	İTÜ **Electrical-Electronics Faculty**	**Electronics and Communication Engineering**

EHB 326E Introduction to Embedded Systems

2018-2019 Fall Semester

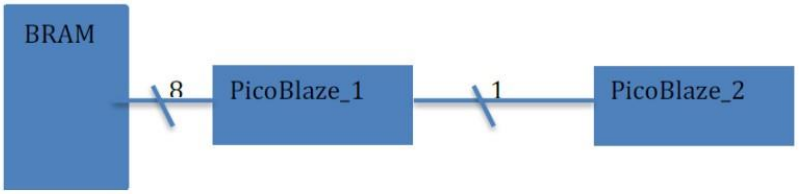
Final Project: Serial and Parallel Transmitting Problem

Prepared by: Mustafa Ghanim

Introduction

In this project, I am asked to program and simulate a digital system consists of initialized values- RAM of 64 Byte size and two Picoblaze processors. The asked problem is specified in the following figure:

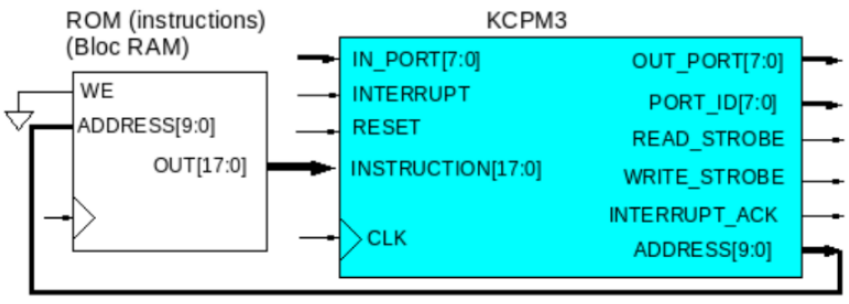
4. Design a system with two PicoBlazes and a single BlockRam. PicoBlaze_1 is directly connected to the BlockRAM. PicoBlaze_1 reads the memory (64 byte, form arbitrary address) and then sends the data to PicoBlaze_2 via 1 bit line.



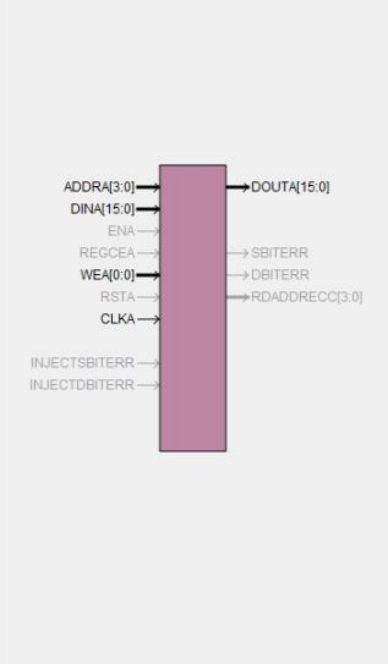
Such a system is used for understanding the principles of serial and parallel bit communication, it is expected that data coming from RAM transfers to the first Picoblaze in 8-bit form while same data will be transferred to the second Picoblaze bit by bit (serially). Since no specified information about what the second Picoblaze will perform for the coming data, I assume that it will gathers it again to 8-bit form and show the reconstructed bytes on its output port. The main idea is to decide which inputs and outputs of the processors are needed, and how to synchronize the data flow in order not to lose data. The general structure of Picoblaze can be shown in this figure:

Interfacing the PicoBlaze

The interfacing of the PicoBlaze can be seen in the figure beneath:

