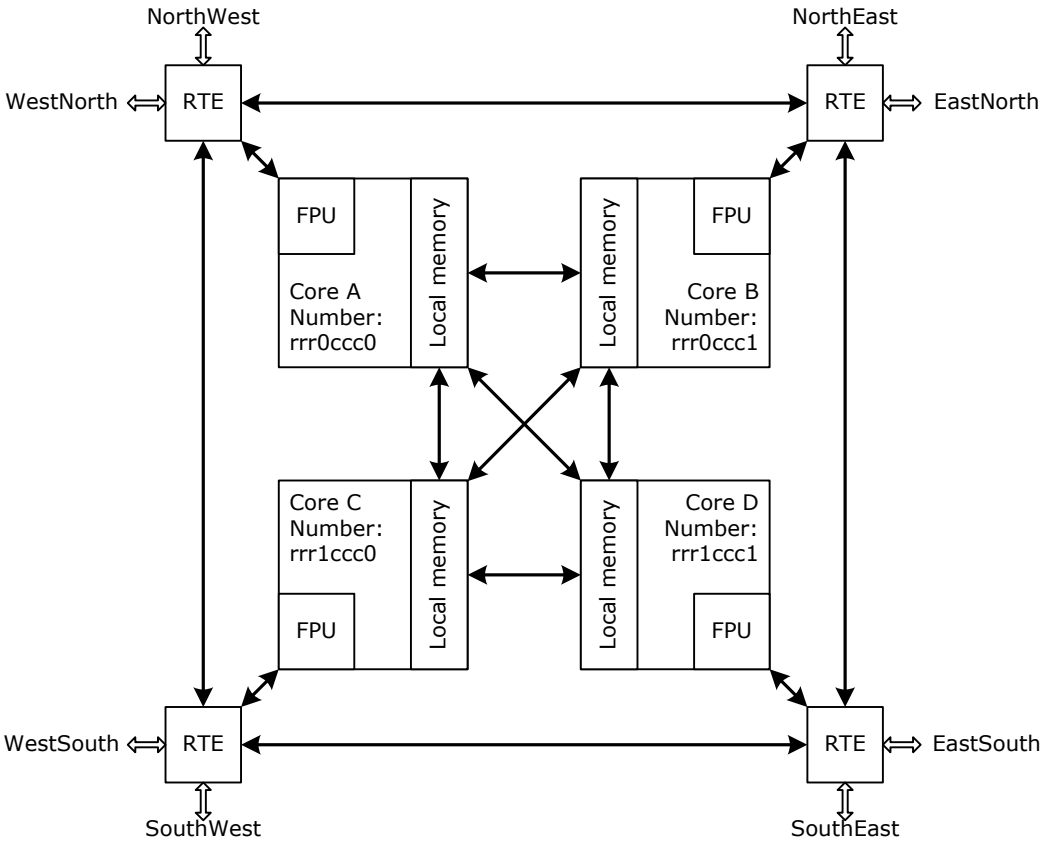
Multi-core processor using optical transceivers.

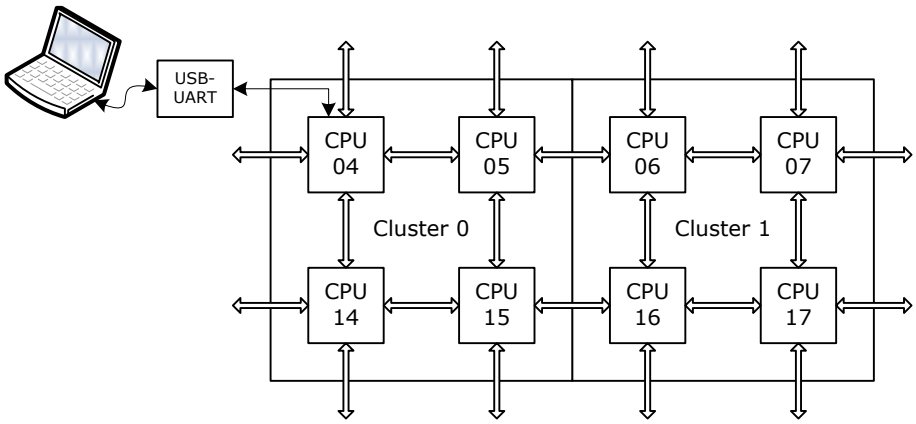
The experiment was done to answer the questions: "is it possible to make a multi-core processor by connecting the cores with optical communication channels and what communication characteristics can be obtained." For the experiment, 2 identical DE5-Net boards with FPGA 5SGXEA7N2F45C2 were taken. These FPGAs allow the configuration of 4-core experimental X16 processors, the architecture of which is shown in the figure below.



