Санкт-Петербургский государственный политехнический университет

Кафедра компьютерных систем и программных технологий

**Курсовая работа**

**Дисциплина**: Высокоуровневое моделирование средствами SystemC

Выполнил студент гр. 13541/2 Муравьев Ф.Э.

(подпись)

Руководитель Мамутова О.В.

(подпись)

“ ” 2017 г.

Санкт-Петербург

2017

**Курсовая работа**

**Курсовое проектирование. Разработка потактовой модели процессора**

**Раздел 1. Спецификация компонентов процессорной системы.**

Обязательные компоненты системы: АЛУ, контроллер памяти, регистровый файл.

Дополнительные компоненты системы по индивидуальному заданию:

- контроллер прямого доступа к памяти (DMA)

- поддержка защиты памяти (MPU)

- интерфейс ввода/вывода (GPIO)

**Структура разработанной процессорной системы:**

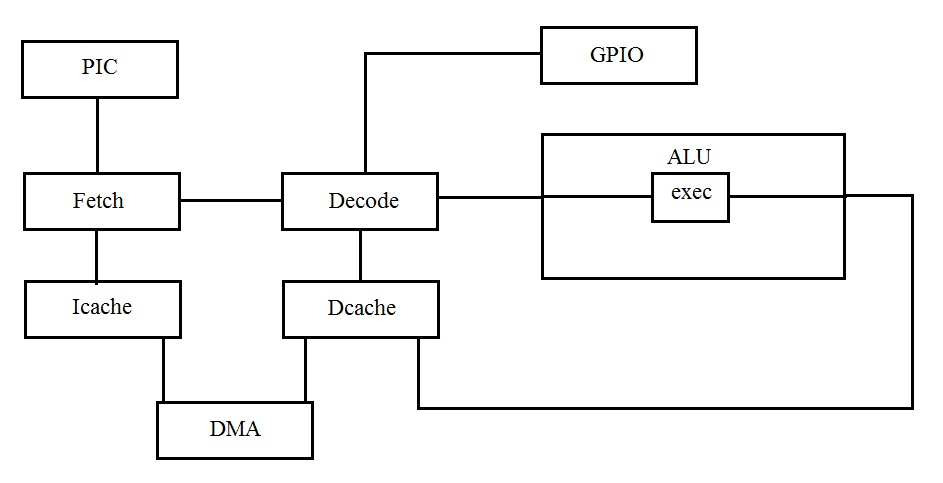


Рис 1.1. Структура разработанной процессорной системы

Модули, входящие в структуру процессорной системы:

1. PIC (Programmable Interrupt Controller) – контроллер прерываний
2. Icache – Instruction cache – модуль, реализующий память инструкций в процессорной системе
3. Dcache – Data cache – модуль, реализующий память данных в процессорной системе
4. Fetch – модуль, основной функцией которого является извлечение инструкций из памяти инструкций
5. Decode – модуль, реализующий декодирование извлеченной инструкции и ее выполнение, также сочетает в себе функции модуля поддержки защиты памяти (MPU). Содержит описание регистрового файла размером 16 регистров.
6. Exec – Арифметическое логическое устройство, отвечает за выполнение арифметических и логических команд
7. DMA – контроллер прямого доступа к памяти
8. GPIO – интерфейс ввода/вывода

**Раздел 2. Спецификация системного окружения для отладки и тестирования процессора.**

Для отладки и тестирования каждого модуля будут разработаны модульные тесты. Наблюдение за результатом работы модуля будет осуществляться с помощью консоли и построения временных диаграмм с последующим их просмотром в программе GTKWave.

**Раздел 3. Спецификация системы команд процессора типа RISC, исходя из выбранного набора компонентов процессорной системы и целевых алгоритмов.**

Команда в разработанном процессоре имеет размерность 32 бита.

0х00000000

Структура команд представлена на рис 3.1.

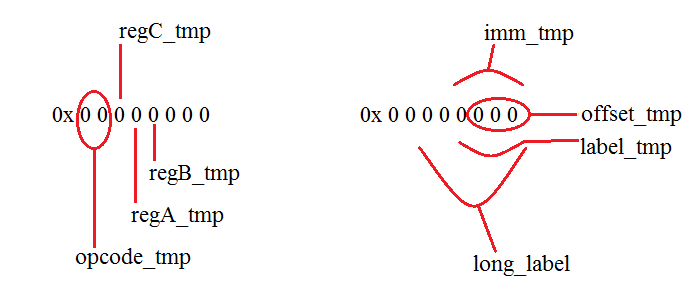


Рис 3.1. Структура команд процессорной системы

Первые два байта команды обозначают код операции – opcode\_tmp.

regC\_tmp – номер регистра-приемника результата.

regА\_tmp – номер регистра-операнда А.

regC\_tmp – номер регистра-операнда В.

Imm\_tmp – абсолютное значение (используется для задания точного значения операнда В)

offset\_tmp – смещение используемое в адресации

label\_tmp, long\_label – метки перехода

Набор команд, реализованных в процессорной системе:

|  |  |  |
| --- | --- | --- |
| Код | Команда | Описание |
| 0х00 | halt | Выводит в консоль состояние внутренних регистров в процессорной системе. Используется только в режиме отладки процессора. |
| 0х01 | add R1, R2, R3 | R1=R2+R3 |
| 0х02 | addi R1, R2, #value | R1=R2+#value |
| 0х03 | addc R1, R2, R3 + Carrybit | R1=R2+R3+Carrybit |
| 0х04 | sub R1, R2, R3 | R1=R2-R3 |
| 0х05 | subi R1, R2, #value | R1=R2-#value |
| 0х06 | subc R1, R2, R3 - Carrybit | R1=R2-R3-Carrybit |
| 0х07 | mul R1, R2, R3 | R1=R2\*R3 |
| 0х08 | div R1, R2, R3 | R1=R2/R3 |
| 0х09 | nand R1, R2, R3 | R1=R2 nand R3 |
| 0х0a | and R1, R2, R3 | R1=R2 and R3 |
| 0х0b | or R1, R2, R3 | R1=R2 or R3 |
| 0х0c | xor R1, R2, R3 | R1=R2 xor R3 |
| 0х0d | NOT R1, R2 | R1= not R2 |
| 0х0e | modulo R1 = R2 mod R3 | R1=R2 % R3 |
| 0х4b | ldgpio R1 | load GPIO. Загрузка из интерфейса ввода/вывода |
| 0х4с | wrgpio R1 | write GPIO. Запись в интерфейс ввода/вывода |
| 0х4d | ld R1, R2, offset | load. Загрузка из памяти данных. R1 – приемник. Адрес= R2+offset |
| 0х4е | wr R1, R2, offset | write. Запись в память данных. R1 – значение. Адрес= R2+offset |
| 0х50 | sec R1, R2 | Смена области защищенной от записи памяти данных. R1 – начальный адрес. R2 – конечный адрес. |
| 0х0f | mov R1, R2 | Копирование регистра R2 в регистр R1 |
| 0х10 | beq R1, R2, label | Переход на метку label если R1=R2 |
| 0х11 | bne R1, R2, label | Переход на метку label если R1!=R2 |
| 0х12 | bgt R1, R2, label | Переход на метку label если R1>R2 |
| 0х13 | bge R1, R2, label | Переход на метку label если R1>=R2 |
| 0х14 | blt R1, R2, label | Переход на метку label если R1<R2 |
| 0х15 | ble R1, R2, label | Переход на метку label если R1=<R2 |
| 0х16 | j label | Переход на метку label |
| 0хe0 | flush register | Обнуление регистров процессора |
| 0хf1 | movi R1, #value | Запись в регистр R1 значения #value |
| 0хff | QUIT | Завершение работы процессора |

**Раздел 4. Разработка описания ядра процессора, компонентов процессорной системы и элементов системного окружения на языке SystemC.**

**4.1. Модуль памяти данных – Dcache**

Листинг 4.1. Файл Dcache.h

|  |
| --- |
| #define MAX\_DATA\_LENGTH 10000  #include "systemc.h"  #include "directive.h"  struct dcache : sc\_module {  sc\_in<signed> datain; // input data  sc\_in<bool> cs; // chip select  sc\_in<bool> we; // write enable  sc\_in<unsigned > addr; // address  sc\_out<signed> dataout; // dataram data out  sc\_out<bool> out\_valid; // output valid  sc\_in\_clk CLK;  // Parameter  unsigned \*dmemory; // data memory  int wait\_cycles; // cycles # it takes to access dmemory  void init\_param(int given\_delay\_cycles) {  wait\_cycles = given\_delay\_cycles;  }    //Constructor  SC\_CTOR(dcache) {  SC\_CTHREAD(entry, CLK.pos());  // initialize instruction dmemory from external file  FILE \*fp = fopen("dcache","r");  int size=0;  int i=0;  int mem\_word;  dmemory = new unsigned[MAX\_DATA\_LENGTH];  printf("\*\* ALERT \*\* DCU: initialize Data Cache\n");  while (fscanf(fp,"%x", &mem\_word) != EOF) {  dmemory[size] = mem\_word;  size++;  }  for (i=size; i<MAX\_DATA\_LENGTH; i++) {  dmemory[i] = 0xdeadbeef;  }  }  // Process functionality in member function below  void entry();  };  void dcache::entry()  {  unsigned int address; // address to DataCache  while (true) {  do { wait(); } while ( !(cs == true) );  address = addr.read();  if (we.read() == true) { // Write operation  wait();  out\_valid.write(false);  if (address < MAX\_DATA\_LENGTH && address >= 0)  {  dmemory[address] = datain.read();  if (PRINT\_DCU) {  cout << "\t\t\t\t\t\t\t-------------------------------" << endl;  printf("\t\t\t\t\t\t\tDCU :wr %x->mem[%d]\n", dmemory[address], address);  cout << "\t\t\t\t\t\t\tat CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t\t\t\t\t-------------------------------" << endl;  }  }  else  {  cout << "\t\t\t\t\t\t\t-------------------------------" << endl;  printf("\t\t\t\t\t\t\tDCU ALERT: \*\*MEMORY OUT OF RANGE\*\*\n");  cout << "\t\t\t\t\t\t\t-------------------------------" << endl;  }    wait();  // Save to file  FILE \*save\_data = fopen("dcache\_out","w");  for (int kol=0; kol<4000; kol++) {  fprintf(save\_data, "0x%08x\n", dmemory[kol]);  }  fclose(save\_data);  //  }  else { // Read operation  wait();  if (address < MAX\_DATA\_LENGTH && address >= 0)  {  dataout.write(dmemory[address]);  if (PRINT\_DCU) {  cout << "\t\t\t\t\t\t\t-------------------------------" << endl;  printf("\t\t\t\t\t\t\tDCU :ld 0x%x(%d)<-mem[%d]\n", dmemory[address], dmemory[address], address);  cout << "\t\t\t\t\t\t\tat CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t\t\t\t\t-------------------------------" << endl;  }  }  else  {  dataout.write(0xffffffff);  cout << "\t\t\t\t\t\t\t-------------------------------" << endl;  printf("\t\t\t\t\t\t\tDCU ALERT: \*\*MEMORY OUT OF RANGE\*\*\n");  if (PRINT\_DCU) {  cout << "\t\t\t\t\t\t\t-------------------------------" << endl;  printf("\t\t\t\t\t\t\tDCU :ld 0x%x(%d)<-mem[error %x]\n", 0xffffffff, 0xffffffff, address);  cout << "\t\t\t\t\t\t\tat CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t\t\t\t\t-------------------------------" << endl;  }  }  out\_valid.write(true);    wait();  out\_valid.write(false);  wait();  }  }  } // end of entry function |

В разработанной процессорной системе используется память на 10000 элементов, однако она может быть увеличена до 232 элементов.

