Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра ЕОМ

Звіт

до лабораторної роботи № 5

з дисципліни «Дослідження програмної моделі RISC CPU»

Виконав:

ст.гр. КІ-38

Хомин В.Б

Прийняв:

Старший викладач

Козак Н. Б.

Львів 2022

Мета роботи: Навчитися здійснювати оцінку структури об’єкта (RISC CPU) на існуючій програмній моделі. Навчитись встановлювати структуру інтерфейсів об’єкта.

ХІД ВИКОНАННЯ РОБОТИ:

```
#include "directive.h"
#include "systemc.h"
#include "bios.h"
#include "paging.h"
#include "icache.h"
#include "fetch.h"
#include "decode.h"
#include "exec.h"
#include "mmxu.h"
#include "floating.h"
#include "dcache.h"
#include "pic.h"
#include <climits>
#include <cstdlib>
#include <time.h>
//#include <sys/times.h>
#include <limits.h>

int sc_main(int ac, char *av[])
{

 //            ************************
ICACHE
**********************************
*
 // ICACHE = ram_cs
 // ICACHE = ram_we
 // ICACHE = addr
 // ICACHE = ram_datain
 // ICACHE = ram_dataout
 // ICACHE = ld_valid = pid_valid
 // ICACHE = ld_data = pid_data
 sc_signal<bool>
icache_valid("ICACHE_VALID") ;

 //  ************************ BIOS
**********************************
*
 sc_signal<bool>     ram_cs("RAM_CS")
;
 sc_signal<bool>
ram_we("RAM_WE") ;
 sc_signal<unsigned >   addr("Address") ;
 sc_signal<unsigned                    >
ram_datain("RAM_DATAIN") ;
 sc_signal<unsigned                    >
ram_dataout("RAM_DATAOUT") ;
 sc_signal<bool>
bios_valid("BIOS_VALID") ;
 const int delay_cycles = 2;

 //  ************************ Paging
**********************************
*
 // Paging paging_din = ram_datain
 // Paging paging_csin = ram_cs
 // Paging paging_wein = ram_we
 // Paging logical_address = addr
 sc_signal<unsigned                    >
icache_din("ICACHE_DIN") ;
 sc_signal<bool>
icache_validin("ICACHE_VALIDIN") ;
 sc_signal<bool>
icache_stall("ICACHE_STALL") ;
 sc_signal<unsigned                    >
paging_dout("PAGING_DOUT") ;
 sc_signal<bool>
paging_csout("PAGING_CSOUT") ;
 sc_signal<bool>
paging_weout("PAGING_WEOUT") ;
 sc_signal<unsigned                    >
physical_address("PHYSICAL_ADDRES
S") ;
 // Paging dataout  = ram_dataout
 // Paging data_valid = icache_valid
 // Paging stall_ifu = stall_fetch



 //  ************************ Fetch
**********************************
*
 // IFU ramdata = ram_dataout
 sc_signal<unsigned                    >
branch_target_address("BRANCH_TAR
GET_ADDRESS") ;
 sc_signal<bool>
next_pc("NEXT_PC") ;
 sc_signal<bool>
branch_valid("BRANCH_VALID") ;
 sc_signal<bool>
stall_fetch("STALL_FETCH") ;
 sc_signal<bool>
pred_fetch("PRED_FETCH") ;
 // IFU ram_valid = bios_valid
 // IFU ram_cs = ram_cs
 // IFU ram_we = ram_we
 // IFU address = addr
 // IFU smc_instrction = ram_datain
 //      IFU      pred_branch_address      =
pred_branch_address
 //      IFU      pred_branch_valid      =
pred_branch_valid
 sc_signal<unsigned>
instruction("INSTRUCTION") ;
 sc_signal<bool>
instruction_valid("INSTRUCTION_VALI
D") ;
 sc_signal<unsigned                    >
program_counter("PROGRAM_COUNT
ER") ;
 sc_signal<bool>
branch_clear("BRANCH_CLEAR") ;
 sc_signal<bool>
pred_fetch_valid("PRED_FETCH_VALI
D") ;
 sc_signal<bool>     reset("RESET") ;

 //  ************************ Branch
**********************************
*
 // BPU: fetch_inst = instruction
 // BPU: fetch_pc = program_counter