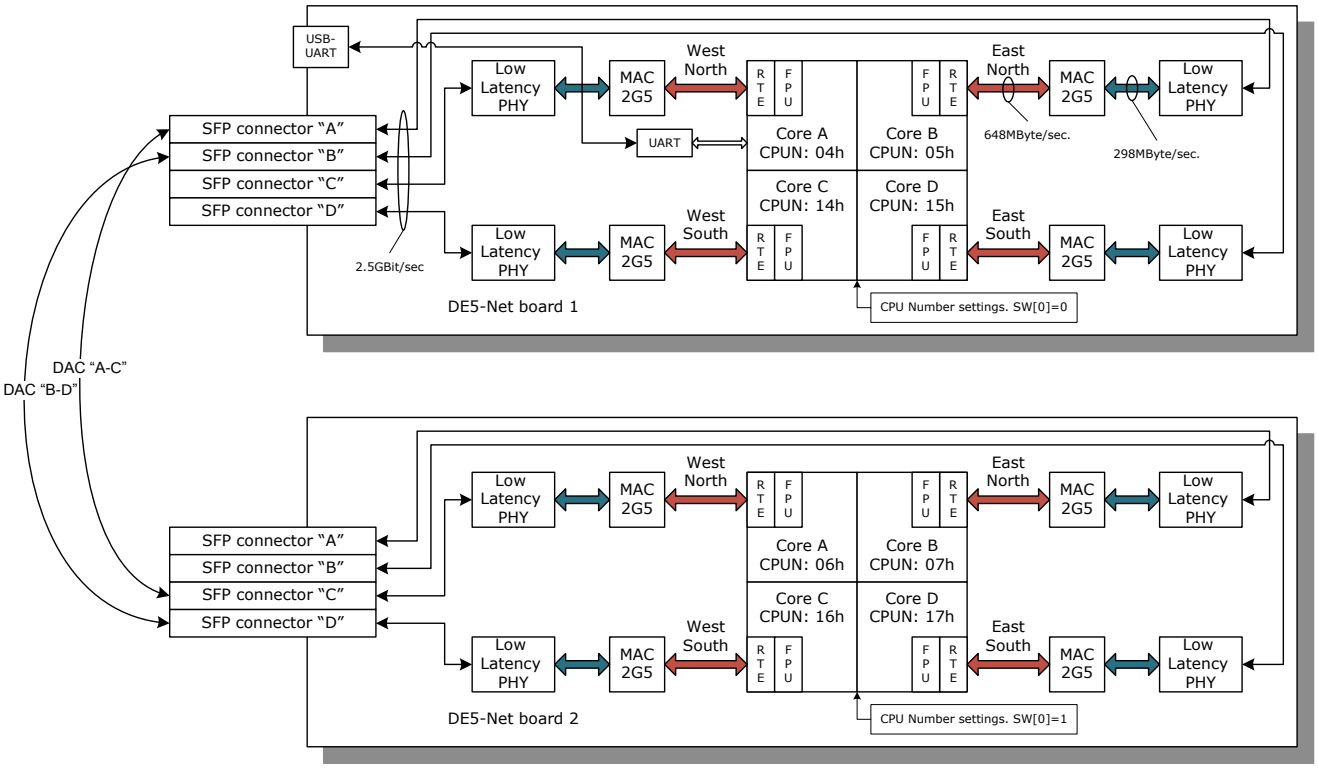
rrr – Row index upper bits
ccc – Column index upper bits

Each core in the cluster has its own local RAM. This memory is accessed at the same speed both for the owner core and for the neighboring 3 cores of the cluster. But if any core in the cluster accesses an object located in a core that does not belong to the cluster, then the network subsystem, consisting of a frame processing unit - FPU and a routing engine - RTE, comes into operation. The transaction is transmitted outside the cluster through one of 8 channels.