**4.2. Модуль памяти инструкций – Icache**

Листинг 4.2. Файл Icache.h

|  |
| --- |
| #define MAX\_CODE\_LENGTH 1000  #include <systemc.h>  #include "directive.h"  struct icache : sc\_module {  sc\_in<unsigned> datain; // modified instruction  sc\_in<bool> cs; // chip select  sc\_in<bool> we; // write enable for SMC(Static Memory Controller)  sc\_in<unsigned > addr; // address  sc\_out<unsigned > dataout; // ram data out  sc\_out<bool> icache\_valid; // output valid  sc\_in\_clk CLK;  // Parameter  unsigned \*icmemory; // icache data memory  int wait\_cycles; // Number of cycles it takes to access imemory  void init\_param(int given\_delay\_cycles) {  wait\_cycles = given\_delay\_cycles;  }    //Constructor  SC\_CTOR(icache) {  SC\_CTHREAD(entry, CLK.pos());  // initialize instruction icmemory from external file  FILE \*fp = fopen("icache","r");  int size=0;  int mem\_word;  icmemory = new unsigned[MAX\_CODE\_LENGTH];  for (size = 0; size < MAX\_CODE\_LENGTH; size++) { // initialize bad data  icmemory[size] = 0xeeeeeeee;  }  size = 0;  while (fscanf(fp,"%x", &mem\_word) != EOF) {  icmemory[size] = mem\_word;  size++;  }  }  // Process functionality in member function below  void entry();  };  void icache::entry()  {  unsigned int address;  while (true) {  do { wait(); } while ( !(cs == true) );  address = addr.read();  if (we.read() == true) { // Write operation  wait();  if (address < MAX\_CODE\_LENGTH && address >= 0)  {  icmemory[address] = datain.read();  printf("------------------------\n");  printf("ICU: write to mem[%d]=%x\n", address, datain.read());  printf("------------------------\n");  }  else  printf("ICU ALERT: \*\*MEMORY OUT OF RANGE\*\*\n");  wait();  }  else { // Read operation  wait(); // Introduce delay needed  if (address >= MAX\_CODE\_LENGTH || address < 0) {  dataout.write(0xffffffff);  printf("ICU ALERT: \*\*MEMORY OUT OF RANGE\*\*\n");  }  else  dataout.write(icmemory[address]);  icache\_valid.write(true);  if (PRINT\_ICU) {  printf("------------------------\n");  printf("ICU: fetching mem[%d]\n", address);  if (address < MAX\_CODE\_LENGTH && address >= 0)  printf("ICU: (%0x)", icmemory[address]);  cout.setf(ios::dec,ios::basefield);  cout << " at CSIM " << sc\_time\_stamp() << endl;  printf("------------------------\n");  }  wait();  icache\_valid.write(false);  wait();  }    }  } |

В разработанной процессорной системе используется память на 1000 элементов, однако она может быть увеличена до 232 элементов. Также первые пять адресов 0х0-0х00000005 зарезервированы под обработку прерываний.

**4.3. Модуль извлечения инструкций из памяти – Fetch**

Листинг 4.3. Файл Fetch.h

|  |
| --- |
| #include "systemc.h"  #include "directive.h"  struct fetch : sc\_module {  sc\_in<unsigned > ramdata; // instruction from RAM  sc\_in<unsigned > branch\_address; // branch target address  sc\_in<bool> next\_pc; // pc ++  sc\_in<bool> branch\_valid; // branch\_valid  sc\_in<bool> stall\_fetch; // STALL\_FETCH  sc\_in<bool> interrupt; // interrrupt  sc\_in<unsigned> int\_vectno; // interrupt vector number  sc\_in<bool> icache\_valid; // Icache input valid  sc\_out<bool> ram\_cs; // RAM chip select  sc\_out<bool> ram\_we; // RAM write enable for SMC  sc\_out<unsigned > address; // address send to RAM  sc\_out<unsigned> instruction; // instruction send to ID  sc\_out<bool> instruction\_valid; // inst valid  sc\_out<bool> interrupt\_ack; // interrupt acknowledge  sc\_out<bool> busy; // interrupt acknowledge  sc\_in\_clk CLK;  // Parameter  int memory\_latency;  void init\_param(int given\_delay\_cycles) {  memory\_latency = given\_delay\_cycles;  }    //Constructor  SC\_CTOR(fetch) {  SC\_CTHREAD(entry, CLK.pos());  }  // Process functionality in member function below  void entry();  };  void fetch::entry()  {  unsigned addr\_tmp=0;  unsigned datao\_tmp=0;  unsigned datai\_tmp=0;  addr\_tmp = 4;  datao\_tmp = 0xdeadbeef;  while (true) {  if (stall\_fetch.read() == false) {  busy=true;  if (interrupt.read() == true) {  ram\_cs.write(true);  addr\_tmp = int\_vectno.read();  address.write(addr\_tmp);  ram\_we.write(false);  wait(memory\_latency);  do { wait(); } while ( !(icache\_valid == true) );  datai\_tmp = ramdata.read();  if (PRINT\_IFU) {  printf("IF ALERT: \*\*INTERRUPT\*\*\n");  cout.setf(ios::hex,ios::basefield);  cout << "------------------------" << endl;  cout << "IFU :" << " mem=0x" << datai\_tmp << endl;  cout << "IFU : pc= " << addr\_tmp ;  cout.setf(ios::dec,ios::basefield);  cout << " at CSIM " << sc\_time\_stamp() << endl;  cout << "------------------------" << endl;  }  instruction\_valid.write(true);  instruction.write(datai\_tmp);  ram\_cs.write(false);  interrupt\_ack.write(true);  if (next\_pc.read() == true) { addr\_tmp++; }  wait();  instruction\_valid.write(false);  interrupt\_ack.write(false);  wait();  }  if (branch\_valid.read() == true) {  ram\_cs.write(true);  addr\_tmp = branch\_address.read();  address.write(addr\_tmp);  ram\_we.write(false);  wait(memory\_latency);  do { wait(); } while ( !(icache\_valid == true) );  datai\_tmp = ramdata.read();  if (PRINT\_IFU) {  printf("IFU ALERT: \*\*BRANCH\*\*\n");  cout.setf(ios::hex,ios::basefield);  cout << "------------------------" << endl;  cout << "IFU :" << " mem=0x" << datai\_tmp << endl;  cout << "IFU : pc= " << addr\_tmp ;  cout.setf(ios::dec,ios::basefield);  cout << " at CSIM " << sc\_time\_stamp() << endl;  cout << "------------------------" << endl;  }  instruction\_valid.write(true);  instruction.write(datai\_tmp);  ram\_cs.write(false);  if (next\_pc.read() == true) { addr\_tmp++; }  wait();  instruction\_valid.write(false);  wait();  } else {  ram\_cs.write(true);  address.write(addr\_tmp);  ram\_we.write(false);  wait(memory\_latency); // For data to appear  do { wait(); } while ( !(icache\_valid == true) );  datai\_tmp = ramdata.read();  if (PRINT\_IFU) {  cout.setf(ios::hex,ios::basefield);  cout << "------------------------" << endl;  cout << "IFU :" << " mem=0x" << datai\_tmp << endl;  cout << "IFU : pc= " << addr\_tmp ;  cout.setf(ios::dec,ios::basefield);  cout << " at CSIM " << sc\_time\_stamp() << endl;  cout << "------------------------" << endl;  }  instruction\_valid.write(true);  instruction.write(datai\_tmp);  ram\_cs.write(false);  if (next\_pc.read() == true) { addr\_tmp++; }  wait();  instruction\_valid.write(false);  wait();  }  wait();  }  else  {  cout << "------------------------" << endl;  cout << "IFU: Pause Fetch" << endl;  cout.setf(ios::dec,ios::basefield);  cout << " at CSIM " << sc\_time\_stamp() << endl;  cout << "------------------------" << endl;  busy=false;  do{ wait(); } while ( !(stall\_fetch.read() == false) );  }  }  } // end of entry function |