 // BPU: fetch_valid = instruction_valid
 //      BPU:      branch_inst_addr      =
branch_instruction_address
 //     BPU:     branch_target_address     =
branch_target_address
 // BPU: branch_valid = branch_valid
 sc_signal<unsigned                    >
          pred_branch_address("PRED_
BRANCH_ADDRESS");
 sc_signal<bool>
pred_branch_valid("PRED_BRANCH_V
ALID") ;
 sc_signal<bool>
pred_tellid("PRED_TELLID") ;
 sc_signal<unsigned>
pred_instruction("PRED_INSTRUCTION
") ;
 sc_signal<bool>
pred_inst_valid("PRED_INST_VALID") ;
```

```
 sc_signal<unsigned                    >
            pred_inst_pc("PRED_INST_P
C");
 // ********************** Decode
*********************************
*
 // ID instruction = instruction
 // ID instruction = instruction_valid
 // ID destreg_write = out_valid
 // ID destreg_write_src = destout
 // ID clear_branch    = branch_clear
 // ID pc = program_counter
 sc_signal<bool>
pred_on("PRED_ON") ;
 sc_signal<unsigned                    >
            branch_instruction_address("B
R_INSTRUCTION_ADDRESS");
 sc_signal<signed>
dram_dataout("DRAM_DATAOUT") ;
 sc_signal<bool>
dram_rd_valid("DRAM_RD_VALID") ;
 sc_signal<unsigned>
dram_write_src("DRAM_WRITE_SRC")
;
 sc_signal<bool>
mem_access("MEM_ACCESS") ;
 sc_signal<unsigned                    >
mem_address("MEM_ADDRESS") ;
 sc_signal<int>       alu_op("ALU_OP") ;
 sc_signal<bool>
mem_write("MEM_WRITE") ;
 sc_signal<unsigned>
alu_src("ALU_SRC") ;
 sc_signal<bool>
reg_write("REG_WRITE") ;
 sc_signal<signed int> src_A("SRC_A") ;
 sc_signal<signed int> src_B("SRC_B") ;
 sc_signal<bool>
forward_A("FORWARD_A") ;
 sc_signal<bool>
forward_B("FORWARD_B") ;
 // ID stall_fetch = stall_fetch
 sc_signal<bool>
decode_valid("DECODE_VALID") ;
 sc_signal<bool>
float_valid("FLOAT_VALID") ;
 sc_signal<bool>
mmx_valid("MMX_VALID") ;
 sc_signal<bool>
pid_valid("PID_VALID") ;
 sc_signal<signed>
pid_data("PID_DATA") ;

 //            **********************
DCACHE
*********************************
*
 sc_signal<signed>
mmic_datain("MMIC_DATAIN")          ;
            /* DCU: datain          */
 sc_signal<unsigned>
mmic_statein("MMIC_STATEIN")      ;/*
DCU: statein */
 sc_signal<bool>
mmic_cs("MMIC_CS") ;
            /* DCU: cs              */
 sc_signal<bool>
mmic_we("MMIC_WE") ;
            /* DCU: we              */
 sc_signal<unsigned                    >
mmic_addr("MMIC_ADDR")  ; /*  DCU:
addr          */
 sc_signal<unsigned>
mmic_dest("MMIC_DEST") ;    /*  DCU:
dest          */
 sc_signal<unsigned>
mmic_destout("MMIC_DESTOUT")    ;/*
DCU: destout */
 sc_signal<signed>
mmic_dataout("MMIC_DATAOUT")   ;/*
DCU: dataout */
 sc_signal<bool>
mmic_out_valid("MMIC_OUT_VALID")
;/* DCU: out_valid*/
 sc_signal<unsigned>
mmic_stateout("MMIC_STATEOUT") ;/*
DCU: stateout */

 // ********************** Execute
*********************************
*
 // EXEC in_valid = decode_valid
 sc_signal<bool>
            in_valid("IN_VALID") ;
 // EXEC opcode = alu_op
 sc_signal<bool>      negate("NEGATE")
;
 sc_signal<int>       add1("ADD1") ;
 sc_signal<bool>
            shift_sel("SHIFT_SEL") ;
 // EXEC dina = src_A
 // EXEC dinb = src_B
 // EXEC dest = alu_src
 sc_signal<bool>                 c("C") ;
 sc_signal<bool>                 v("V") ;
 sc_signal<bool>                 z("Z") ;
 sc_signal<signed>
            dout("DOUT") ;
 sc_signal<bool>
            out_valid("OUTPUT_VALID"
) ;