A 4-core cluster can communicate with other cores or clusters through 8 channels. 2 northern, 2 eastern, 2 southern and 2 western. The figure below shows the connection diagram of two 4-core clusters, carried out in the experiment.

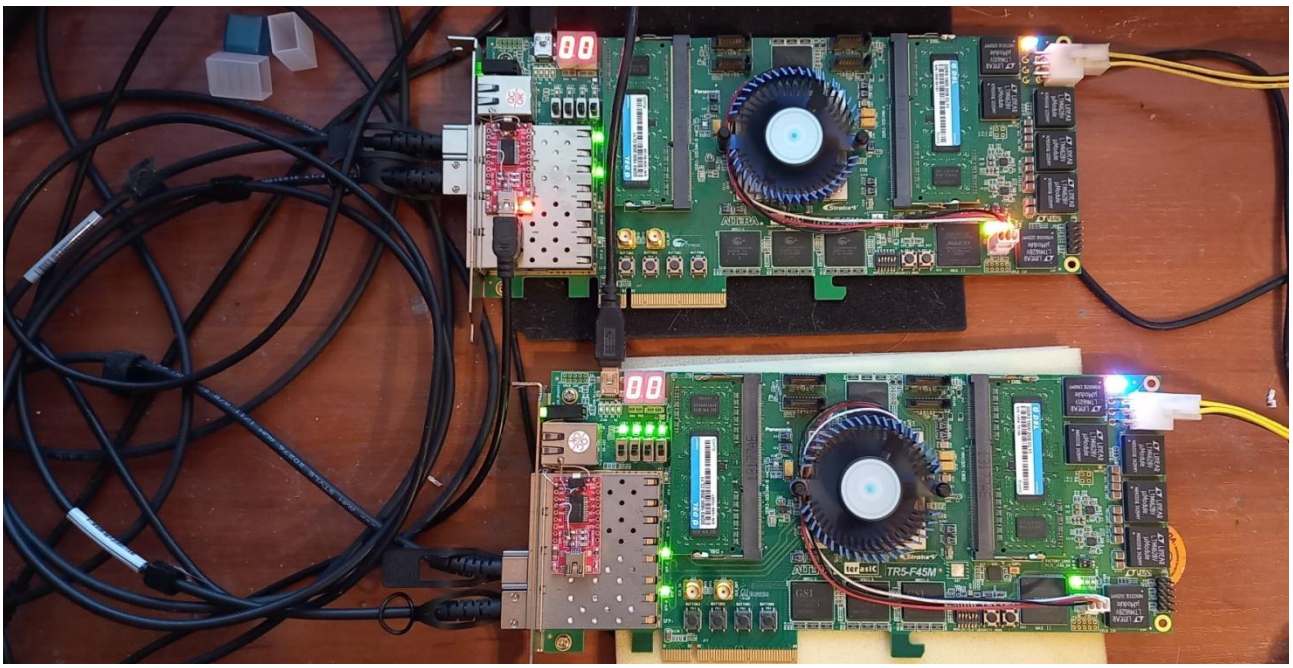


And below is the structure of the equipment assembled in the experiment, as well as a photo of the equipment.



Synchronized by internal processor clock 170MHz MAC<->RTE connection
STBO – data strobe from RTE to the MAC
NETDO[32:0] – data from RTE to the MAC (32 bit data and one bit of the first word flag)
STBI – data strobe from MAC to the RTE
NETDI[32:0] – data from MAC to the RTE (32 bit data and one bit of the first word flag)