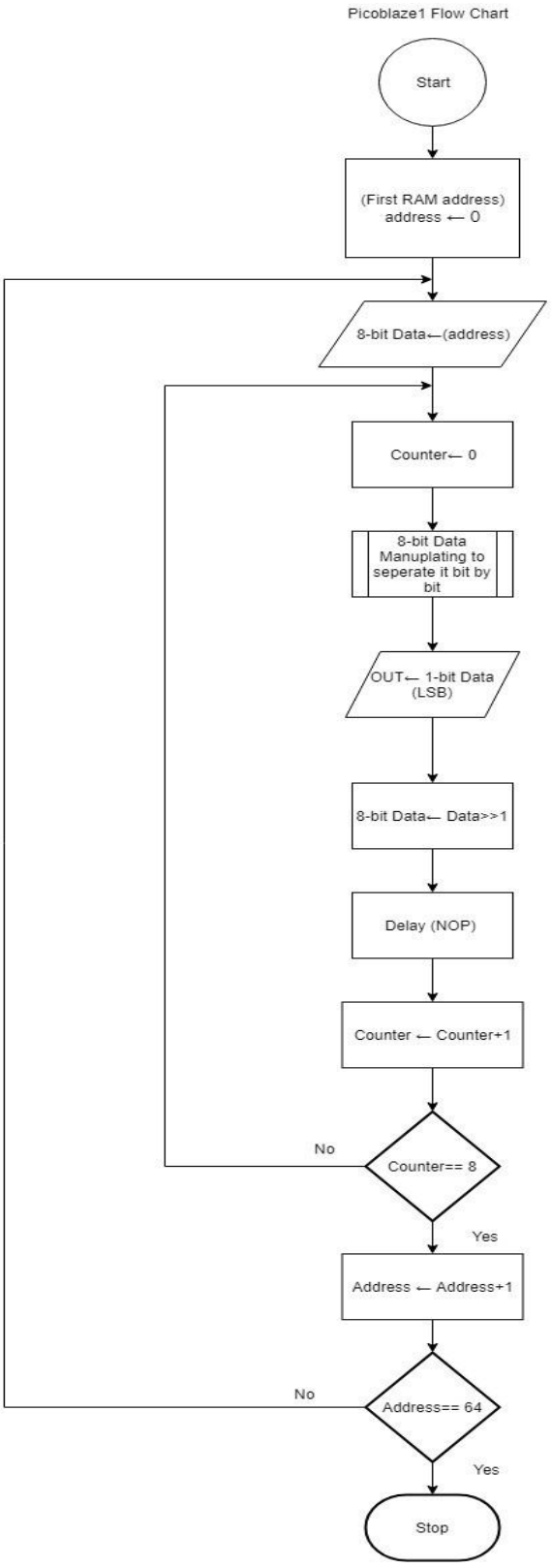


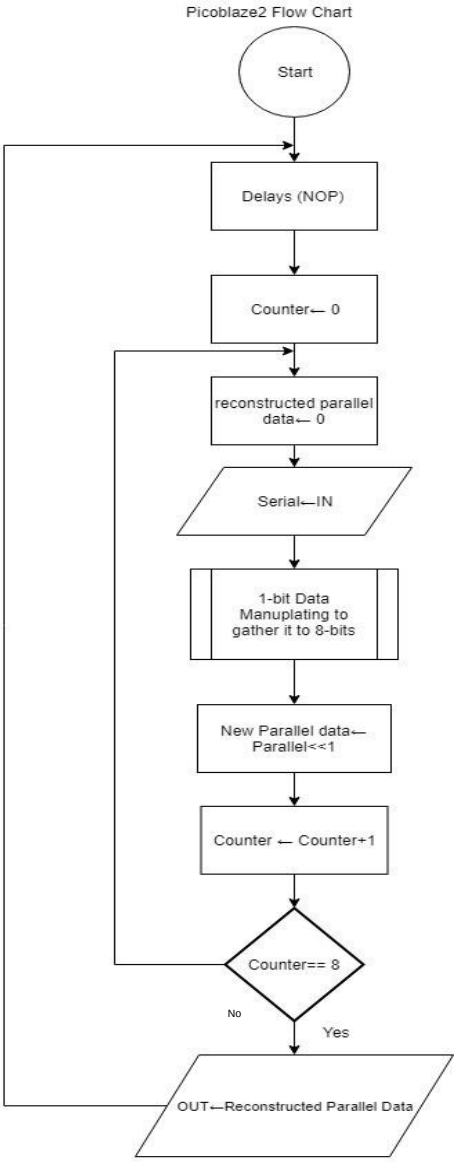
And the general block of typical generated RAM in ISE program can be shown as:



Since data on the RAM will be initialized by user, we will only read data from the RAM, so we are concern about only its input address and output data ports. The first Picoblaze ID_Output port will be connected to RAM's address port to maintain the address increment operation, and the DOUTA output port of the RAM will be connected to the input port of the first Picoblaze. As for the second Picoblaze, it deals only with the first Picoblaze directly. We will only its input and output ports and READ/WRITE_STROBE outputs if necessary.