**4.4. Модуль декодирования извлеченной инструкции – Decode**

Листинг 4.4. Файл Decode.h

|  |
| --- |
| #include <climits>  #include "systemc.h"  #include "directive.h"  struct decode : sc\_module {  sc\_in<unsigned> instruction; // fetched instruction  sc\_in<bool> instruction\_valid; // input valid  sc\_in<bool> destreg\_write; // register write enable  sc\_in<unsigned> destreg\_write\_src; // which register to write?  sc\_in<signed> alu\_dataout; // data from ALU  sc\_in<signed> dram\_dataout; // data from Dcache  sc\_in<bool> dram\_rd\_valid; // Dcache read data valid  sc\_in<signed> fpu\_dout; // data from FPU(floating point unit)  sc\_in<bool> fpu\_valid; // FPU data valid  sc\_in<unsigned> fpu\_destout; // write to which register  sc\_out<bool> next\_pc; // next pc ++ ?  sc\_out<bool> branch\_valid; // branch valid signal  sc\_out<unsigned > branch\_target\_address; // branch target address  sc\_out<bool> mem\_access; // memory access valid  sc\_out<unsigned > mem\_address; // memory physical address  sc\_out<int> alu\_op; // ALU/FPU/MMU Opcode  sc\_out<bool> mem\_write; // memory write enable  sc\_out<unsigned> alu\_src; // destination register number  sc\_out<signed int> src\_A; // operand A  sc\_out<signed int> src\_B; // operand B  sc\_out<bool> decode\_valid; // decoder output valid  sc\_out<bool> float\_valid; // enable FPU  sc\_out<bool> gpio\_access; //gpio access valid  sc\_out<signed int> dramdata;  sc\_in<bool> dma\_access; //DMA access  sc\_out<bool> ireq0; // interrupt request 0  sc\_in\_clk CLK;  signed int cpu\_reg[16]; //CPU register  unsigned int dram\_write\_src; //Whitch register write data from dram  //Constructor  SC\_CTOR(decode) {  SC\_CTHREAD(entry, CLK.pos());  FILE \*fp = fopen("register","r");  int size=0;  int mem\_word;  printf("\*\* ALERT \*\* ID: initialize Architectural Registers\n");  while (fscanf(fp,"%x", &mem\_word) != EOF) {  cpu\_reg[size] = mem\_word;  size++;  }  dram\_write\_src=0;  }  // Process functionality in member function below  void entry();  };  void decode::entry()  {  unsigned int instr\_tmp = 0;  unsigned int opcode\_tmp = 0;  unsigned int regA\_tmp = 0;  unsigned int regB\_tmp = 0;  unsigned int regC\_tmp = 0;  unsigned int imm\_tmp = 0;  unsigned int offset\_tmp = 0;  signed int label\_tmp = 0;  unsigned int longlabel\_tmp = 0;  unsigned int lastreg\_tmp = 0;  signed int srcA\_tmp = 0;  signed int srcB\_tmp = 0;  signed int srcC\_tmp = 0;  int i;  unsigned int start\_addr = 0;  unsigned int fin\_addr =10000;    branch\_valid.write(false);  decode\_valid.write(false);  float\_valid.write(false);  wait(2);  while (true) {  if (destreg\_write.read() == true) {  cpu\_reg[destreg\_write\_src.read()] = alu\_dataout.read();  if (PRINT\_ID) {  cout << "\t\t\t-------------------------------" << endl;  printf("\t\t\tID: R%d=0x%x(%d) fr ALU", destreg\_write\_src.read(), alu\_dataout.read(),alu\_dataout.read());  cout.setf(ios::dec,ios::basefield);  cout << " at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  }  if (dram\_rd\_valid.read() == true && dma\_access.read()!=true) {  cpu\_reg[dram\_write\_src] = dram\_dataout.read();  if (PRINT\_ID) {  cout << "\t\t\t-------------------------------" << endl;  printf("\t\t\tID: R%d=0x%x(%d) fr MemLd/GPIO", dram\_write\_src, dram\_dataout.read(), dram\_dataout.read());  cout.setf(ios::dec,ios::basefield);  cout << " at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  dram\_write\_src=0;  }  if (fpu\_valid.read() == true) {  cpu\_reg[fpu\_destout.read()] = fpu\_dout.read();  if (PRINT\_ID) {  cout << "\t\t\t-------------------------------" << endl;  printf("\t\t\tID: R%d=0x%x fr MMX", fpu\_destout.read(), fpu\_dout.read());  cout.setf(ios::dec,ios::basefield);  cout << " at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  }  if ((instruction\_valid.read() == true)) {  instr\_tmp = instruction.read();  opcode\_tmp = (instr\_tmp & 0xff000000) >> 24;  regC\_tmp = (instr\_tmp & 0x00f00000) >> 20;  regA\_tmp = (instr\_tmp & 0x000f0000) >> 16;  regB\_tmp = (instr\_tmp & 0x0000f000) >> 12;  imm\_tmp = (instr\_tmp & 0x0000ffff);  offset\_tmp = (instr\_tmp & 0x00000fff);  label\_tmp = (instr\_tmp & 0x0000ffff);  longlabel\_tmp = (instr\_tmp & 0x00ffffff);  //printf("opcode = %d regC = %d regA = %d regB = %d\n",opcode\_tmp, regC\_tmp, regA\_tmp, regB\_tmp);  srcA\_tmp = cpu\_reg[regA\_tmp];  srcB\_tmp = cpu\_reg[regB\_tmp];  srcC\_tmp = cpu\_reg[regC\_tmp];  wait();  lastreg\_tmp = regC\_tmp;  if (PRINT\_ID) {  cout << "\t\t\t-------------------------------" << endl;  }  switch(opcode\_tmp) {  case 0x0: // halt  if (PRINT\_ID) {  printf("\n\n\t\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  printf("\t\t\tID: REGISTERS DUMP");  cout << " at CSIM " << sc\_time\_stamp() << endl;  printf("\t\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  printf("REG :==================================================================\n");  for(i =0; i<16; i++){  printf(" R%2d(%08x) ",i, cpu\_reg[i]);  if ((i==3) || (i== 11) ||(i==7) || (i==15)){  printf("\n");  }  }  printf("=======================================================================\n\n");  }  wait();  wait();  break;  case 0x01: // add R1, R2, R3  if (PRINT\_ID) {  printf("\t\t\tID: R%d= R%d(=%d)+R%d(=%d)",  regC\_tmp, regA\_tmp, srcA\_tmp, regB\_tmp, srcB\_tmp);  cout << " \n\t\t\t : at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  src\_A.write(srcA\_tmp);  src\_B.write(srcB\_tmp);  alu\_src.write(regC\_tmp);  alu\_op.write(3);  decode\_valid.write(true);  wait();  decode\_valid.write(false);  wait();  break;  case 0x02: // addi R1, R2, #value  if (PRINT\_ID) {  printf("\t\t\tID: R%d= R%d(=%d)+%d",  regC\_tmp, regA\_tmp, srcA\_tmp, imm\_tmp);  cout << " \n\t\t\t : at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  src\_A.write(srcA\_tmp);  src\_B.write(imm\_tmp);  alu\_src.write(regC\_tmp);  alu\_op.write(3);  decode\_valid.write(true);  wait();  decode\_valid.write(false);  wait();  break;  case 0x03: // addc R1, R2, R3 + Carrybit  if (PRINT\_ID) {  printf("\t\t\tID: R%d= R%d(=%d)+R%d(=%d)+C",  regC\_tmp, regA\_tmp, srcA\_tmp, regB\_tmp, srcB\_tmp);  cout << " \n\t\t\t : at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  src\_A.write(srcA\_tmp);  src\_B.write(srcB\_tmp);  alu\_src.write(regC\_tmp);  alu\_op.write(1);  decode\_valid.write(true);  wait();  decode\_valid.write(false);  wait();  break;  case 0x04: // sub R1, R2, R3  if (PRINT\_ID) {  printf("\t\t\tID: R%d=R%d(=%d)-R%d(=%d)",  regC\_tmp, regA\_tmp, srcA\_tmp, regB\_tmp, srcB\_tmp);  cout << " \n\t\t\t : at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  src\_A.write(srcA\_tmp);  src\_B.write(srcB\_tmp);  alu\_src.write(regC\_tmp);  alu\_op.write(4);  decode\_valid.write(true);  wait();  decode\_valid.write(false);  wait();  break;  case 0x05: // subi R1, R2, #value  if (PRINT\_ID) {  printf("\t\t\tID: R%d=R%d(=%d)-%d",  regC\_tmp, regA\_tmp, srcA\_tmp, imm\_tmp);  cout << " \n\t\t\t : at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  src\_A.write(srcA\_tmp);  src\_B.write(imm\_tmp);  alu\_src.write(regC\_tmp);  alu\_op.write(4);  decode\_valid.write(true);  wait();  decode\_valid.write(false);  wait();  break;  case 0x06: // subc R1, R2, R3 - Carrybit  if (PRINT\_ID) {  printf("\t\t\tID: R%d=R%d(=%d)-R%d(=%d)-C",  regC\_tmp, regA\_tmp, srcA\_tmp, regB\_tmp, srcB\_tmp);  cout << " \n\t\t\t : at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  src\_A.write(srcA\_tmp);  src\_B.write(srcB\_tmp);  alu\_src.write(regC\_tmp);  alu\_op.write(2);  decode\_valid.write(true);  wait();  decode\_valid.write(false);  wait();  break;  case 0x07: // mul R1, R2, R3  if (PRINT\_ID) {  printf("\t\t\tID: R%d=R%d(=%d)\*R%d(=%d)",  regC\_tmp, regA\_tmp, srcA\_tmp, regB\_tmp, srcB\_tmp);  cout << " \n\t\t\t : at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  src\_A.write(srcA\_tmp);  src\_B.write(srcB\_tmp);  alu\_src.write(regC\_tmp);  alu\_op.write(5);  decode\_valid.write(true);  wait();  decode\_valid.write(false);  wait();  break;  case 0x08: // div R1, R2, R3  if (PRINT\_ID) {  printf("\t\t\tID: R%d=R%d(=%d)/R%d(=%d)",  regC\_tmp, regA\_tmp, srcA\_tmp, regB\_tmp, srcB\_tmp);  cout << " \n\t\t\t : at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  src\_A.write(srcA\_tmp);  src\_B.write(srcB\_tmp);  alu\_src.write(regC\_tmp);  alu\_op.write(6);  decode\_valid.write(true);  wait();  decode\_valid.write(false);  wait();  break;  case 0x09: // nand R1, R2, R3  if (PRINT\_ID) {  printf("\t\t\tID: R%d=R%d(=%x) nand R%d(=%x)",  regC\_tmp, regA\_tmp, srcA\_tmp, regB\_tmp, srcB\_tmp);  cout << " \n\t\t\t : at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  src\_A.write(srcA\_tmp);  src\_B.write(srcB\_tmp);  alu\_src.write(regC\_tmp);  alu\_op.write(7);  decode\_valid.write(true);  wait();  decode\_valid.write(false);  wait();  break;  case 0x0a: // and R1, R2, R3  if (PRINT\_ID) {  printf("\t\t\tID: R%d=R%d(=%x) and R%d(=%x)",  regC\_tmp, regA\_tmp, srcA\_tmp, regB\_tmp, srcB\_tmp);  cout << " \n\t\t\t : at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  src\_A.write(srcA\_tmp);  src\_B.write(srcB\_tmp);  alu\_src.write(regC\_tmp);  alu\_op.write(8);  decode\_valid.write(true);  wait();  decode\_valid.write(false);  wait();  break;  case 0x0b: // or R1, R2, R3  if (PRINT\_ID) {  printf("\t\t\tID: R%d=R%d(=%x) or R%d(=%x)",  regC\_tmp, regA\_tmp, srcA\_tmp, regB\_tmp, srcB\_tmp);  