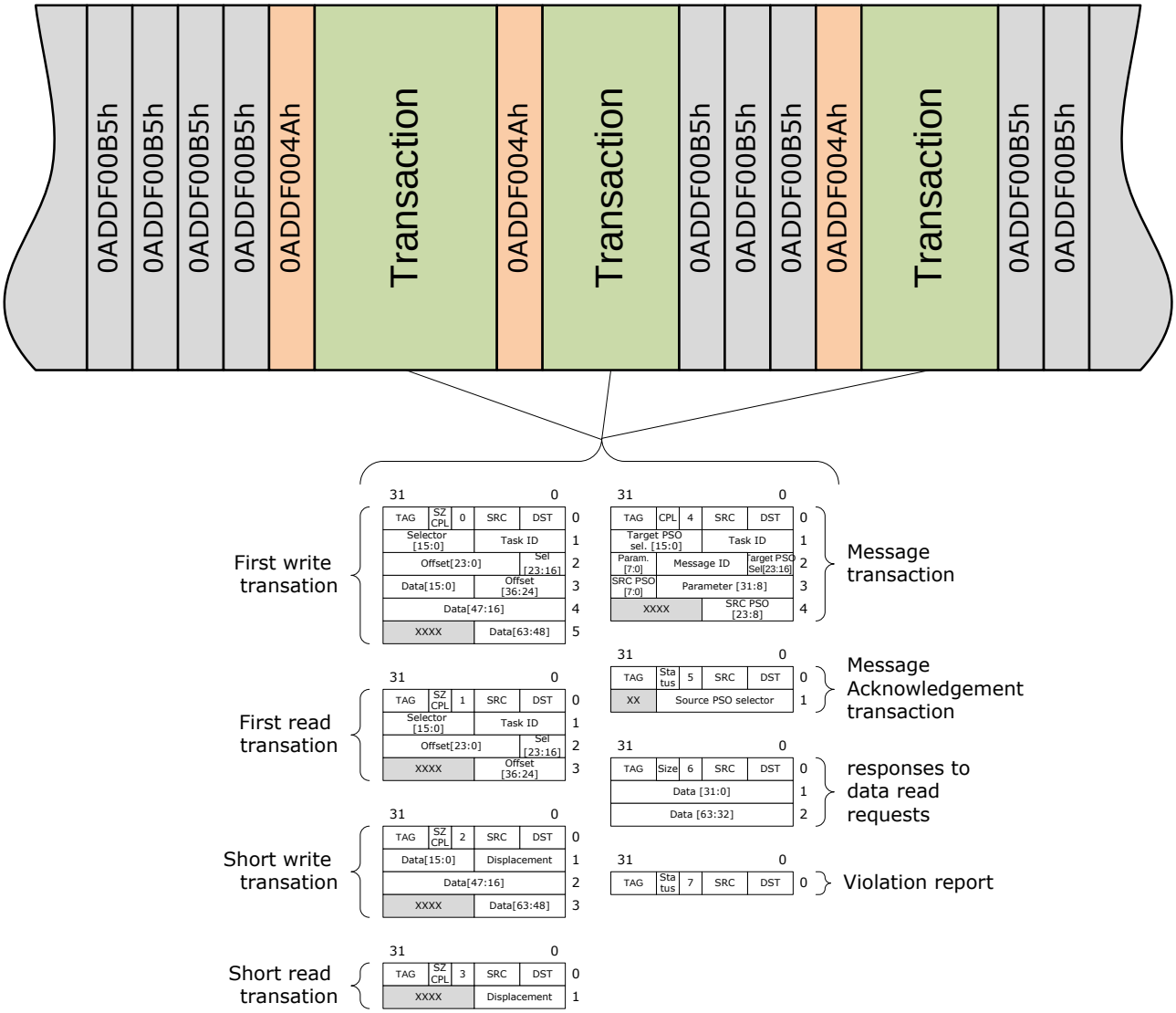
MAC TX channel:
TXCLK – transmit clock from HPY to the MAC (78.125MHz)
TXDATA[31:0] – transmit data from MAC to the PHY
MAC RX channel:
RXCLK – receive clock from PHY to the MAC (78.125MHz)
RXDATA[31:0] – receive data from PHY to the mAC



Fiber optic DAC (Direct Attach Cable) was chosen as the data transmission cable. Low Latency PHY v17.0 was used, configured for a transfer rate of 2.5Gb/sec. In fact, this PHY only used the PLL, serializer/deserializer, and transmit/receive queues.

