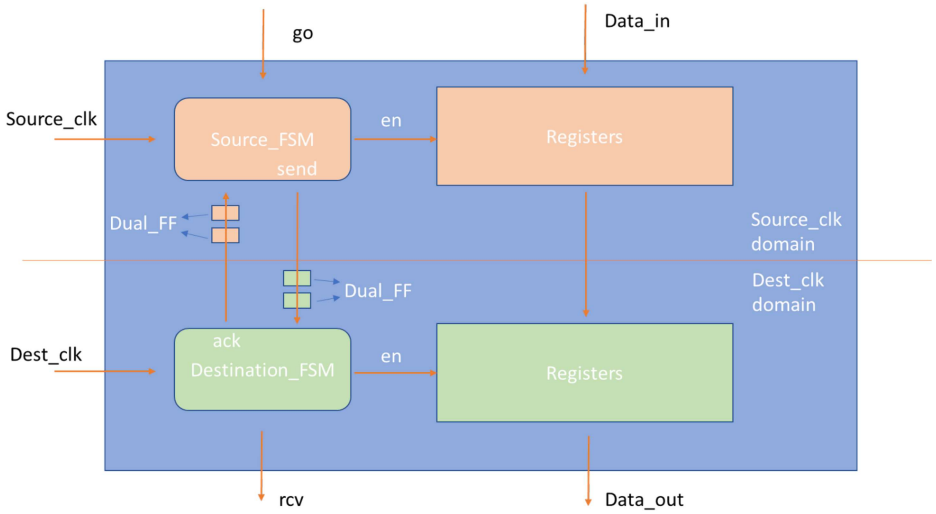

Fig. 3 handshake synchronizer simple schematic

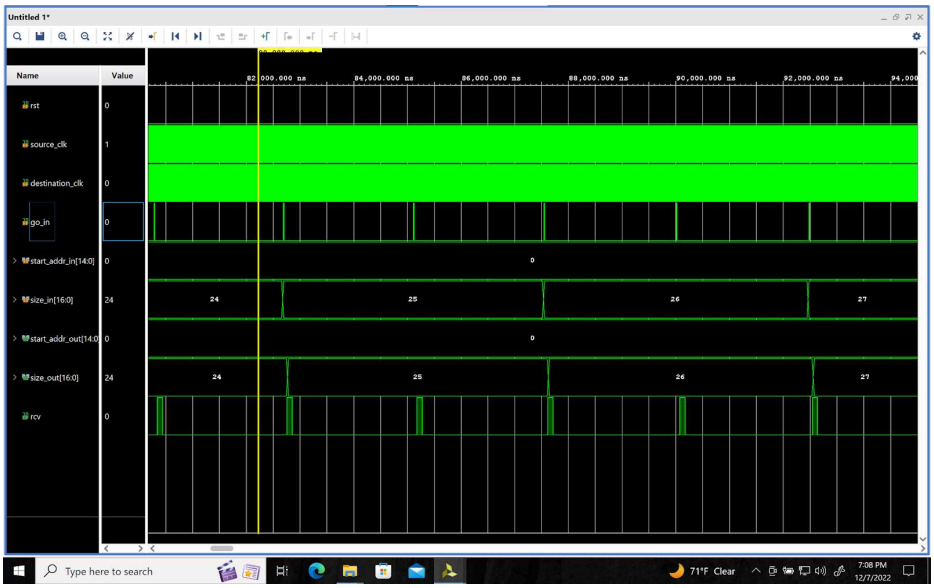

Fig. 4 handshake synchronizer simulation results.

4) Read Address generator:
After getting check_size, go and start address, the address generator is going to generate DRAM read address. The generator has a counter and a 2-process FSMD. The counter is used to generate addresses. The FSMD is responsible for generating dram_rd_en to tell DRAM which address is valid and can be used. When go is asserted, the FSM and counter will reset. The start address will be assigned to dram_addr signal . However, they will wait for addr_gen_en signal to be high. The logic of addr_gen_en is (dram_ready AND not (prog_full)), which means the generator works only when dram can be accessed and the fifo is about to be full. Then two parts start to work. The counter will maintain working state until addr_gen_en is low signal, but we do not need so many addresses. Therefore, the FSMD will stop to assert dram_rd_en signal when the counter generates sufficient addresses. The next go signal will repeat the same progress.