 sc_signal<unsigned>
            destout("DESTOUT") ;

 // ********************** Floating
point
*****************************
 // FPU in_valid = float_valid
 // FPU opcode = alu_op
 // FPU floata = src_A
 // FPU floatb = src_B
 // FPU dest = alu_src
 sc_signal<signed>
            fdout("FDOUT") ;
 sc_signal<bool>
            fout_valid("FOUT_VALID") ;
 sc_signal<unsigned>
            fdestout("FDESTOUT") ;

 //   ********************** PIC
*********************************
*******
 sc_signal<bool>
            ireq0("IREQ0") ;
 sc_signal<bool>
            ireq1("IREQ1") ;
 sc_signal<bool>
            ireq2("IREQ2") ;
 sc_signal<bool>
            ireq3("IREQ3") ;
 // PIC cs = interrupt_ack
 // PIC intack_cpu = interrupt_ack
 sc_signal<bool>
            rd_wr("RD_WR") ;
 sc_signal<bool>
            intreq("INTREQ") ;
 sc_signal<unsigned>
            vectno("VECTNO") ;
 sc_signal<bool>
            intack("INTACK") ;
 sc_signal<bool>
            intack_cpu("INTACK_CPU") ;

 //   ********************** MMX
*********************************
*
 // MMX mmx_valid = mmx_valid
 // MMX opcode = alu_op
 // MMX mmxa = src_A
 // MMX mmxb = src_B
 // MMX dest = dest
 // MMX mmxdout = fdout
 // MMX mmxout_valid = fpu_valid
```

```
 // MMX mmxdestout = fpu_destout

 //  ********************** DSP
**********************************
*******
 sc_signal<int>
          dsp_in1("DPS_IN1");
 sc_signal<int>
          dsp_out1("DSP_OUT1");
 sc_signal<bool>
          dsp_data_valid("DSP_DATA_
VALID");
 sc_signal<bool>
          dsp_input_valid("DSP_INPUT
_VALID");
 sc_signal<bool>
          dsp_data_requested("DSP_DA
TA_REQUESTED");



 sc_trace_file          *tf          =
sc_create_vcd_trace_file("signals");
 ((vcd_trace_file*)tf)-
>sc_set_vcd_time_unit(0);
 sc_trace(tf,                icache_valid,
"ICACHE_VALID");
 sc_trace(tf, ram_cs, "RAM_CS");
 sc_trace(tf, ram_we, "RAM_WE");
 sc_trace(tf, addr, "Address");
 sc_trace(tf,                ram_datain,
"RAM_DATAIN");
 sc_trace(tf,                ram_dataout,
"RAM_DATAOUT");
 sc_trace(tf, bios_valid, "BIOS_VALID");
 sc_trace(tf,                icache_din,
"ICACHE_DIN");
 sc_trace(tf,              icache_validin,
"ICACHE_VALIDIN");
 sc_trace(tf,                icache_stall,
"ICACHE_STALL");
 sc_trace(tf,                paging_dout,
"PAGING_DOUT");
 sc_trace(tf,                paging_csout,
"PAGING_CSOUT");
 sc_trace(tf,                paging_weout,
"PAGING_WEOUT");
 sc_trace(tf,       branch_target_address,
"BRANCH_TARGET_ADDRESS");
 sc_trace(tf, next_pc, "NEXT_PC");
 sc_trace(tf,                branch_valid,
"BRANCH_VALID");
 sc_trace(tf,                stall_fetch,
"STALL_FETCH");
 sc_trace(tf,                pred_fetch,
"PRED_FETCH");
 sc_trace(tf,                instruction,
"INSTRUCTION");
 sc_trace(tf,                instruction_valid,
"INSTRUCTION_VALID");
 sc_trace(tf,                program_counter,
"PROGRAM_COUNTER");
 sc_trace(tf,                branch_clear,
"BRANCH_CLEAR");
 sc_trace(tf,                pred_fetch_valid,
"PRED_FETCH_VALID");
 sc_trace(tf, reset, "RESET");
 sc_trace(tf,          pred_branch_address,
"PRED_BRANCH_ADDRESS");
 sc_trace(tf,                pred_branch_valid,
"PRED_BRANCH_VALID");
 sc_trace(tf,                pred_tellid,
"PRED_TELLID");
 sc_trace(tf,                pred_instruction,
"PRED_INSTRUCTION");
 sc_trace(tf,                pred_inst_valid,