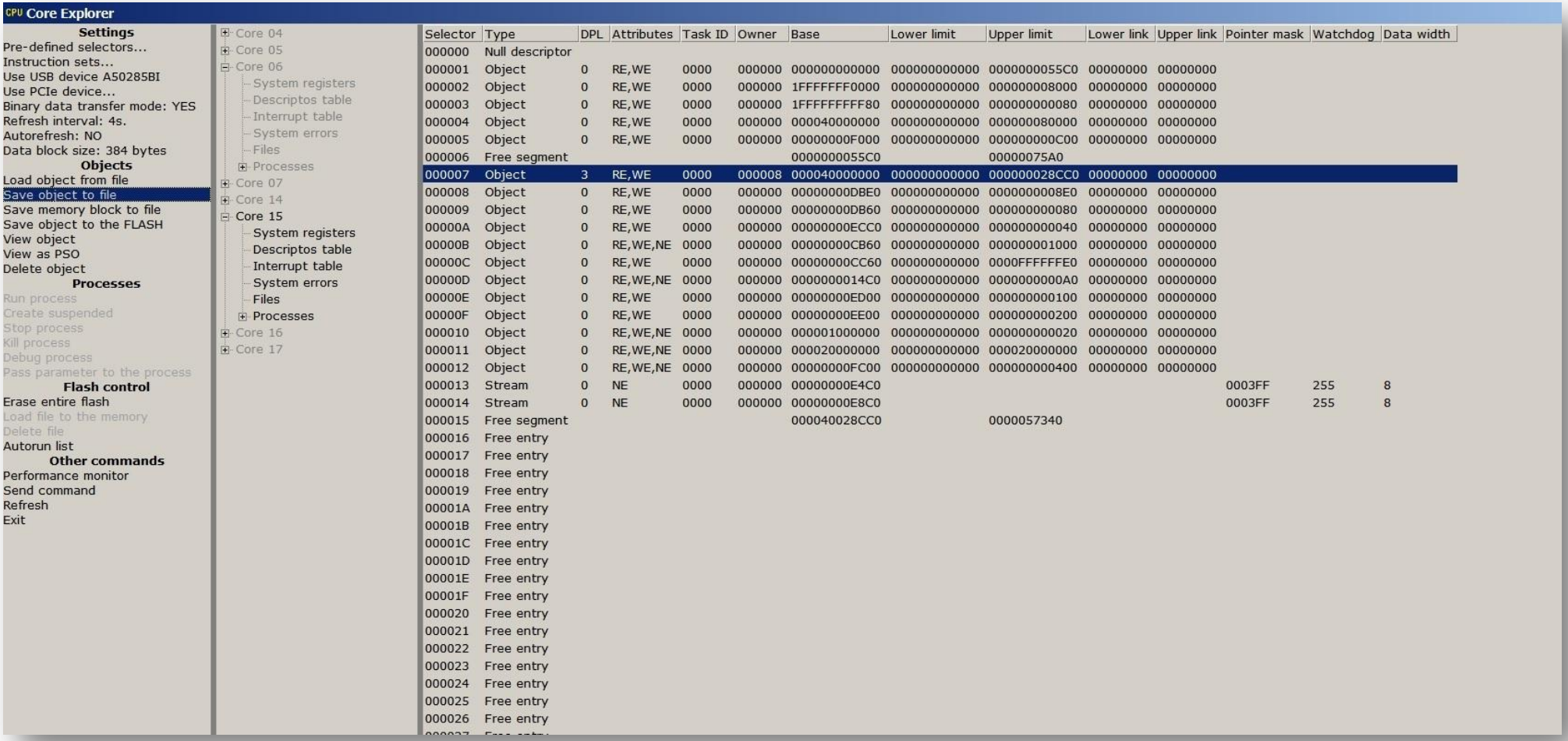
Each PHY is connected by 32-bit TxD and RxD data buses, as well as TxCLK and RxCLK clock signals to the MAC2G5 modules. These modules implement a protocol for transmitting and receiving inter-core transactions, align received data streams onto 32-bit word boundaries based on 32-bit synchronizing codes, generate these codes in transmitting channels, generate transaction start codes, and transmit transactions. Transactions contain control information, address information and data.