cout << " \n\t\t\t : at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  src\_A.write(srcA\_tmp);  src\_B.write(srcB\_tmp);  alu\_src.write(regC\_tmp);  alu\_op.write(9);  decode\_valid.write(true);  wait();  decode\_valid.write(false);  wait();  break;  case 0x0c: // xor R1, R2, R3  if (PRINT\_ID) {  printf("\t\t\tID: R%d=R%d(=%x) xor R%d(=%x)",  regC\_tmp, regA\_tmp, srcA\_tmp, regB\_tmp, srcB\_tmp);  cout << " \n\t\t\t : at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  src\_A.write(srcA\_tmp);  src\_B.write(srcB\_tmp);  alu\_src.write(regC\_tmp);  alu\_op.write(10);  decode\_valid.write(true);  wait();  decode\_valid.write(false);  wait();  break;  case 0x0d: // NOT R1, R2  if (PRINT\_ID) {  printf("\t\t\tID: R%d= NOT R%d(=%x)",  regC\_tmp, regA\_tmp, srcA\_tmp);  cout << " \n\t\t\t : at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  src\_A.write(srcA\_tmp);  src\_B.write(0);  alu\_src.write(regC\_tmp);  alu\_op.write(11);  decode\_valid.write(true);  wait();  decode\_valid.write(false);  wait();  break;  case 0x0e: // modulo R1 = R2 mod R3  if (PRINT\_ID) {  printf("\t\t\tID: R%d= R%d(=%x) mod R%d(=%x)",  regC\_tmp, regA\_tmp, srcA\_tmp, regB\_tmp, srcB\_tmp);  cout << " \n\t\t\t : at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  src\_A.write(srcA\_tmp);  src\_B.write(srcB\_tmp);  alu\_src.write(regC\_tmp);  alu\_op.write(14);  decode\_valid.write(true);  wait();  decode\_valid.write(false);  wait();  break;  case 0x4b: // ldgpio R1  if (PRINT\_ID) {  printf("\t\t\tID: R%d<=gpio",  regC\_tmp);  cout << " \n\t\t\t : at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  dram\_write\_src=regC\_tmp;  gpio\_access.write(true);  mem\_write.write(false);  wait();  gpio\_access.write(false);  wait();  break;  case 0x4c: // wrgpio R1  if (PRINT\_ID) {  printf("\t\t\tID: R%d(0x%x)=>gpio",  regC\_tmp, srcC\_tmp);  cout << " \n\t\t\t : at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  gpio\_access.write(true);  mem\_write.write(true);  src\_A.write(srcC\_tmp);  wait();  gpio\_access.write(false);  mem\_write.write(false);  wait();  break;  case 0x4d: // ld R1, R2, offset  if (PRINT\_ID) {  printf("\t\t\tID: R%d<=mem[R%d=(%d)+%d]",  regC\_tmp, regA\_tmp, srcA\_tmp, offset\_tmp);  cout << " \n\t\t\t : at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  dram\_write\_src=regC\_tmp;  mem\_access.write(true);  mem\_write.write(false);  offset\_tmp = offset\_tmp + srcA\_tmp;  mem\_address.write(offset\_tmp);  wait();  mem\_access.write(false);  wait();  break;  case 0x4e: // wr R1, R2, offset  if (PRINT\_ID) {  printf("\t\t\tID: R%d(0x%x)=>mem[R%d(=%d) + %d]",  regC\_tmp, srcC\_tmp, regA\_tmp, srcA\_tmp, offset\_tmp);  cout << " \n\t\t\t : at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  offset\_tmp = offset\_tmp + srcA\_tmp;  if (!(offset\_tmp>=start\_addr && offset\_tmp<=fin\_addr))  {  mem\_access.write(true);  mem\_write.write(true);  mem\_address.write(offset\_tmp);  dramdata.write(srcC\_tmp);  wait();  mem\_access.write(false);  mem\_write.write(false);  wait();  }  else  {  printf("\t\t\tID: You dont have permission to write in this mem area=0x%x",  offset\_tmp);  cout << " \n\t\t\t : at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  cpu\_reg[15]=0xf0000000+offset\_tmp;  ireq0.write(true);  wait();  ireq0.write(false);  }  break;  case 0x50: // sec R1, R2  if(srcC\_tmp<srcA\_tmp)  {  start\_addr=srcC\_tmp;  fin\_addr=srcA\_tmp;  printf("\t\t\tID: Change secure area: start=0x%x fin=0x%x",  srcC\_tmp, srcA\_tmp);  cout << " \n\t\t\t : at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  else  {  printf("\t\t\tID: Error in change secure area: start=0x%x fin=0x%x",  srcC\_tmp, srcA\_tmp);  cout << " \n\t\t\tStart>Fin\n\t\t\t : at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  break;  case 0x0f: // mov R1, R2  if (PRINT\_ID) {  printf("\t\t\tID: R%d=R%d(=%d)",  regC\_tmp, regA\_tmp, srcA\_tmp);  cout << " \n\t\t\t : at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  src\_A.write(srcA\_tmp);  src\_B.write(0);  alu\_src.write(regC\_tmp);  alu\_op.write(3);  decode\_valid.write(true);  wait();  decode\_valid.write(false);  wait();  break;  case 0x10: // beq R1, R2, label  if (srcC\_tmp == srcA\_tmp) {  branch\_target\_address.write(label\_tmp);  branch\_valid.write(true);  if (PRINT\_ID) {  printf("\t\t\tID: beq R%d(=%d), R%d(=%d), pc=(%d).\n",  regC\_tmp, srcC\_tmp, regA\_tmp, srcA\_tmp, label\_tmp);  }  } else {  if (PRINT\_ID) {  printf("\t\t\tID: beq R%d(=%d) != R%d(=%d),pc++.\n",  regC\_tmp, srcC\_tmp, regA\_tmp, srcA\_tmp);  }  }  if (PRINT\_ID) {  cout << "\t\t\tID: at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  wait();  decode\_valid.write(false);  branch\_valid.write(false);  wait();  break;  case 0x11: // bne R1, R2, label  if (srcC\_tmp != srcA\_tmp) {  branch\_target\_address.write(label\_tmp);  branch\_valid.write(true);  if (PRINT\_ID) {  printf("\t\t\tID: bne R%d(=%d), R%d(=%d), pc=(%d).\n",  regC\_tmp, srcC\_tmp, regA\_tmp, srcA\_tmp, label\_tmp);  }  } else {  if (PRINT\_ID) {  printf("\t\t\tID: bne R%d(=%d) = R%d(=%d),pc++.\n",  regC\_tmp, srcC\_tmp, regA\_tmp, srcA\_tmp);  }  }  if (PRINT\_ID) {  cout << "\t\t\tID: at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  wait();  decode\_valid.write(false);  branch\_valid.write(false);  wait();  break;  case 0x12: // bgt R1, R2, label  if (srcC\_tmp > srcA\_tmp) {  branch\_target\_address.write(label\_tmp);  branch\_valid.write(true);  if (PRINT\_ID) {  printf("\t\t\tID: bgt R%d(=%d)>R%d(=%d), pc=(%d).\n",  regC\_tmp, srcC\_tmp, regA\_tmp, srcA\_tmp, label\_tmp);  }  } else {  if (PRINT\_ID) {  printf("\t\t\tID: bgt R%d(=%d) <= R%d(=%d),pc++.\n",  regC\_tmp, srcC\_tmp, regA\_tmp, srcA\_tmp);  }  }  if (PRINT\_ID) {  cout << "\t\t\tID: at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  wait();  decode\_valid.write(false);  branch\_valid.write(false);  wait();  break;  case 0x13: // bge R1, R2, label  if (srcC\_tmp >= srcA\_tmp) {  branch\_target\_address.write(label\_tmp);  branch\_valid.write(true);  if (PRINT\_ID) {  printf("\t\t\tID: bge R%d(=%d)>=R%d(=%d), pc+=(%d).\n",  regC\_tmp, srcC\_tmp, regA\_tmp, srcA\_tmp, label\_tmp);  }  } else {  if (PRINT\_ID) {  printf("\t\t\tID: bge R%d(=%d) < R%d(=%d),pc++.\n",  regC\_tmp, srcC\_tmp, regA\_tmp, srcA\_tmp);  }  }  if (PRINT\_ID) {  cout << "\t\t\tID: at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  wait();  decode\_valid.write(false);  branch\_valid.write(false);  wait();  break;  case 0x14: // blt R1, R2, label  if (srcC\_tmp < srcA\_tmp) {  branch\_target\_address.write(label\_tmp);  branch\_valid.write(true);  if (PRINT\_ID) {  printf("\t\t\tID: blt R%d(=%d)<R%d(=%d), pc=(%d).\n",  regC\_tmp, srcC\_tmp, regA\_tmp, srcA\_tmp, label\_tmp);  }  } else {  if (PRINT\_ID) {  printf("\t\t\tID: blt R%d(=%d) >= R%d(=%d), pc++.\n",  regC\_tmp, srcC\_tmp, regA\_tmp, srcA\_tmp);  }  }  if (PRINT\_ID) {  cout << "\t\t\tID: at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  wait();  decode\_valid.write(false);  branch\_valid.write(false);  wait();  break;  case 0x15: // ble R1, R2, label  if (srcC\_tmp <= srcA\_tmp) {  branch\_target\_address.write(label\_tmp);  branch\_valid.write(true);  if (PRINT\_ID) {  printf("\t\t\tID: ble R%d(=%d)<=R%d(=%d), pc=(%d).\n",  regC\_tmp, srcC\_tmp, regA\_tmp, srcA\_tmp, label\_tmp);  }  } else {  if (PRINT\_ID) {  printf("\t\t\tID: ble R%d(=%d)>R%d(=%d), pc++.\n",  regC\_tmp, srcC\_tmp, regA\_tmp, srcA\_tmp);  }  }  if (PRINT\_ID) {  cout << "\t\t\tID: at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  wait();  decode\_valid.write(false);  branch\_valid.write(false);  wait();  break;  case 0x16: // j label  branch\_target\_address.write(longlabel\_tmp);  branch\_valid.write(true);  if (PRINT\_ID) {  printf("\t\t\tID: pc jump to => (%d).", longlabel\_tmp);  cout << " \n\t\t\t : at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  wait();  decode\_valid.write(false);  branch\_valid.write(false);  wait();  break;  case 0xe0: // flush register  if (PRINT\_ID) {  printf("\t\t\tID: flush all registers");  cout << " \n\t\t\t : at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  src\_A.write(0);  src\_B.write(0);  alu\_src.write(0);  alu\_op.write(0);  for (i = 0; i< 16; i++) {  cpu\_reg[i] = 0;  }  wait();  wait();  break;  case 0xf1: // movi R1, #value  if (PRINT\_ID) {  printf("\t\t\tID: R%d=%d",  regC\_tmp, imm\_tmp);  cout << " at CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t-------------------------------" << endl;  }  src\_A.write(imm\_tmp);  src\_B.write(0);  alu\_src.write(regC\_tmp);  alu\_op.write(3);  decode\_valid.write(true);  wait();  decode\_valid.write(false);  wait();  break;  case 0xff: // QUIT  printf("\t\t\tID: - SHUTDOWN - ");  cout << "at CSIM " << sc\_time\_stamp() << endl;  decode\_valid.write(false);  float\_valid.write(false);  wait();  printf("\t\t\tID: - PLEASE WAIT ...... - \n");  cout << "\t\t\t-------------------------------" << endl;  sc\_stop();  printf("\n\n\n////////////////////////////////////////////////////////////////////////////////\n");  wait();  wait();  break;  default :  printf("\t\t\tID: INVALID OPCODE");  cout << " \n\t\t\t : at CSIM " << sc\_time\_stamp() << endl;  wait();  break;  }  next\_pc.write(true);  wait();  } else {  next\_pc.write(true);  wait();  }  }  } // end of entry function |