"PRED_INST_VALID");
 sc_trace(tf,                pred_inst_pc,
"PRED_INST_PC");
 sc_trace(tf, pred_on, "PRED_ON");
 sc_trace(tf,   branch_instruction_address,
"BR_INSTRUCTION_ADDRESS");
 sc_trace(tf,                dram_dataout,
"DRAM_DATAOUT");
 sc_trace(tf,                dram_rd_valid,
"DRAM_RD_VALID");
 sc_trace(tf,                dram_write_src,
"DRAM_WRITE_SRC");
 sc_trace(tf,                mem_access,
"MEM_ACCESS");
 sc_trace(tf,                mem_address,
"MEM_ADDRESS");
 sc_trace(tf, alu_op, "ALU_OP");
 sc_trace(tf,                mem_write,
"MEM_WRITE");
 sc_trace(tf, alu_src, "ALU_SRC");
 sc_trace(tf, reg_write, "REG_WRITE");
 sc_trace(tf, src_A, "SRC_A");
 sc_trace(tf, src_B, "SRC_B");
 sc_trace(tf,                forward_A,
"FORWARD_A");
 sc_trace(tf,                forward_B,
"FORWARD_B");
 sc_trace(tf,                decode_valid,
"DECODE_VALID");
 sc_trace(tf,                float_valid,
"FLOAT_VALID");
 sc_trace(tf,                mmx_valid,
"MMX_VALID");
 sc_trace(tf, pid_valid, "PID_VALID");
 sc_trace(tf, pid_data, "PID_DATA");
 sc_trace(tf,                mmic_datain,
"MMIC_DATAIN");
 sc_trace(tf,                mmic_statein,
"MMIC_STATEIN");
 sc_trace(tf, mmic_cs, "MMIC_CS");
 sc_trace(tf, mmic_we, "MMIC_WE");
 sc_trace(tf,                mmic_addr,
"MMIC_ADDR");
 sc_trace(tf,                mmic_dest,
"MMIC_DEST");
 sc_trace(tf,                mmic_destout,
"MMIC_DESTOUT");
 sc_trace(tf,                mmic_dataout,
"MMIC_DATAOUT");
 sc_trace(tf,              mmic_out_valid,
"MMIC_OUT_VALID");
 sc_trace(tf,                mmic_stateout,
"MMIC_STATEOUT");
 sc_trace(tf, in_valid, "IN_VALID");
 sc_trace(tf, negate, "NEGATE");
 sc_trace(tf, add1, "ADD1");
 sc_trace(tf, shift_sel, "SHIFT_SEL");
 sc_trace(tf, c, "C");
 sc_trace(tf, v, "V");
 sc_trace(tf, z, "Z");
 sc_trace(tf, dout, "DOUT");
 sc_trace(tf,                out_valid,
"OUTPUT_VALID");
 sc_trace(tf, destout, "DESTOUT");
 sc_trace(tf, fdout, "FDOUT");
 sc_trace(tf,                fout_valid,
"FOUT_VALID");
 sc_trace(tf, fdestout, "FDESTOUT");
 sc_trace(tf, ireq0, "IREQ0");
 sc_trace(tf, ireq1, "IREQ1");
 sc_trace(tf, ireq2, "IREQ2");
 sc_trace(tf, ireq3, "IREQ3");
 sc_trace(tf, rd_wr, "RD_WR");
 sc_trace(tf, intreq, "INTREQ");
 sc_trace(tf, vectno, "VECTNO");
 sc_trace(tf, intack, "INTACK");
 sc_trace(tf,                intack_cpu,
"INTACK_CPU");
 sc_trace(tf, dsp_in1, "DPS_IN1");
 sc_trace(tf, dsp_out1, "DSP_OUT1");
 sc_trace(tf,                dsp_data_valid,
"DSP_DATA_VALID");
 sc_trace(tf,                dsp_input_valid,
"DSP_INPUT_VALID");
 sc_trace(tf,          dsp_data_requested,
"DSP_DATA_REQUESTED");


////////////////////////////////////////////////////////////
//////////////
```

```
//
                MAIN PROGRAM

////////////////////////////////////////////////////////////
///////////////
  sc_clock clk("Clock", 1, 0.5, 0.0);


printf("////////////////////////////////////////////////////
//////////////////////\n");
  printf("//      This code is written at SYNOPSYS, Inc.\n");

printf("////////////////////////////////////////////////////
//////////////////////\n");
  printf("//  Module    : main of CPU Model\n");
  printf("//  Author    : Martin Wang\n");
  printf("//  Company   : SYNOPSYS, Inc.\n");
  printf("//  Purpose   : This is a simple CPU modeling using SystemC.\n");