In IDLE state, MAC2G5 modules broadcast code 0ADDF00B5h. The receiver looks for this sequence in the input data stream. This code can be split into 2 32-bit words in the input stream received from the PHY. Low Latency PHY does not align data bits to byte boundaries, much less to 32-bit word boundaries. This function is implemented by MAC2G5. MAC2G5 searches for the position of the zero bit of the IDLE 0ADDF00B5h code and, after detecting this code, goes into transaction waiting mode. Any transaction begins with the sequence 0ADDF004Ah, as shown in the figure below.



The length of a transaction depends on its type. After receiving a known number of 32-bit words, the MAC expects either IDLE 0ADDF00B5h or the start of a new transaction code. If there is neither one nor the other code, then a loss of connection is detected and the controller begins to select a bit shift to detect the idle code in the incoming sequence of 32-bit words. Why are the codes 0ADDF00B5h and 0ADDF004Ah chosen for the idle and transaction start states? Because a zero byte at position [15:8] cannot be in the first word of a transaction of any type, since there cannot be a core with number 00h in the network and there cannot be such a transaction source.

After loading the configuration into the FPGA of both DE5-Net boards, CoreExplorer expectedly received a list of 8 cores detected in the multi-core system. An experiment was made to practically determine the time for reading data from the local memory of all cluster cores by a program executed in the 04h core.



Selector	Type	DPL	Attributes	Task ID	Owner	Base	Lower limit	Upper limit	Lower link	Upper link	Pointer mask	Watchdog	Data width
000000	Null descriptor												
000001	Object	0	RE,WE	0000	000000	000000000000	000000000000	0000000055C0	00000000	00000000			
000002	Object	0	RE,WE	0000	000000	1FFFFFFF0000	000000000000	000000008000	00000000	00000000			
000003	Object	0	RE,WE	0000	000000	1FFFFFFFFF80	000000000000	000000000080	00000000	00000000			
000004	Object	0	RE,WE	0000	000000	000040000000	000000000000	000000080000	00000000	00000000			
000005	Object	0	RE,WE	0000	000000	00000000F000	000000000000	000000000C00	00000000	00000000			
000006	Free segment					0000000055C0		00000075A0					
000007	Object	3	RE,WE	0000	000008	000040000000	000000000000	000000028CC0	00000000	00000000			
000008	Object	0	RE,WE	0000	000000	00000000DBE0	000000000000	0000000008E0	00000000	00000000			
000009	Object	0	RE,WE	0000	000000	00000000DB60	000000000000	000000000080	00000000	00000000			
00000A	Object	0	RE,WE	0000	000000	00000000ECC0	000000000000	000000000040	00000000	00000000			
00000B	Object	0	RE,WE,NE	0000	000000	00000000CB60	000000000000	000000001000	00000000	00000000			
00000C	Object	0	RE,WE	0000	000000	00000000CC60	000000000000	0000FFFFFFE0	00000000	00000000			
00000D	Object	0	RE,WE,NE	0000	000000	0000000014C0	000000000000	0000000000A0	00000000	00000000			
00000E	Object	0	RE,WE	0000	000000	00000000ED00	000000000000	000000000100	00000000	00000000			
00000F	Object	0	RE,WE	0000	000000	00000000EE00	000000000000	000000000200	00000000	00000000			
000010	Object	0	RE,WE,NE	0000	000000	000001000000	000000000000	000000000020	00000000	00000000			
000011	Object	0	RE,WE,NE	0000	000000	000020000000	000000000000	000020000000	00000000	00000000			
000012	Object	0	RE,WE,NE	0000	000000	00000000FC00	000000000000	000000000400	00000000	00000000			
000013	Stream	0	NE	0000	000000	00000000E4C0					0003FF	255	8
000014	Stream	0	NE	0000	000000	00000000E8C0					0003FF	255	8
000015	Free segment					000040028CC0		0000057340					
000016	Free entry												
000017	Free entry												
000018	Free entry												
000019	Free entry												
00001A	Free entry												
00001B	Free entry												
00001C	Free entry												
00001D	Free entry												
00001E	Free entry												
00001F	Free entry												
000020	Free entry												
000021	Free entry												
000022	Free entry												
000023	Free entry												
000024	Free entry												
000025	Free entry												
000026	Free entry												