Decode – модуль, реализующий декодирование извлеченной инструкции и ее выполнение, также сочетает в себе функции модуля поддержки защиты памяти (MPU). Содержит описание регистрового файла размером 16 регистров.

**4.5. Модуль арифметического устройства – exec**

Листинг 4.5. Файл Exec.h

|  |
| --- |
| #include "systemc.h"  #include "directive.h"  struct exec : sc\_module {  sc\_in<bool> in\_valid; // input valid  sc\_in<int> opcode; // opcode from ID  sc\_in<signed int> dina; // operand A  sc\_in<signed int> dinb; // operand B  sc\_in<unsigned> dest; // destination register number  sc\_out<bool> C; // Carry bit  sc\_out<bool> V; // Overflow bit  sc\_out<bool> Z; // Zero bit  sc\_out<signed int> dout; // output data  sc\_out<bool> out\_valid; // output valid  sc\_out<unsigned> destout; // write to which registers?  sc\_in\_clk CLK;  SC\_CTOR(exec) {  SC\_CTHREAD(entry, CLK.pos());  }    void entry();  };  void exec::entry(){  int opcode\_tmp = 0;  int add1\_tmp = 0;  signed int dina\_tmp = 0;  signed int dinb\_tmp = 0;  signed int dout\_tmp = 0;  unsigned int dest\_tmp = 0;  wait(3);  while(1) {  if (in\_valid.read() == true) {  dina\_tmp = dina.read();  dinb\_tmp = dinb.read();  opcode\_tmp = opcode.read();  dest\_tmp = dest.read();  // output MUX  switch (opcode\_tmp) {  case 0: // Stall  dout\_tmp = dout\_tmp;  wait();  break;  case 1: // add with carry  dout\_tmp = dina\_tmp + dinb\_tmp + add1\_tmp;  wait();  break;  case 2: // sub with carry  dout\_tmp = dina\_tmp - dinb\_tmp - add1\_tmp;  wait();  break;  case 3: // add without carry  dout\_tmp = dina\_tmp + dinb\_tmp;  wait();  break;  case 4: // sub without carry  dout\_tmp = dina\_tmp - dinb\_tmp;  wait();  break;  case 5: // multiply assume  dout\_tmp = dina\_tmp \* dinb\_tmp;  wait();  break;  case 6: // divide assume  if (dinb\_tmp == 0) {  printf("Division Exception - Divide by zero \n");  } else {  dout\_tmp = dina\_tmp / dinb\_tmp;  }  wait();  break;  case 7: // bitwise NAND  dout\_tmp = ~(dina\_tmp & dinb\_tmp);  wait();  break;  case 8: // bitwise AND  dout\_tmp = dina\_tmp & dinb\_tmp;  wait();  break;  case 9: // bitwise OR  dout\_tmp = dina\_tmp | dinb\_tmp;  wait();  break;  case 10: // bitwise XOR  dout\_tmp = dina\_tmp ^ dinb\_tmp;  wait();  break;  case 11: // bitwise complement  dout\_tmp = ~ dina\_tmp;  wait();  break;  case 12: // left shift  dout\_tmp = dina\_tmp << dinb\_tmp;  wait();  break;  case 13: // right shift  dout\_tmp = dina\_tmp >> dinb\_tmp;  wait();  break;  case 14: // modulo  dout\_tmp = dina\_tmp % dinb\_tmp;  wait();  break;  default:  printf("ALU: Bad Opcode %d.\n",opcode\_tmp);  break;  }  dout.write(dout\_tmp);  out\_valid.write(true);  destout.write(dest\_tmp);  if (dout\_tmp == 0) {  Z.write(true);  } else {  Z.write(false);  }  if (dout\_tmp > 2^32) {  V.write(true);  }else {  V.write(false);  }  if (PRINT\_ALU) {  printf("\t\t\t\t\t\t\t-------------------------------\n");  cout << "\t\t\t\t\t\t\tALU :" << " op= " << opcode\_tmp  << " A= " << dina\_tmp << " B= " << dinb\_tmp << endl;  cout << "\t\t\t\t\t\t\tALU :" << " R= " << dout\_tmp << "-> R" << dest\_tmp;  cout << " at CSIM " << sc\_time\_stamp() << endl;  printf("\t\t\t\t\t\t\t-------------------------------\n");  }  wait();  out\_valid.write(false);  wait();  } else {  wait();  }    }  } |

**4.6. Модуль прямого доступа к памяти - DMA**

Листинг 4.6. Файл dma.h

|  |
| --- |
| #include "systemc.h"  #include "directive.h"  struct dma : sc\_module {  sc\_in<unsigned > addr; // address  sc\_in<signed> data\_in\_dma; // input data  sc\_in<bool> i\_d\_cache; // icache or dcache  sc\_in<bool> valid\_data; // input data valid  sc\_in<bool> read\_write; // read or write  sc\_out<signed> data\_out; // data from mem  sc\_out<unsigned> imem\_data;  sc\_out<bool> imem\_write\_read;  sc\_out<unsigned> imem\_addr;  sc\_out<bool> icache\_access;  sc\_out<bool> dcache\_access;  sc\_out<bool> dma\_access;  sc\_out<bool> stall\_fetch;  sc\_in<unsigned> idata\_in\_cpu;  sc\_in<bool> imem\_valid;  sc\_in<bool> busy;  sc\_out<signed> dmem\_data;  sc\_out<bool> dmem\_write\_read;  sc\_out<unsigned> dmem\_addr;  sc\_in<signed> ddata\_in\_cpu;  sc\_in<bool> dmem\_valid;  sc\_in\_clk CLK;  // Parameter  unsigned addr\_var;  unsigned data\_in\_var;  bool where\_var;  bool read\_write\_var;  unsigned data\_out\_var;  int wait\_cycles; // cycles # it takes to access dmemory  void init\_param(int given\_delay\_cycles) {  wait\_cycles = given\_delay\_cycles;  }  //Constructor  SC\_CTOR(dma) {  SC\_CTHREAD(entry, CLK.pos());  addr\_var=0;  data\_in\_var=0;  where\_var=false;  read\_write\_var=false;  data\_out\_var=0;  }  // Process functionality in member function below  void entry();  };  void dma::entry()  {  while (true) {  do {  wait();  } while ( !(valid\_data.read() == true) );  addr\_var=addr.read();  data\_in\_var=data\_in\_dma.read();  where\_var=i\_d\_cache.read();  read\_write\_var=read\_write.read();    if(where\_var==true)// icache access  {  stall\_fetch=true;  do{ wait(); } while ( !(busy.read() == false) );  icache\_access=true;  imem\_write\_read=read\_write\_var;  imem\_addr=addr\_var;  if (read\_write\_var==true)  {  imem\_data=data\_in\_var;  if (PRINT\_DMA) {  cout << "-------------------------------" << endl;  printf("DMA: data=0x%x(%d) to icache[%x]",data\_in\_var, data\_in\_var, addr\_var);  cout.setf(ios::dec,ios::basefield);  cout << " at CSIM " << sc\_time\_stamp() << endl;  cout << "-------------------------------" << endl;  }  wait(wait\_cycles);  icache\_access=false;  stall\_fetch=false;  imem\_write\_read=0;  imem\_addr=0;  imem\_data=0;  wait();  }  else  {  do{ wait(); } while ( !(imem\_valid.read() == true) );  data\_out = idata\_in\_cpu.read();  if (PRINT\_DMA) {  cout << "-------------------------------" << endl;  printf("DMA: data=0x%x(%d) fr icache", idata\_in\_cpu.read(), idata\_in\_cpu.read());  cout.setf(ios::dec,ios::basefield);  cout << " at CSIM " << sc\_time\_stamp() << endl;  cout << "-------------------------------" << endl;  icache\_access=false;  stall\_fetch=false;  imem\_write\_read=0;  imem\_addr=0;  }  }  }// dcache access  else  {  dcache\_access=true;  dmem\_write\_read=read\_write\_var;  dmem\_addr=addr\_var;  if (read\_write\_var==true)  {  dmem\_data=data\_in\_var;  if (PRINT\_DMA) {  cout << "-------------------------------" << endl;  printf("DMA: data=0x%x(%d) to dcache[%x]",data\_in\_var, data\_in\_var, addr\_var);  cout.setf(ios::dec,ios::basefield);  cout << " at CSIM " << sc\_time\_stamp() << endl;  cout << "-------------------------------" << endl;  }  wait(wait\_cycles);  dcache\_access=false;  dmem\_write\_read=0;  dmem\_addr=0;  dmem\_data=0;  wait();  }  else  {  dma\_access=true;  do{ wait(); } while ( !(dmem\_valid.read() == true) );  data\_out = ddata\_in\_cpu.read();  if (PRINT\_DMA) {  cout << "-------------------------------" << endl;  printf("DMA: data=0x%x(%d) fr dcache", ddata\_in\_cpu.read(), ddata\_in\_cpu.read());  cout.setf(ios::dec,ios::basefield);  cout << " at CSIM " << sc\_time\_stamp() << endl;  cout << "-------------------------------" << endl;  dcache\_access=false;  dma\_access=false;  dmem\_write\_read=0;  dmem\_addr=0;  }  }  }  do {  wait();  } while ( !(valid\_data.read() == false) );  }  } // end of entry function |

**4.7. Модуль интерфейса ввода/вывода – GPIO**

Листинг 4.7. Файл GPIO.h

|  |
| --- |
| #include "systemc.h"  #include "directive.h"  struct gpio : sc\_module {  sc\_in<signed> datain; // input data from processor  sc\_in<signed> gpio\_data\_in; //input data from gpio  sc\_in<bool> cs; // chip select  sc\_in<bool> we; // write enable  sc\_out<signed> dataout; // data out to processor  sc\_out<signed> gpio\_dataout; //output data from gpio  sc\_out<bool> out\_valid; // output valid  sc\_in\_clk CLK;  // Parameter  unsigned gpio\_out; // data out  unsigned gpio\_in; // data in  int wait\_cycles; // cycles # it takes to access  void init\_param(int given\_delay\_cycles) {  wait\_cycles = given\_delay\_cycles;  }    //Constructor  SC\_CTOR(gpio) {  SC\_CTHREAD(entry, CLK.pos());  gpio\_in=0;  gpio\_out=0;  }  // Process functionality in member function below  void entry();  };  void gpio::entry()  {  while (true) {  do {  wait();  gpio\_dataout.write(gpio\_out);  gpio\_in=gpio\_data\_in.read();  } while ( !(cs == true) );  if (we.read() == true) { // Write operation  wait();  out\_valid.write(false);  gpio\_out = datain.read();  if (PRINT\_GPIO) {  cout << "\t\t\t\t\t\t\t-------------------------------" << endl;  printf("\t\t\t\t\t\t\tGPIO :wr %x->gpio out\n", gpio\_out);  cout << "\t\t\t\t\t\t\tat CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t\t\t\t\t-------------------------------" << endl;  }  wait();  }  else { // Read operation  wait();  dataout.write(gpio\_in);  out\_valid.write(true);  if (PRINT\_DCU) {  cout << "\t\t\t\t\t\t\t-------------------------------" << endl;  printf("\t\t\t\t\t\t\tGPIO :ld 0x%x(%d)<-gpio in\n", gpio\_in, gpio\_in);  cout << "\t\t\t\t\t\t\tat CSIM " << sc\_time\_stamp() << endl;  cout << "\t\t\t\t\t\t\t-------------------------------" << endl;  }  wait();  out\_valid.write(false);  wait();  }  }  } // end of entry function |