  printf("//                Instruction Set Architecure defined by Martin Wang.\n");
  printf("//              \n");
  printf("//            SystemC (TM) Copyright (c) 1988-2000 by Synopsys, Inc.  \n");
  printf("//              \n");

printf("////////////////////////////////////////////////////
//////////////////////\n");
  cout << "// IN THIS MACHINE Interger is " << sizeof (int) << " bytes.\n";
  cout << "// IN THIS MACHINE Floating is " << sizeof (float) << " bytes.\n";
  cout << "// IN THIS MACHINE Double is " << sizeof (double) << " bytes.\n";
  printf("//     \n");
  printf("//     \n");
  printf("//
.,,uod8B8bou,,.\n");
  printf("//
..,uod8BBBBBBBBBBBBBBBRPFT?l!i:.\n");
  printf("//
,=m8BBBBBBBBBBBBBBBRPFT?!||||||||||||\n");
  printf("//
!...:!TVBBBRPFT||||||||!!^^\"\"\"    ||||\n");
  printf("//                !.......:!?||||!!^^\"\"\"    ||||\n");
  printf("//              !.........||||        ### # #  ||||\n");
  printf("//            !.........|||| ### # # # #  ||||\n");
  printf("//          !.........|||| #     # # # #  ||||\n");
  printf("//          !.........|||| #     # # #  # ||||\n");
  printf("//          !.........|||| #     ##    # # ||||\n");
  printf("//          !.........|||| #     #     ### ||||\n");
  printf("//              `.........|||| #     # # ,|||\n");
  printf("//          .;.......|||| ###         _.-!!|||||\n");
  printf("//      .,uodWBBBBb.....||||      _.-!!|||||||||!:'\n");
  printf("//
!YBBBBBBBBBBBBBBb..!|||:..-!!|||||||!iof68BBBBBb....\n");
  printf("//
!..YBBBBBBBBBBBBBBb!!|||||||!iof68BBBBBBRPFT?!::   `.\n");
  printf("//
!....YBBBBBBBBBBBBBBbaaitf68BBBBBBRPFT?!::::::::     `.\n");
  printf("//
!......YBBBBBBBBBBBBBBBBBBBRPFT?!::::::;;!^\"`;:::      `.\n");
  printf("//
!........YBBBBBBBBBBRPFT?!::::::::::^"..
.:::::::;       iBBbo.\n");
  printf("//
`..........YBRPFT?!::::::::::::::::::::::::;iof68bo.     WBBBBbo.\n");
  printf("//
`..........:::::::::::::::::::::::;iof688888888888b.    `YBBBP^'\n");
  printf("//
`........::88::::::::::::;iof688888888888888888888b.    `\n");
  printf("//
`.......::81::::::;iof6888888888888888888888888888b.\n");
  printf("//
`....::::;iof688888888888888888888888888888888899fT!\n");
  printf("//
`..::!88888888888888888888888888888888899fT|!^\"'\"\n");
  printf("//                                `'
!!9888888888888888888888888899fT|!^\"'\n");
  printf("//
`!!88888888888888899fT|!^\"'\"\n");
  printf("//
`!988888888899fT|!^\"'\"\n");
  printf("//                    `!9899fT|!^\"'\"\n");
  printf("//                        `!^\"'\"\n");
  printf("//    \n");
  printf("//    \n");

printf("////////////////////////////////////////////////////
//////////////////////\n\n\n");


  sc_trace_file *wf = sc_create_vcd_trace_file("signals");
  sc_trace(wf, clk, "CLK");
  sc_trace(wf, ram_cs, "ram_cs");
  sc_trace(wf, ram_we, "ram_we");
  sc_trace(wf, addr, "addr");
  sc_trace(wf, instruction, "instruction");
  sc_trace(wf, mem_address, "mem_address");
  sc_trace(wf, alu_op, "alu_op");
  sc_trace(wf, src_A, "src_A");
  sc_trace(wf, src_B, "src_B");
  sc_trace(wf, alu_src, "alu_src");
  sc_trace(wf, destout, "destout");
  sc_trace(wf, decode_valid, "decode_valid");
  sc_trace(wf, program_counter, "program_counter");

  fetch  IFU("FETCH_BLOCK");
         IFU.init_param(delay_cycles);
                IFU << ram_dataout << branch_target_address << next_pc << branch_valid