It is necessary to pay attention to the fact that this is only an experiment in which data exchange between cores was carried out in the most inefficient, slowest way - by software transferring an array of 64-bit data from the memory of one core to the memory of another. In this case, scalar machine instructions were used. Some code to help you understand the process is given below.

```
28              amode           r6,4
29              amode           r8,2
30              li              r7,0
31              li              r9,IOSelector
32              li              r7,2                    ; 512 cycles
33              lar             ar7,r9
34              li              r10,10h                 ; timer offset
35              amode           r10,2
36              li              r0,10h
37              st              mar2:r8,r0              ; lock process switching
38              ldq             r14,mar3:r10            ; read timer before the cycle   R14
39  A2:
40              ldq             r0,mar0:r6
41              st              mar1:r6,r0
42              loop            r7,displacement A2
43  ;
44              ldq             r15,mar3:r10            ; read timer after the cycle    R15
45              li              r0,0F0h
46              st              mar2:r8,r0              ; unlock process switching
47  A3:
48              jnear           Displacement A3
49
```

In practice, of course, data exchange between cores must be based on block write transactions, block data transfer in one direction. A block read will be less efficient, but for a block read, the read requester must be able to issue requests without waiting for each individual data item to be received. In the code segment shown above, a new data transfer did not begin until a 64-bit word arrived in the R0 register and was subsequently written to local memory. This is an example of how not to program block transfers in practice.

Using the system timer, the time it took to transfer a block of 512 64-bit words was measured. The start time of the cycle was entered into register R14, and the end time into R15. The system timer contains a counter register synchronized by the main clock signal of the cores, the frequency of which in this experiment is 170 MHz.

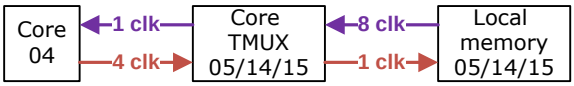
Data transfer paths and time of transferring the 1st 64-bit word.

Path 1: Local read/write.

Memory read total: 13 clk

Reading and writing are performed within the local memory of one core. The total cycle execution time is 7200 cycles of the main processor clock, 14 clock cycles per 1 “read-write-jump on the cycle counter”. 1 clock more than the calculated 13.

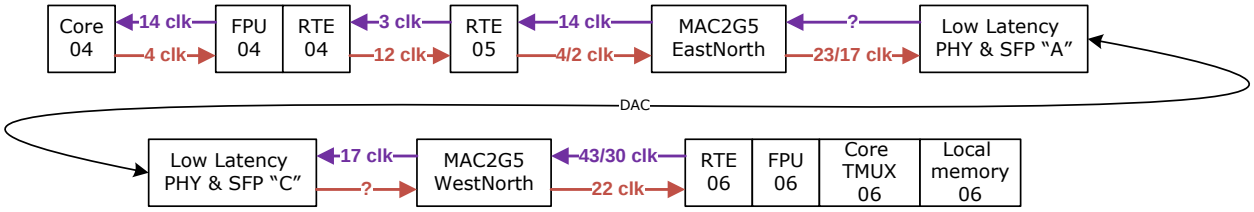
Path 2. Reading from the memory of a neighboring core located in the current cluster.



Memory read total: 14 clk

In this case, the read transaction was sent to the memory subsystem of the neighboring core through its transaction multiplexer. Data transfer was completed in 7700 clock cycles, 15 clock cycles per 64-bit word transfer.

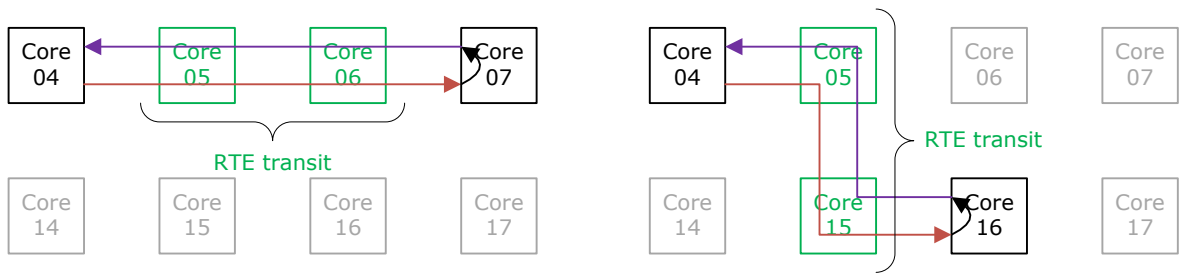
Path 3. Data is read from the local memory of the core located in a neighboring cluster. Shortest way.