**4.8. Модуль контроллера прерываний - PIC**

Листинг 4.7. Файл pic.h

|  |
| --- |
| #include "systemc.h"  struct pic : sc\_module {  sc\_in<bool> ireq0; // interrupt request 0  sc\_in<bool> ireq1; // interrupt request 1  sc\_in<bool> ireq2; // interrupt request 2  sc\_in<bool> ireq3; // interrupt request 3  sc\_in<bool> intack\_cpu; // interrupt acknowledge from CPU  sc\_out<bool> intreq; // interrupt request to CPU  sc\_out<unsigned> vectno; // vector number  //Constructor  SC\_CTOR(pic) {  SC\_METHOD(entry);  dont\_initialize();  sensitive << ireq0 << ireq1 << ireq2 << ireq3 << intack\_cpu;  }  // Process functionality in member function below  void entry();  };  void pic::entry(){  if (ireq0.read() == true) {  intreq.write(true);  vectno.write(0);  } else if (ireq1.read() == true) {  intreq.write(true);  vectno.write(1);  } else if (ireq2.read() == true) {  intreq.write(true);  vectno.write(2);  } else if (ireq3.read() == true) {  intreq.write(true);  vectno.write(3);  }  if (intack\_cpu.read() == true) {  intreq.write(false);  }  } |

**Раздел 5. Модульное тестирование разработанной системы.**

**5.1. Модуль памяти данных – Dcache**

Листинг 5.1. Модульное тестирование Dcache

|  |
| --- |
| #include "directive.h"  #include "systemc.h"  #include "dcache.h"  #include <climits>  #include <cstdlib>  #include <time.h>  #include <limits.h>  int sc\_main(int ac, char \*av[])  {  sc\_core::sc\_report\_handler::set\_actions( "/IEEE\_Std\_1666/deprecated",  sc\_core::SC\_DO\_NOTHING );  sc\_trace\_file \*wf = sc\_create\_vcd\_trace\_file("Processor\_waveform");  const int delay\_cycles = 2;  sc\_signal<signed> dram\_dataout("DRAM\_DATAOUT") ;  sc\_signal<bool> dram\_rd\_valid("DRAM\_RD\_VALID") ;  sc\_signal<bool> mem\_access("MEM\_ACCESS") ;  sc\_signal<unsigned > mem\_address("MEM\_ADDRESS") ;  sc\_signal<bool> mem\_write("MEM\_WRITE") ;  sc\_signal<signed int> dramdata("dramdata");  sc\_trace(wf, dram\_dataout, "DCACHE\_dram\_dataout");  sc\_trace(wf, dram\_rd\_valid, "DCACHE\_dram\_rd\_valid");  sc\_trace(wf, mem\_access, "DCACHE\_mem\_access");  sc\_trace(wf, mem\_address, "DCACHE\_mem\_address");  sc\_trace(wf, mem\_write, "DCACHE\_mem\_write");  sc\_trace(wf, dramdata, "DCACHE\_dramdata");  ////////////////////////////////////////////////////////////////////////////  // MAIN PROGRAM  ////////////////////////////////////////////////////////////////////////////  sc\_clock clk("Clock", 1, SC\_NS, 0.5, 0.0, SC\_NS);  sc\_trace(wf, clk, "CLK");  dcache DCACHE("DCACHE\_BLOCK");  DCACHE.init\_param(delay\_cycles);  DCACHE << dramdata << mem\_access << mem\_write << mem\_address << dram\_dataout  << dram\_rd\_valid << clk;  time\_t tbuffer = time(NULL);  //6ns  sc\_start(6, SC\_NS);  mem\_access=1;  mem\_write=0;  mem\_address=3;  cout << "@" << sc\_time\_stamp() << " Read mem[0x3]\n" << endl;  sc\_start(2, SC\_NS);  //8ns  assert(dram\_rd\_valid.read() == true);  assert(dram\_dataout.read() == 3);  cout << "@" << sc\_time\_stamp() << "mem[0x3]=0x3\n" << endl;  mem\_access=0;  sc\_start(2, SC\_NS);  //10ns  mem\_access=1;  mem\_write=1;  dramdata=0xff;  mem\_address=3;  cout << "@" << sc\_time\_stamp() << " Write mem[0x3]=0xff\n" << endl;  sc\_start(2, SC\_NS);  mem\_access=0;  //12ns  sc\_start(6, SC\_NS);  mem\_access=1;  mem\_write=0;  mem\_address=3;  cout << "@" << sc\_time\_stamp() << " Read mem[0x3]\n" << endl;  sc\_start(2, SC\_NS);  //14ns  assert(dram\_rd\_valid.read() == true);  assert(dram\_dataout.read() == 0xff);  cout << "@" << sc\_time\_stamp() << "mem[0x3]=0xff\n" << endl;  mem\_access=0;  sc\_start(2, SC\_NS);  cout << "Time for simulation = " << (time(NULL) - tbuffer) << endl;  sc\_close\_vcd\_trace\_file(wf);    return 0;  } |

**Результаты выполнения теста:**

Консоль:

|  |
| --- |
| SystemC 2.3.1-Accellera --- Feb 18 2017 01:17:28  Copyright (c) 1996-2014 by all Contributors,  ALL RIGHTS RESERVED  \*\* ALERT \*\* DCU: initialize Data Cache  Info: (I702) default timescale unit used for tracing: 1 ps (Processor\_waveform.vcd)  @6 ns Read mem[0x3]  -------------------------------  DCU :ld 0x3(3)<-mem[3]  at CSIM 7 ns  -------------------------------  @8 nsmem[0x3]=0x3  @10 ns Write mem[0x3]=0xff  -------------------------------  DCU :wr ff->mem[3]  at CSIM 11 ns  -------------------------------  @18 ns Read mem[0x3]  -------------------------------  DCU :ld 0xff(255)<-mem[3]  at CSIM 19 ns  -------------------------------  @20 nsmem[0x3]=0xff  Time for simulation = 0  Для продолжения нажмите любую клавишу . . . |

GTKWave:

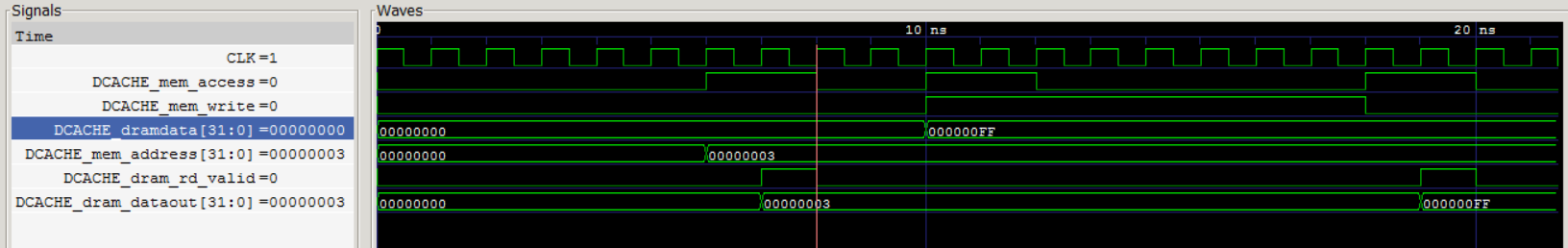


Рис 5.1. Результаты тестирования

**5.2. Модуль памяти инструкций – Icache**

Листинг 5.2. Модульное тестирование Dcache

|  |
| --- |
| #include "directive.h"  #include "systemc.h"  #include "icache.h"  #include <climits>  #include <cstdlib>  #include <time.h>  #include <limits.h>  int sc\_main(int ac, char \*av[])  {  sc\_core::sc\_report\_handler::set\_actions( "/IEEE\_Std\_1666/deprecated",  sc\_core::SC\_DO\_NOTHING );  sc\_trace\_file \*wf = sc\_create\_vcd\_trace\_file("Processor\_waveform");  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ICACHE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  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> icache\_valid("ICACHE\_VALID") ;  const int delay\_cycles = 2;  // Dump the desired signals  sc\_trace(wf, ram\_cs, "ICACHE\_ram\_cs");  sc\_trace(wf, ram\_we, "ICACHE\_ram\_we");  sc\_trace(wf, addr, "ICACHE\_addr");  sc\_trace(wf, ram\_datain, "ICACHE\_ram\_datain");  sc\_trace(wf, ram\_dataout, "ICACHE\_ram\_dataout");  sc\_trace(wf, icache\_valid, "ICACHE\_icache\_valid");  ////////////////////////////////////////////////////////////////////////////  // MAIN PROGRAM  ////////////////////////////////////////////////////////////////////////////  sc\_clock clk("Clock", 1, SC\_NS, 0.5, 0.0, SC\_NS);  sc\_trace(wf, clk, "CLK");    icache ICACHE("ICACHE\_BLOCK");  ICACHE.init\_param(delay\_cycles);  ICACHE << ram\_datain << ram\_cs << ram\_we << addr << ram\_dataout  << icache\_valid << clk;  time\_t tbuffer = time(NULL);  //6ns  sc\_start(6, SC\_NS);  ram\_cs=1;  ram\_we=0;  addr=3;  cout << "@" << sc\_time\_stamp() << " Read mem[0x3]\n" << endl;  sc\_start(2, SC\_NS);  //8ns  assert(icache\_valid.read() == true);  assert(ram\_dataout.read() == 0xfffffff0);  cout << "@" << sc\_time\_stamp() << "mem[0x3]=0xfffffff0\n" << endl;  ram\_cs=0;  sc\_start(2, SC\_NS);  //10ns  ram\_cs=1;  ram\_we=1;  ram\_datain=0xff;  addr=3;  cout << "@" << sc\_time\_stamp() << " Write mem[0x3]=0xff\n" << endl;  sc\_start(2, SC\_NS);  ram\_cs=0;  //12ns  sc\_start(6, SC\_NS);  ram\_cs=1;  ram\_we=0;  addr=3;  cout << "@" << sc\_time\_stamp() << " Read mem[0x3]\n" << endl;  sc\_start(2, SC\_NS);  //14ns  assert(icache\_valid.read() == true);  assert(ram\_dataout.read() == 0xff);  cout << "@" << sc\_time\_stamp() << "mem[0x3]=0xff\n" << endl;  ram\_cs=0;  sc\_start(2, SC\_NS);  cout << "Time for simulation = " << (time(NULL) - tbuffer) << endl;  sc\_close\_vcd\_trace\_file(wf);  return 0;  } |