                << stall_fetch << intreq << vectno << bios_valid << icache_valid
                << pred_fetch << pred_branch_address << pred_branch_valid << ram_cs << ram_we
                << addr << ram_datain << instruction << instruction_valid << program_counter
                << intack_cpu << branch_clear << pred_fetch_valid << reset << clk;

  decode  IDU("DECODE_BLOCK");
                IDU << reset << instruction << pred_instruction << instruction_valid
                << pred_inst_valid << out_valid << destout << dout << dram_dataout
                << dram_rd_valid << destout << fdout << fout_valid << fdestout
```

```
                << branch_clear << dsp_data_valid << program_counter << pred_on
                << branch_instruction_address << next_pc << branch_valid
                << branch_target_address << mem_access << mem_address << alu_op
                << mem_write << alu_src << reg_write << src_A << src_B << forward_A
                << forward_B << stall_fetch << decode_valid << float_valid << mmx_valid
                << pid_valid << pid_data << clk;

 exec
        IEU("EXEC_BLOCK");
                IEU << reset << decode_valid << alu_op << negate << add1 << shift_sel
                << src_A << src_B << forward_A << forward_B << alu_src << c << v << z
                << dout << out_valid << destout << clk;

 floating   FPU("FLOAT_BLOCK");
                // order dependent
                FPU << float_valid << alu_op << src_A << src_B << alu_src
                << fdout << fout_valid << fdestout << clk;

 mmxu
        MMXU("MMX_BLOCK");
                MMXU << mmx_valid << alu_op << src_A << src_B << alu_src
                << fdout << fout_valid << fdestout << clk;

 bios
        BIOS("BIOS_BLOCK");
        BIOS.init_param(delay_cycles);

        BIOS.datain(ram_datain);
        // order independent
        BIOS.cs(ram_cs);
                BIOS.we(ram_we);
                BIOS.addr(addr);

        BIOS.dataout(ram_dataout);

        BIOS.bios_valid(bios_valid);

        BIOS.stall_fetch(stall_fetch);
                BIOS.CLK(clk);

 paging
        PAGING("PAGING_BLOCK");
                PAGING << ram_datain << ram_cs << ram_we << addr << icache_din
                << icache_validin << icache_stall << paging_dout << paging_csout
                << paging_weout << physical_address << ram_dataout << icache_valid
                << stall_fetch << clk;

 icache
        ICACHE("ICACHE_BLOCK");

        ICACHE.init_param(delay_cycles);
                ICACHE << paging_dout << paging_csout << paging_weout
                << physical_address << pid_valid << pid_data << icache_din << icache_validin
                << icache_stall << clk;

 dcache
        DCACHE("DCACHE_BLOCK");

        DCACHE.init_param(delay_cycles);
                DCACHE << mmic_datain << mmic_statein << mmic_cs << mmic_we << mmic_addr
                << mmic_dest << mmic_destout << mmic_dataout << mmic_out_valid << mmic_stateout << clk;

 pic
        APIC("PIC_BLOCK");
                APIC << ireq0 << ireq1 << ireq2 << ireq3 <<intack_cpu << rd_wr
                << intack_cpu << intreq << intack << vectno;

 time_t tbuffer = time(NULL);

 sc_start(clk, -1);

 //cout << "Time for simulation = " << (time(NULL) - tbuffer) << endl;
 sc_close_vcd_trace_file(tf);
  return 0; /* this is necessary */
}
```