Flow Charts of the Designed Algorithm:





Before Implementing this system directly on ISE environment, I have written two codes on Pico blaze simulator in order to observe the system's data flow. The difference between this simulation and the Verilog simulation, that this one allows the user to select the starting address, and the user has to enter the RAM data values manually. At the first Pico blaze code, the user can change the corresponding RAM data for each address if he/she does the simulation line by line, however a direct RUN command will drive all addresses with one RAM value. First picoblaze code is given as:

```
selectadr           DSIN      $00
selectadr_id        EQU       $00
selectadr_r         EQU       s0
adrout              DSOUT     $01
adrout_id           EQU       $01
adrout_r            EQU       s1
readfram            DSIN      $02
readfram_id         EQU       $02
readfram_r          EQU       s2
send                DSOUT     $03
send_id             EQU       $03
send_r              EQU       s3
counter_r           EQU       s4

main:

                    IN        selectadr_r, selectadr_id
                    COMP      selectadr_r, $40
                    JUMP      Z, stop
                    LOAD      adrout_r, selectadr_r
                    OUT       adrout_r, adrout_id
                    IN        readfram_r, readfram_id
                    JUMP      do

nextadr:            ADD       adrout_r, $01
                    COMP      adrout_r, $40

                    JUMP      Z, stop
                    OUT       adrout_r, adrout_id
                    IN        readfram_r, readfram_id
                    JUMP      do



                    LOAD      counter_r, $00
                    IN        readfram_r, readfram_id

do:

                    LOAD      send_r, $01
                    ADD       counter_r, $01
                    AND       send_r, readfram_r
                    RR        readfram_r
                    COMP      counter_r, $08
                    OUT       send_r, send_id
                    JUMP      NZ, do
                    JUMP      nextadr


stop:               JUMP      stop
```

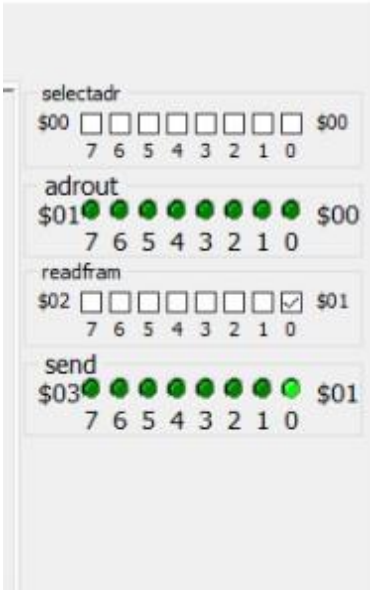
And the second Picoblaze code is given as:

```
gelen               DSIN      $00
gelen_id            EQU       $00
gelen_r             EQU       s0
cikis               DSOUT     $01
cikis_id            EQU       $01
cikis_r             EQU       s1
temp                EQU       s2
bitcounter          EQU       s3
basla:
                    LOAD      bitcounter, $00
temel:
                    ADD       bitcounter, $01
                    IN        gelen_r, gelen_id
                    LOAD      temp, gelen_r
                    RR        temp
                    OR        cikis_r, temp

                    OUT       cikis_r, cikis_id
                    COMP      bitcounter, $08
                    JUMP      Z, yenidenbasla
                    SR0       cikis_r

                    JUMP      temel
yenidenbasla:
                    LOAD      bitcounter, $00
                    LOAD      cikis_r, $00
                    OUT       cikis_r, cikis_id
                    JUMP      basla
stop:               JUMP      stop
```

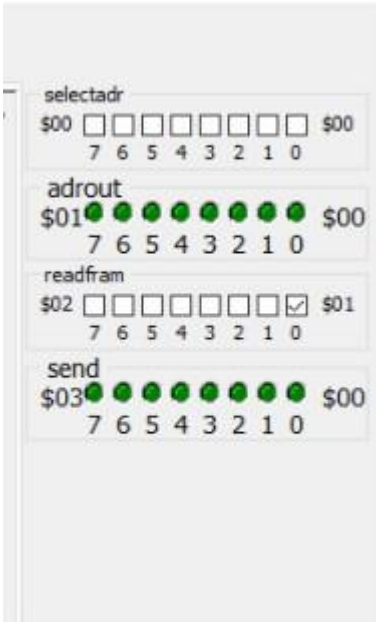
To see how it works, let us assume that we select the first address (00h) and increase the RAM data corresponding to each address by one starting from 1.