Memory read total: 156/135+X clk (Long read form/short read form) Where X – delay value in the both PHY

A data request from core 04h comes to FPU/04h, from FPU to RTE/04h, broadcast to RTE/05h, sent to MAC2G5/05h, via Low Latency PHY/05h transmitted via DAC to Low Latency PHY/06h, RTE/06h, FPU/06h and goes to the transaction multiplexer of the 06h core, is processed by the memory subsystem of the 06h core and returns back through the same chain of modules to the 04h core. The above figure shows the transaction delays in all modules. The total execution time for transferring a data block was 92553 clock cycles, 180 clock cycles per 64-bit word. Calculations show that the minimum duration of a read transaction should be 135 clock cycles of a core operating at 170 MHz. A short read request will be transmitted through the DAC in 7 clock cycles, and the read data will be transmitted in 9 clock cycles. This corrects the numbers to 172 and 151. In the test, only the first transaction can be completed in 172 clock cycles, the remaining 511 in 151. Apparently, a minimum of 29 clock cycles are added on delays in the FIFO of both PHYs.

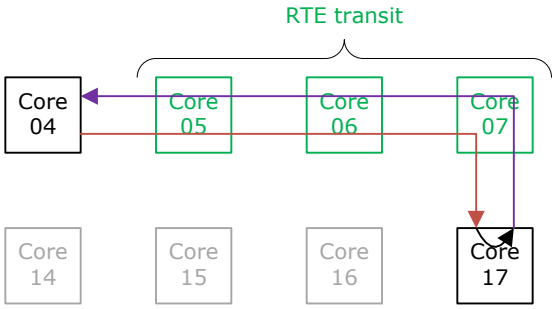
Path 4. Data is read from the core memory located in a neighboring cluster and double relaying to RTE is used to transfer transactions.



Reading data from the memory of core 07 or core 16. In the first version, transactions transit through the RTE of cores 05 and 06, and in the second, transactions transit through RTE 05 and 15. In such transactions, another 5 clock cycles are added, i.e. 151+5=156 cycles. The test showed that the system spent 185-188 clock cycles to

transmit one 64-bit word. And here again 185-156=29 cycles were apparently added as the transaction passed through the PHY 4 times. A 4 times pass is passing a read request through 2 PHYs, and then passing the data in the opposite direction, also through 2 PHYs.

Path 5: Reading data from core 17.



Another relay is being added via RTE. Transit RTEs become RTEs of cores 05, 06 and 07. The execution time of a short data read transaction will be 156+5=161 clock cycles. In the experiment, a block of 512 words was transmitted in 99,260 clock cycles, 194 clock cycles per word of data. The difference between the minimum calculated value and the experimental one is 33 cycles.

Conclusion.

The system showed stable performance. Files of different sizes were repeatedly transferred from the computer's hard drive to the memory of each core and read back, followed by comparison with the original file. The highest transfer rate that can be achieved in this network is possible in the case of block data writes and is determined by the transmission time of the short form of a 64-bit word write transaction over the connection between the MAC2G5 and the PHY. This connection operates at 78.125 MHz, which is more than 2 times lower than the operating frequency of the cores, FPU and RTE. The short form of a write transaction is transferred between the MAC2G5 and the Low Latency PHY in 4 TxCLK or RxCLK clock cycles. These 4 clock cycles include: transaction start code 0ADDF004Ah, 32 bits of the transaction control word and 2 32-bit data words. 19531250 64-bit words per second or 156.25MByte/sec. Theoretically, this can be increased by almost 4 times if you configure the Low Latency PHY transceiver at a speed of about 9.5G. It is also possible to use 2 or 4 transceivers in parallel for one inter-core connection, which will further increase the throughput, but that is another story.