Індивідуальне завдання:

```
ldpid  0
movi    R3, 0
movi  R1, 11
movi  R2, 20
add   R3, R2, R1
```

Рис.1. Результат виконання програми

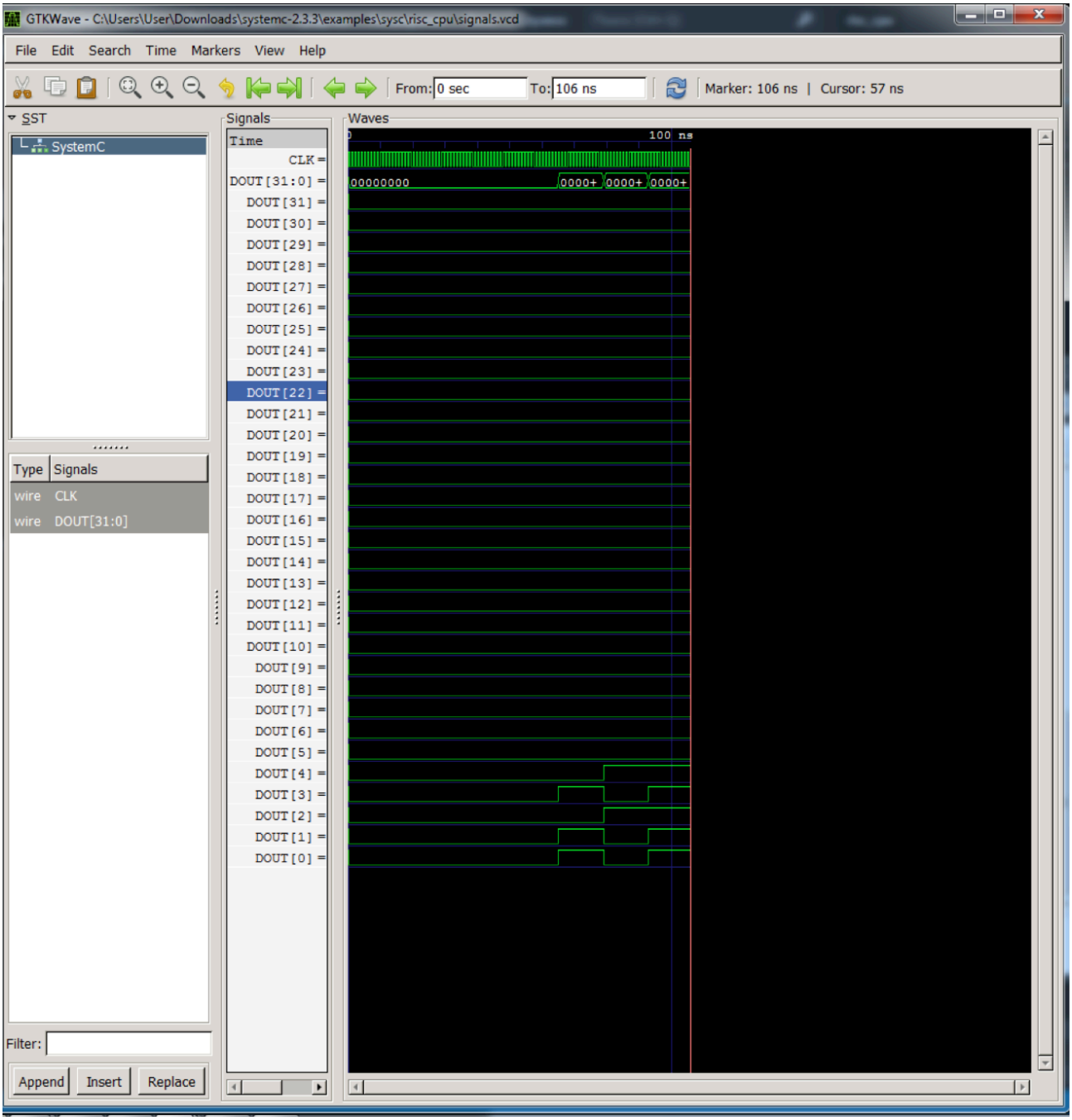

Рис.2. Часова діаграма