For the first Pico blaze:

At address (00h) and RAM value of (1) we have such output at the first bit (LSM) :



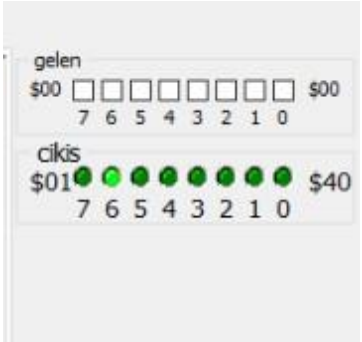
The code loop related to bit transmitting data in serial will repeat more 7 times giving zero output as follows for the same RAM address:



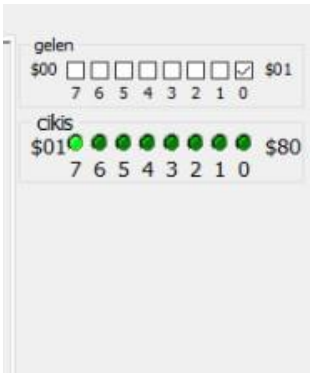
However, if our data for that address is not (1) but say (2), the second cycle output would be 1 and, the first cycle, and other 6 cycles outputs would be zero. Thus, serial transmitting is achieved.

The second Picoblaze needs to manipulate serial data for 8 cycles to be able to form the reconstructed parallel data. Since we are sending serial bits through LSB, we would do some manipulations to reform the correct places of them, such manipulations are explained in the code. For second Picoblaze input and output, assume that we are receiving this sequence on the LSB line; (1 then 0 then 0 then 1 then 0 then 1 then 0 then 0) we would expect to see such an output after 8 cycles in the loop (0 then 0 then 1 then 0 then 1 then 0 then 0 then 1). Let's try:

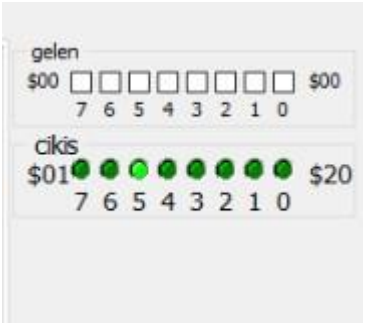
First cycle :



second cycle:

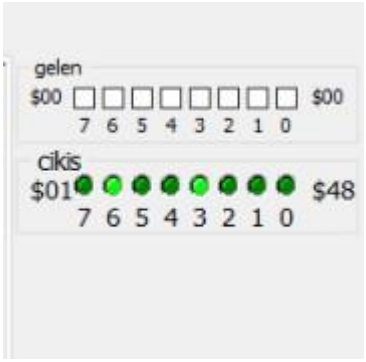


Third cycle:

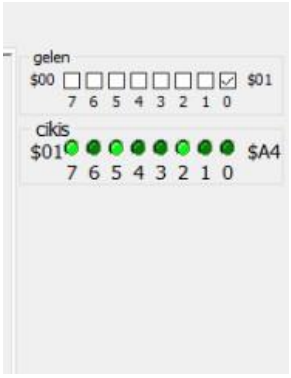


Fourth cycle:

Fifth cycle:

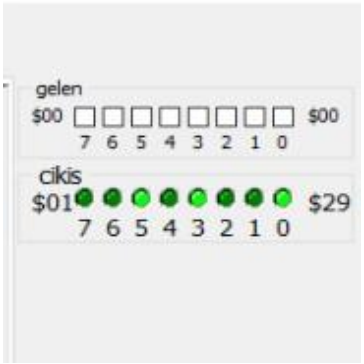


sixth cycle:



Seventh cycle:

Last cycle for one byte



We see that our last cycle represents our desired and correct output. This can be applied to any RAM data coming from any RAM address as well.

ISE Digital Analysis and Simulation:

To simplify the logic design of the system. The starting address has been chosen to be fixed at 00h. To achieve the synchronization goal between Pico1 and Pico2, the number of instructions in each Picoblaze has been calculated considering the jumps too, if they are not equal, we add delays (NOBs) accordingly.