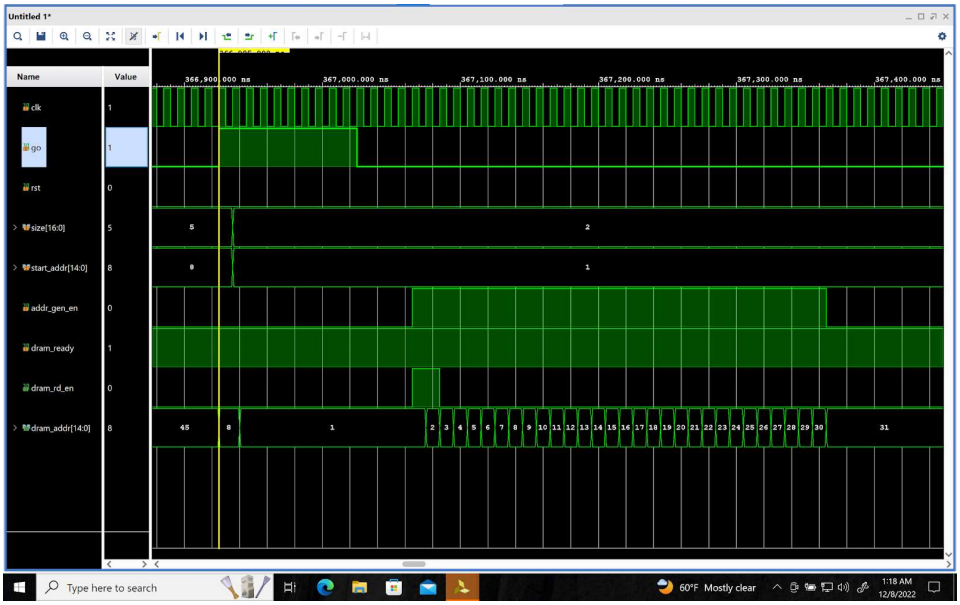

Fig. 5 Address generator simulation result

5) Done signal generator:

The FIFO splits the 32 bits data to two 16 bits data. After reading specific numbers of data, the done signal generator starts to work to output a done signal to tell user_app the data it needs is ready. In the generator, its core is a 2-process FSMD. It has a variable to record the number of data which have been read. When the variable is equal to the size, it is going to assert done until the next go asserted. If the size is an odd number, there will be useless data left in the FIFO. Therefore, the done signal also uses a dual Flip-Flop synchronizer to cross clock-domain, reaching the rst port of the FIFO to delete left data.

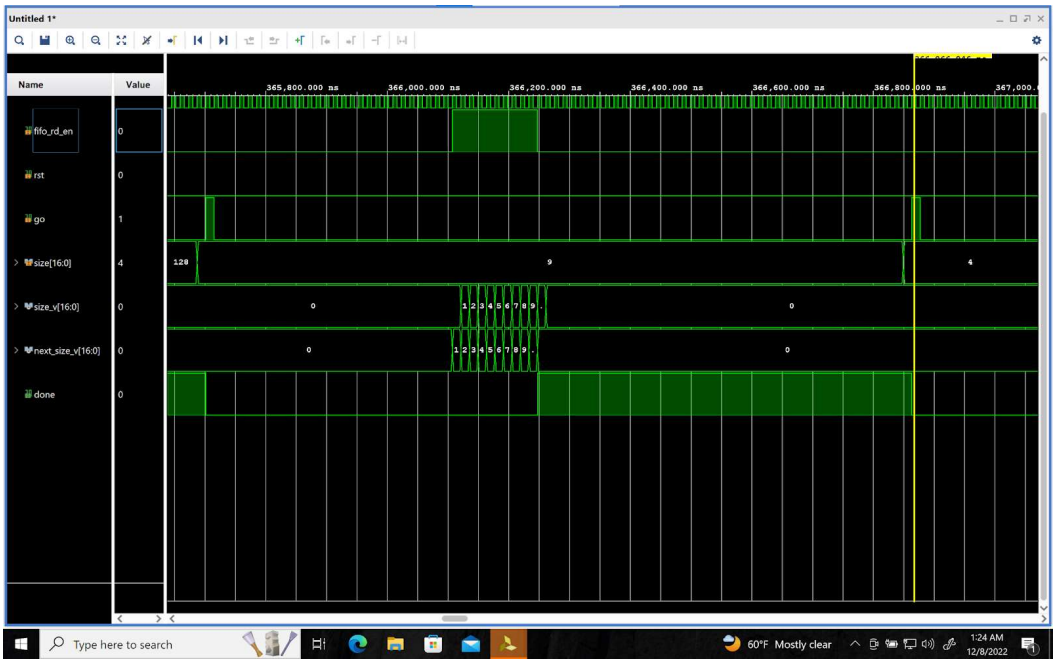

Fig. 6 Done signal generator simulation result