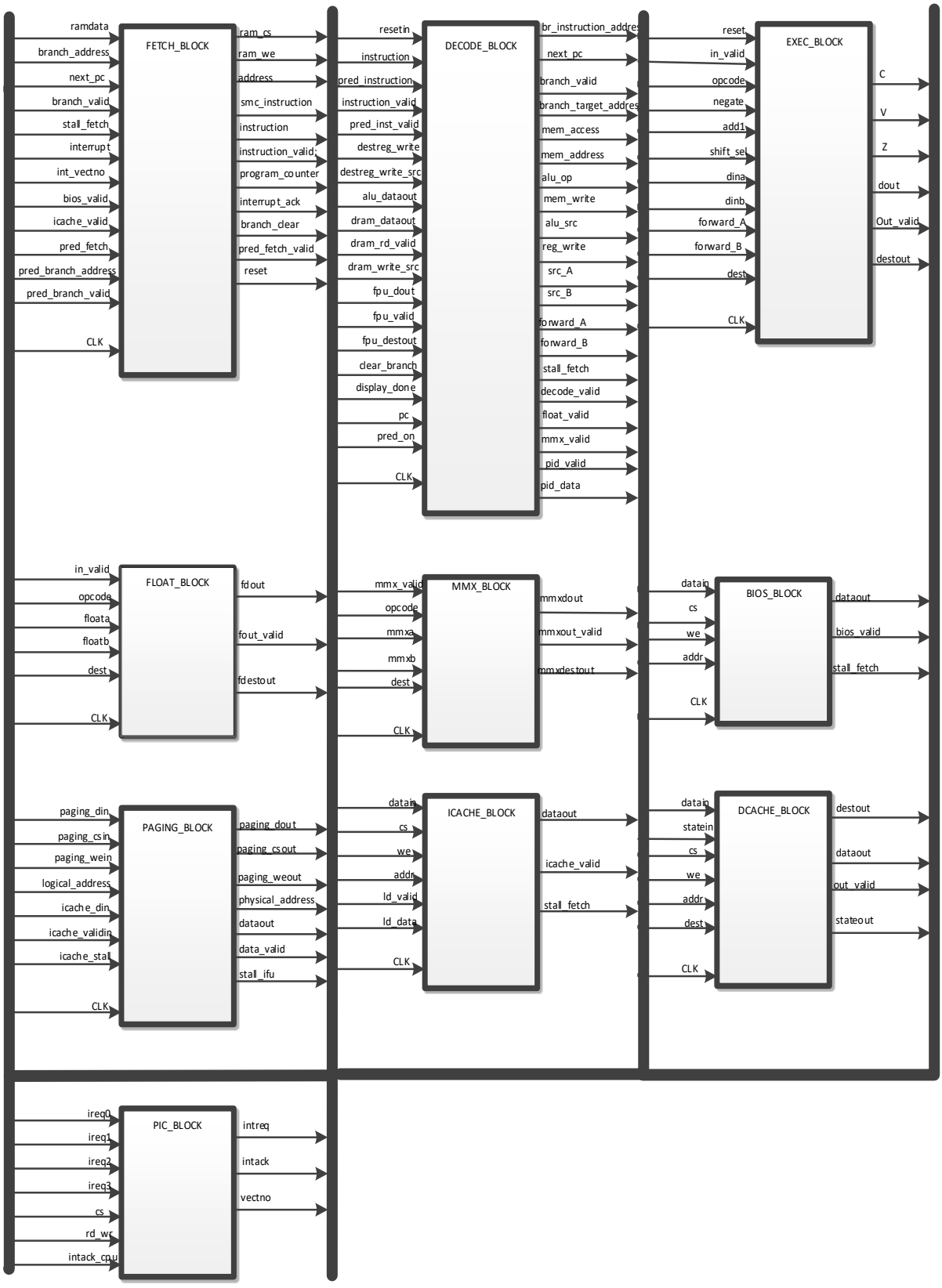

Рис.3 Структурна схема модифікованого конвеєра

Висновок: Навчився здійснювати оцінку структури об’єкта (RISC CPU) на існуючій програмній моделі. Навчився встановлювати структуру інтерфейсів об’єкта.