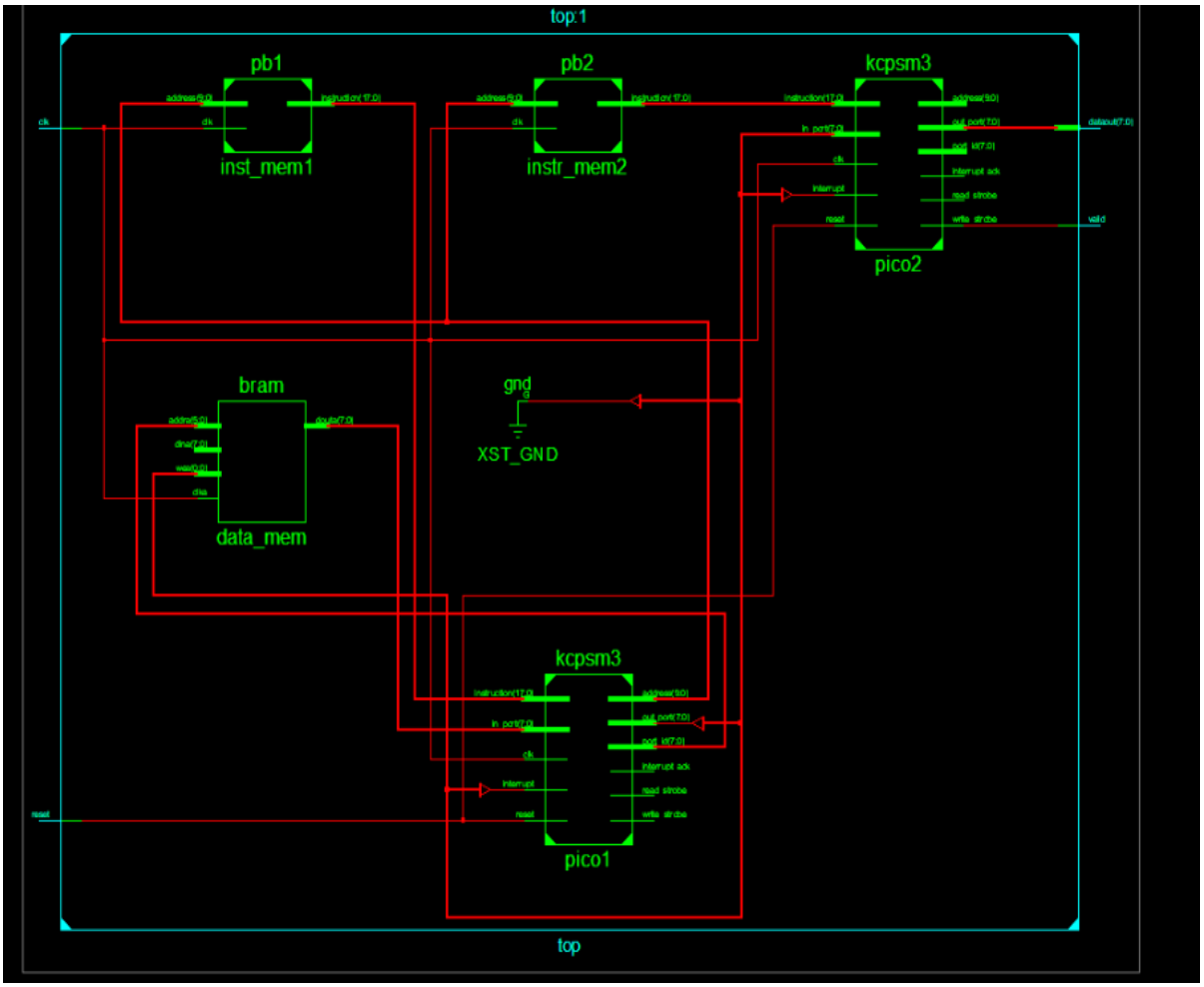
KCPSM3 code of Pico1 to be compiled to Verilog module:

```
CONSTANT  START_ADDRESS, 00
CONSTANT  END_ADDRESS, 40
NAMEREG s0,address
NAMEREG s1,data
NAMEREG s2,counter
NAMEREG s3,send

start:
          LOAD  counter,00
          LOAD address, START_ADDRESS
loop:
          INPUT     data, (address)
          COMPARE address,END_ADDRESS
          JUMP NZ, do


          JUMP stop




do:

    LOAD      send,send
    LOAD       send, 01
    AND        send, data
    OUTPUT     send, (counter)

    RR         data

    ADD        counter, 01
    COMPARE    counter,08
    JUMP       NZ, do

    ADD       address,01
    LOAD      counter,00
    JUMP       loop
stop:

JUMP stop
```

KCPSM3 code of Pico2 to be compiled to Verilog module:

```
NAMEREG s0, gelen_r
NAMEREG s1,cikis_r
NAMEREG s2,temp
NAMEREG s3,bitcounter

delay:

                    LOAD      gelen_r,gelen_r
                    LOAD      gelen_r,gelen_r
                    LOAD      gelen_r,gelen_r
                    LOAD      gelen_r,gelen_r
                    LOAD      gelen_r,gelen_r
                    LOAD      gelen_r,gelen_r
                    LOAD      gelen_r,gelen_r


basla:
                    LOAD      gelen_r,gelen_r

                    LOAD      bitcounter, 00
temel:
                    INPUT     gelen_r, (bitcounter)
                   LOAD      temp, gelen_r
                    RR        temp
                    OR        cikis_r, temp
                   SR0       cikis_r
                    ADD       bitcounter, 01
                    COMPARE   bitcounter, 08
                    JUMP      Z, yenidenbasla



                    JUMP      temel
yenidenbasla:
                    OUTPUT    cikis_r, (bitcounter)


                    LOAD      bitcounter, 00
                    LOAD      cikis_r, 00
                    JUMP      basla
```

RTL Schematic level of the system:



Top module code:

```
module top(
                input clk, reset,
                output [7:0] dataout,
                output valid
    );

        wire    [9:0]   pico1_address, pico2_address  ;
        wire    [17:0]pico1_instruction , pico2_instruction ;
        wire  [7:0]     pico1_port_id, pico2_port_id;
        wire  [7:0] pico1_out_port, pico1_in_port, pico2_in_port ;
        wire pico1_read_strobe;
        wire pico1_write_strobe;
        wire pico2_read_strobe;

        wire line;

        assign line = pico1_out_port[0];
        assign pico2_in_port = {7'b0000000, line};
```

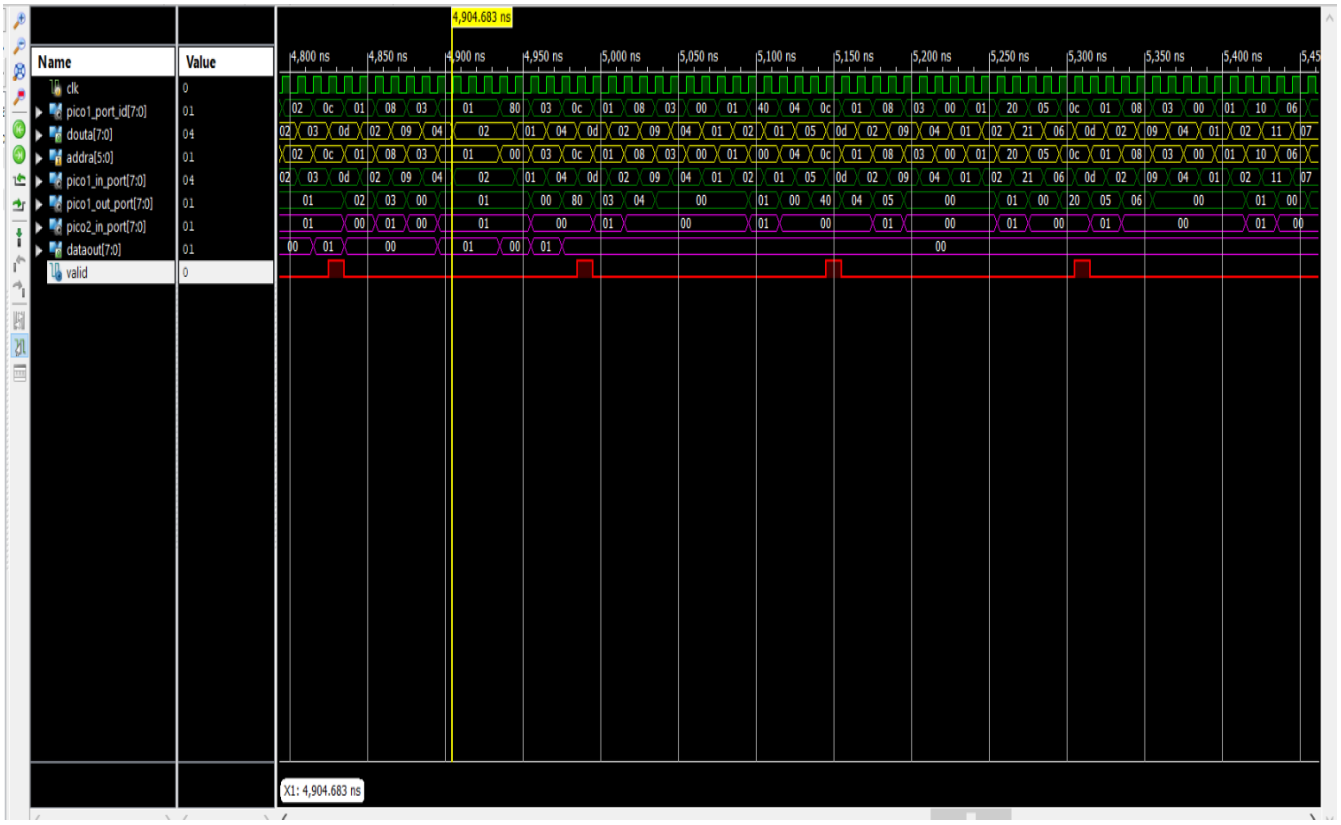
```
        // Instantiate pico
        kcpsm3 pico1 (
                 .address(pico1_address),
                 .instruction(pico1_instruction),
                 .read_strobe(pico1_read_strobe),
                 .write_strobe(pico1_write_strobe),
                 .port_id(pico1_port_id),
                 .out_port(pico1_out_port),
                 .in_port(pico1_in_port),
                 .interrupt(0),
                 .reset(reset),
                 .clk(clk)
                 );

         kcpsm3 pico2 (
                 .port_id(pico2_port_id),
                 .address(pico2_address),
                 .instruction(pico2_instruction),
                 .write_strobe(valid),
.read_strobe(pico2_read_strobe),
                 .out_port(dataout),
                 .in_port(pico2_in_port),
                 .interrupt(0),
                 .reset(reset),
                 .clk(clk)
                 );

        pb1 inst_mem1 (
                 .address(pico1_address),
                 .instruction(pico1_instruction),
                 .clk(clk)
                 );


        pb2 instr_mem2 (
                 .address(pico1_address),
                 .instruction(pico2_instruction),
                 .clk(clk)
                 );

        bram data_mem (
          .clka(clk), // input clka
          .wea(0), // input wea
          .addra(pico1_port_id), // input [5 : 0] addra
          .douta(pico1_in_port) // output [7 : 0] douta
        );

endmodule
```

Testbench code:

```
module test;

reg clk, reset;
wire [7:0] dataout;
wire valid;

top instance_name (
    .clk(clk),
    .reset(reset),
    .dataout(dataout),
    .valid(valid)
    );



always begin
        clk = ~clk;
        #5;
end

initial begin
        clk = 0;
        reset = 1;
        #20;
        reset = 0;
end
```

Although that I have connected processors, and RAM wires correctly (with no problem in synthesis behavior) unfortunately, I could not get the expected output data values in the simulation due to a questionable reason. With no large experience in Verilog language, it is very difficult for me to connect all the system with the programming parts in synchronized way in the given short period. My code is working fine in Picoblaze Simulation environment, that makes me to suspect a problem either in synchronization or logic design. I have tried to set the acceptable output only when Valid is HIGH, however, this did not work too (output data is only 0 or 1 at Valid HIGH, which may say that reconstruction operation from serial data has failed). One other solution is, do not make input data and output data in real-time, transfer all inputs to the RAM of the second Picoblaze, then operate them after some delay. Using MUX and DEMUX techniques which I am not experienced in, can be useful for organizing data flow correctly, READ_STROBE and WRITE_STROBE of the first Picoblaze can be benefited too. One another reason of the failure of the simulation can be not defining the sensitivity of transitions according to the rising edge of clocks.

Simulation Plots:



File used for initializing the RAM (ceo format):

```
MEMORY_INITIALIZATION_RADIX=10;

MEMORY_INITIALIZATION_VECTOR=

1,
2,
3,
4,
5,
6,
7,
8,
9,
10,
11,
12,
13,
14,
15,
16,
17,
18,
19,
20,
21,
22,
23,
24,
25,
26,
27,
28,
29,
30,
31,
32,
33,
34,
35,
36,
37,
38,
39,
40,
41,
42,
43,
44,
45,
46,
47,
48,
49,
50,
51,
52,
53,
54,
55,
56,
57,
58,
59,
60,
61,
62,
63,
64,
```