**Результаты выполнения теста:**

Консоль:

|  |
| --- |
| SystemC 2.3.1-Accellera --- Feb 18 2017 01:17:28  Copyright (c) 1996-2014 by all Contributors,  ALL RIGHTS RESERVED  Info: (I702) default timescale unit used for tracing: 1 ps (Processor\_waveform.vcd)  @6 ns Read mem[0x3]  ------------------------  ICU: fetching mem[3]  ICU: (fffffff0) at CSIM 7 ns  ------------------------  @8 nsmem[0x3]=0xfffffff0  @10 ns Write mem[0x3]=0xff  ------------------------  ICU: write to mem[3]=ff  ------------------------  @18 ns Read mem[0x3]  ------------------------  ICU: fetching mem[3]  ICU: (ff) at CSIM 19 ns  ------------------------  @20 nsmem[0x3]=0xff  Time for simulation = 0  Для продолжения нажмите любую клавишу . . . |

GTKWave:

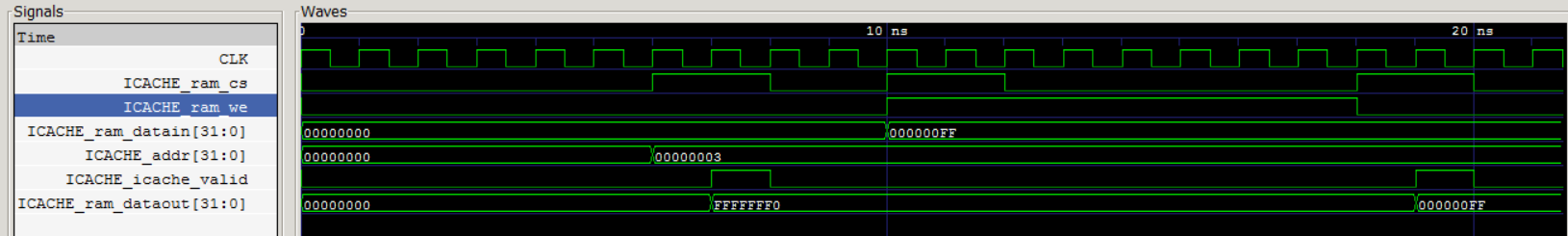


Рис 5.2. Результаты тестирования

**5.3. Модуль контроллер прерываний – PIC**

Листинг 5.3. Модульное тестирование PIC

|  |
| --- |
| #include "directive.h"  #include "systemc.h"  #include "pic.h"  #include <climits>  #include <cstdlib>  #include <time.h>  #include <limits.h>  int sc\_main(int ac, char \*av[])  {  sc\_core::sc\_report\_handler::set\_actions( "/IEEE\_Std\_1666/deprecated",  sc\_core::SC\_DO\_NOTHING );  sc\_trace\_file \*wf = sc\_create\_vcd\_trace\_file("Processor\_waveform");  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PIC \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  sc\_signal<bool> ireq0("IREQ0") ;  sc\_signal<bool> ireq1("IREQ1") ;  sc\_signal<bool> ireq2("IREQ2") ;  sc\_signal<bool> ireq3("IREQ3") ;  sc\_signal<bool> intreq("INTREQ") ;  sc\_signal<unsigned> vectno("VECTNO") ;  sc\_signal<bool> intack\_cpu("INTACK\_CPU") ;  sc\_trace(wf, ireq0, "PIC\_ireq0");  sc\_trace(wf, ireq1, "PIC\_ireq1");  sc\_trace(wf, ireq2, "PIC\_ireq2");  sc\_trace(wf, ireq3, "PIC\_ireq3");  sc\_trace(wf, intreq, "PIC\_intreq");  sc\_trace(wf, vectno, "PIC\_vectno");  sc\_trace(wf, intack\_cpu, "PIC\_intack\_cpu");    ////////////////////////////////////////////////////////////////////////////  // MAIN PROGRAM  ////////////////////////////////////////////////////////////////////////////  sc\_clock clk("Clock", 1, SC\_NS, 0.5, 0.0, SC\_NS);  sc\_trace(wf, clk, "CLK");      pic APIC("PIC\_BLOCK");  APIC << ireq0 << ireq1 << ireq2 << ireq3 <<intack\_cpu  << intreq << vectno;  time\_t tbuffer = time(NULL);  //6ns  sc\_start(6, SC\_NS);  ireq0=1;  cout << "@" << sc\_time\_stamp() << " Interrapt 0\n" << endl;  sc\_start(2, SC\_NS);  ireq0=0;  //8ns  assert(intreq.read() == true);  assert(vectno.read() == 0x0);  cout << "@" << sc\_time\_stamp() << " Sucess interrapt 0\n" << endl;  intack\_cpu=1;  sc\_start(2, SC\_NS);  //10ns  assert(intreq.read() == false);  intack\_cpu=0;  //6ns  sc\_start(6, SC\_NS);  ireq1=1;  cout << "@" << sc\_time\_stamp() << " Interrapt 1\n" << endl;  sc\_start(2, SC\_NS);  ireq1=0;  //8ns  assert(intreq.read() == true);  assert(vectno.read() == 0x1);  cout << "@" << sc\_time\_stamp() << " Sucess interrapt 1\n" << endl;  intack\_cpu=1;  sc\_start(2, SC\_NS);  //10ns  assert(intreq.read() == false);  intack\_cpu=0;  //6ns  sc\_start(6, SC\_NS);  ireq2=1;  cout << "@" << sc\_time\_stamp() << " Interrapt 2\n" << endl;  sc\_start(2, SC\_NS);  ireq2=0;  //8ns  assert(intreq.read() == true);  assert(vectno.read() == 0x2);  cout << "@" << sc\_time\_stamp() << " Sucess interrapt 2\n" << endl;  intack\_cpu=1;  sc\_start(2, SC\_NS);  //10ns  assert(intreq.read() == false);    intack\_cpu=0;  //6ns  sc\_start(6, SC\_NS);  ireq3=1;  cout << "@" << sc\_time\_stamp() << " Interrapt 3\n" << endl;  sc\_start(2, SC\_NS);  ireq3=0;  //8ns  assert(intreq.read() == true);  assert(vectno.read() == 0x3);  cout << "@" << sc\_time\_stamp() << " Sucess interrapt 3\n" << endl;  intack\_cpu=1;  sc\_start(2, SC\_NS);  //10ns  assert(intreq.read() == false);  sc\_close\_vcd\_trace\_file(wf);  return 0;  } |

**Результаты выполнения теста:**

Консоль:

|  |
| --- |
| SystemC 2.3.1-Accellera --- Feb 18 2017 01:17:28  Copyright (c) 1996-2014 by all Contributors,  ALL RIGHTS RESERVED  Info: (I702) default timescale unit used for tracing: 1 ps (Processor\_waveform.vcd)  @6 ns Interrapt 0  @8 ns Sucess interrapt 0  @16 ns Interrapt 1  @18 ns Sucess interrapt 1  @26 ns Interrapt 2  @28 ns Sucess interrapt 2  @36 ns Interrapt 3  @38 ns Sucess interrapt 3  Для продолжения нажмите любую клавишу . . . |

GTKWave:

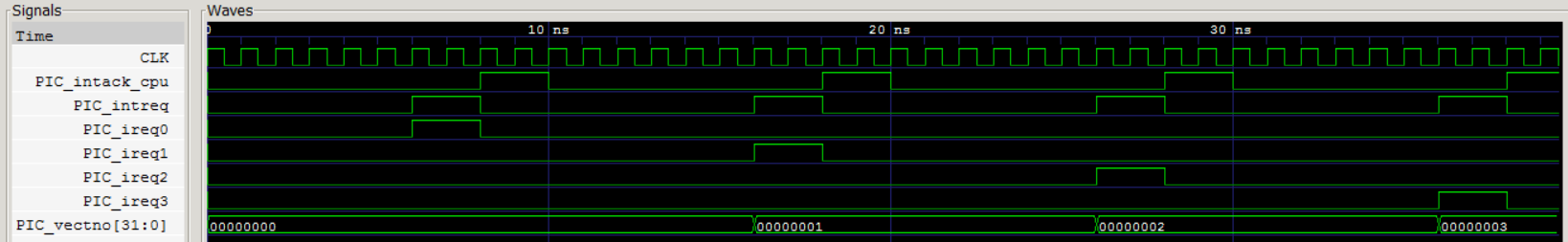


Рис 5.3. Результаты тестирования

**5.4. Модуль прямого доступа к памяти – DMA**

Листинг 5.4. Модульное тестирование DMA

|  |
| --- |
| #include "directive.h"  #include "systemc.h"  #include "icache.h"  #include "fetch.h"  #include "decode.h"  #include "exec.h"  #include "floating.h"  #include "dcache.h"  #include "pic.h"  #include "GPIO.h"  #include "dma.h"  #include <climits>  #include <cstdlib>  #include <time.h>  #include <limits.h>  int sc\_main(int ac, char \*av[])  {  sc\_core::sc\_report\_handler::set\_actions( "/IEEE\_Std\_1666/deprecated",  sc\_core::SC\_DO\_NOTHING );  sc\_trace\_file \*wf = sc\_create\_vcd\_trace\_file("Processor\_waveform");  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ICACHE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  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> icache\_valid("ICACHE\_VALID") ;  const int delay\_cycles = 2;  // Dump the desired signals  sc\_trace(wf, ram\_cs, "ICACHE\_ram\_cs");  sc\_trace(wf, ram\_we, "ICACHE\_ram\_we");  sc\_trace(wf, addr, "ICACHE\_addr");  sc\_trace(wf, ram\_datain, "ICACHE\_ram\_datain");  sc\_trace(wf, ram\_dataout, "ICACHE\_ram\_dataout");  sc\_trace(wf, icache\_valid, "ICACHE\_icache\_valid");  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Fetch \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // IFU ramdata = ram\_dataout  sc\_signal<unsigned > branch\_target\_address("BRANCH\_TARGET\_ADDRESS") ;  sc\_signal<bool> next\_pc("NEXT\_PC") ;  sc\_signal<bool> branch\_valid("BRANCH\_VALID") ;  sc\_signal<bool> stall\_fetch("STALL\_FETCH") ;  // IFU ram\_cs = ram\_cs  // IFU ram\_we = ram\_we  // IFU address = addr  sc\_signal<unsigned> instruction("INSTRUCTION") ;  sc\_signal<bool> instruction\_valid("INSTRUCTION\_VALID") ;  sc\_signal<bool> busy("busy") ;  sc\_trace(wf, branch\_target\_address, "FETCH\_branch\_target\_address");  sc\_trace(wf, next\_pc, "FETCH\_next\_pc");  sc\_trace(wf, branch\_valid, "FETCH\_branch\_valid");  sc\_trace(wf, stall\_fetch, "FETCH\_stall\_fetch");  sc\_trace(wf, instruction, "FETCH\_instruction");  sc\_trace(wf, instruction\_valid, "FETCH\_instruction\_valid");  sc\_trace(wf, busy, "FETCH\_busy");  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Decode \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // ID instruction = instruction  // ID instruction = instruction\_valid  // ID destreg\_write = out\_valid  // ID destreg\_write\_src = destout  // ID alu\_dataout = dout from EXEC  sc\_signal<signed> dram\_dataout("DRAM\_DATAOUT") ;  sc\_signal<bool> dram\_rd\_valid("DRAM\_RD\_VALID") ;  // ID next\_pc = next\_pc  // ID branch\_valid = branch\_valid  // ID branch\_target\_address = branch\_target\_address  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") ;  // ID stall\_fetch = stall\_fetch  sc\_signal<bool> decode\_valid("DECODE\_VALID") ;  sc\_signal<bool> float\_valid("FLOAT\_VALID") ;  sc\_signal<bool> gpio\_access("GPIO\_ACCESS");  sc\_signal<signed int> dramdata("dramdata");  sc\_trace(wf, dram\_dataout, "DECODE\_dram\_dataout");  sc\_trace(wf, dram\_rd\_valid, "DECODE\_dram\_rd\_valid");  sc\_trace(wf, mem\_access, "DECODE\_mem\_access");  sc\_trace(wf, mem\_address, "DECODE\_mem\_address");  sc\_trace(wf, alu\_op, "DECODE\_alu\_op");  sc\_trace(wf, mem\_write, "DECODE\_mem\_write");  sc\_trace(wf, alu\_src, "DECODE\_alu\_src");  sc\_trace(wf, reg\_write, "DECODE\_reg\_write");  sc\_trace(wf, src\_A, "DECODE\_src\_A");  sc\_trace(wf, src\_B, "DECODE\_src\_B");  sc\_trace(wf, decode\_valid, "DECODE\_decode\_valid");  sc\_trace(wf, float\_valid, "DECODE\_float\_valid");  sc\_trace(wf, gpio\_access, "DECODE\_gpio\_access");  sc\_trace(wf, dramdata, "DECODE\_dramdata");  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* DCACHE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //DECODE sc\_signal<signed int> src\_A("SRC\_A") ;  //DECODE sc\_signal<bool> mem\_access("MEM\_ACCESS") ;  //DECODE sc\_signal<bool> mem\_write("MEM\_WRITE") ;  //DECODE sc\_signal<unsigned > mem\_address("MEM\_ADDRESS") ;  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Execute \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // EXEC in\_valid = decode\_valid  sc\_signal<bool> in\_valid("IN\_VALID") ;  // EXEC opcode = alu\_op  // 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") ;  sc\_trace(wf, in\_valid, "EXEC\_in\_valid");  sc\_trace(wf, c, "EXEC\_c");  sc\_trace(wf, v, "EXEC\_v");  sc\_trace(wf, z, "EXEC\_z");  sc\_trace(wf, dout, "EXEC\_dout");  sc\_trace(wf, out\_valid, "EXEC\_out\_valid");  sc\_trace(wf, destout, "EXEC\_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") ;  sc\_trace(wf, fdout, "FLOATING\_fdout");  sc\_trace(wf, fout\_valid, "FLOATING\_fout\_valid");  sc\_trace(wf, fdestout, "FLOATING\_fdestout");  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PIC \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  sc\_signal<bool> ireq0("IREQ0") ;  sc\_signal<bool> ireq1("IREQ1") ;  sc\_signal<bool> ireq2("IREQ2") ;  sc\_signal<bool> ireq3("IREQ3") ;  sc\_signal<bool> intreq("INTREQ") ;  sc\_signal<unsigned> vectno("VECTNO") ;  sc\_signal<bool> intack\_cpu("INTACK\_CPU") ;  sc\_trace(wf, ireq0, "PIC\_ireq0");  sc\_trace(wf, ireq1, "PIC\_ireq1");  sc\_trace(wf, ireq2, "PIC\_ireq2");  sc\_trace(wf, ireq3, "PIC\_ireq3");  sc\_trace(wf, intreq, "PIC\_intreq");  sc\_trace(wf, vectno, "PIC\_vectno");  sc\_trace(wf, intack\_cpu, "PIC\_intack\_cpu");  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* GPIO \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  sc\_signal<signed> gpio\_data\_in("GPIO\_DATA\_IN"); //input data from gpio  sc\_signal<signed> gpio\_dataout("GPIO\_DATA\_OUT"); //output data from gpio  sc\_trace(wf, gpio\_data\_in, "GPIO\_gpio\_data\_in");  sc\_trace(wf, gpio\_dataout, "GPIO\_gpio\_dataout");    // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* DMA \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  sc\_signal<unsigned > dma\_addr("addr"); // address  sc\_signal<signed> dma\_data\_in\_dma("data\_in\_dma"); // input data  sc\_signal<bool> dma\_i\_d\_cache("i\_d\_cache"); // icache or dcache  sc\_signal<bool> dma\_valid\_data("valid\_data"); // input data valid  sc\_signal<bool> dma\_read\_write("read\_write"); // read or write  sc\_signal<signed> dma\_data\_out("data\_out"); // data from mem  sc\_signal<bool> dma\_access("dma\_access");  sc\_trace(wf, dma\_addr, "DMA\_addr");  sc\_trace(wf, dma\_data\_in\_dma, "DMA\_data\_in\_dma");  sc\_trace(wf, dma\_i\_d\_cache, "DMA\_i\_d\_cache");  sc\_trace(wf, dma\_valid\_data, "DMA\_valid\_data");  sc\_trace(wf, dma\_read\_write, "DMA\_read\_write");  sc\_trace(wf, dma\_data\_out, "DMA\_data\_out");  sc\_trace(wf, dma\_access, "DMA\_dma\_access");  ////////////////////////////////////////////////////////////////////////////  // MAIN PROGRAM  ////////////////////////////////////////////////////////////////////////////  sc\_clock clk("Clock", 1, SC\_NS, 0.5, 0.0, SC\_NS);  sc\_trace(wf, clk, "CLK");  fetch IFU("FETCH\_BLOCK");  IFU.init\_param(delay\_cycles);  IFU << ram\_dataout << branch\_target\_address << next\_pc << branch\_valid  << stall\_fetch << intreq << vectno << icache\_valid << ram\_cs << ram\_we  << addr << instruction << instruction\_valid << intack\_cpu << busy << clk;  decode IDU("DECODE\_BLOCK");  IDU << instruction << instruction\_valid << out\_valid << destout << dout  << dram\_dataout << dram\_rd\_valid << fdout << fout\_valid << fdestout  << next\_pc << branch\_valid << branch\_target\_address << mem\_access  << mem\_address << alu\_op << mem\_write << alu\_src << src\_A << src\_B  << decode\_valid << float\_valid << gpio\_access << dramdata << dma\_access << ireq0 <<clk;  exec IEU("EXEC\_BLOCK");  IEU << decode\_valid << alu\_op << src\_A << src\_B << alu\_src << c << v << z  << dout << out\_valid << destout << clk;  floating FPU("FLOAT\_BLOCK");  FPU << float\_valid << alu\_op << src\_A << src\_B << alu\_src << fdout  << fout\_valid << fdestout << clk;  icache ICACHE("ICACHE\_BLOCK");  ICACHE.init\_param(delay\_cycles);  ICACHE << ram\_datain << ram\_cs << ram\_we << addr << ram\_dataout  << icache\_valid << clk;  dcache DCACHE("DCACHE\_BLOCK");  DCACHE.init\_param(delay\_cycles);  DCACHE << dramdata << mem\_access << mem\_write << mem\_address << dram\_dataout  << dram\_rd\_valid << clk;  gpio GPIO("GPIO\_BLOCK");  GPIO.init\_param(delay\_cycles);  GPIO << src\_A << gpio\_data\_in << gpio\_access << mem\_write << dram\_dataout << gpio\_dataout  << dram\_rd\_valid << clk;  dma DMA("DMA\_BLOCK");  DMA.init\_param(delay\_cycles);  DMA << dma\_addr << dma\_data\_in\_dma << dma\_i\_d\_cache << dma\_valid\_data << dma\_read\_write << dma\_data\_out  << ram\_datain << ram\_we << addr << ram\_cs << mem\_access << dma\_access << stall\_fetch << ram\_dataout << icache\_valid << busy  << dramdata << mem\_write << mem\_address << dram\_dataout << dram\_rd\_valid << clk;  pic APIC("PIC\_BLOCK");  APIC << ireq0 << ireq1 << ireq2 << ireq3 <<intack\_cpu  << intreq << vectno;  time\_t tbuffer = time(NULL);  gpio\_data\_in = 0xff;  sc\_start(6, SC\_NS);  dma\_addr=3;  dma\_i\_d\_cache=1;  dma\_read\_write=1;  dma\_valid\_data=1;  dma\_data\_in\_dma=0xffff;  cout << "@" << sc\_time\_stamp() << " Write icache mem[0x3]\n" << endl;  sc\_start(8, SC\_NS);  //14ns  assert(addr.read() == 3);  assert(ram\_cs.read() == 1);  assert(stall\_fetch.read() == 1);  assert(ram\_datain.read() == 0xffff);    dma\_addr=3;  dma\_i\_d\_cache=1;  dma\_read\_write=0;  dma\_valid\_data=0;  dma\_data\_in\_dma=0xffff;  sc\_start(4, SC\_NS);  //18ns  dma\_addr=3;  dma\_i\_d\_cache=1;  dma\_read\_write=0;  dma\_valid\_data=1;  sc\_start(4, SC\_NS);  dma\_valid\_data=0;  sc\_start(6, SC\_NS);  //28ns  assert(dma\_data\_out.read() == 0xffff);  //18ns  dma\_addr=3;  dma\_i\_d\_cache=0;  dma\_read\_write=1;  dma\_valid\_data=1;  dma\_data\_in\_dma=0xeeee;  sc\_start(4, SC\_NS);  dma\_valid\_data=0;  sc\_start(8, SC\_NS);  dma\_addr=3;  dma\_i\_d\_cache=0;  dma\_read\_write=0;  dma\_valid\_data=1;  sc\_start(4, SC\_NS);    cout << "Time for simulation = " << (time(NULL) - tbuffer) << endl;  sc\_close\_vcd\_trace\_file(wf);    return 0;  } |

**Результаты выполнения теста:**

GTKWave:

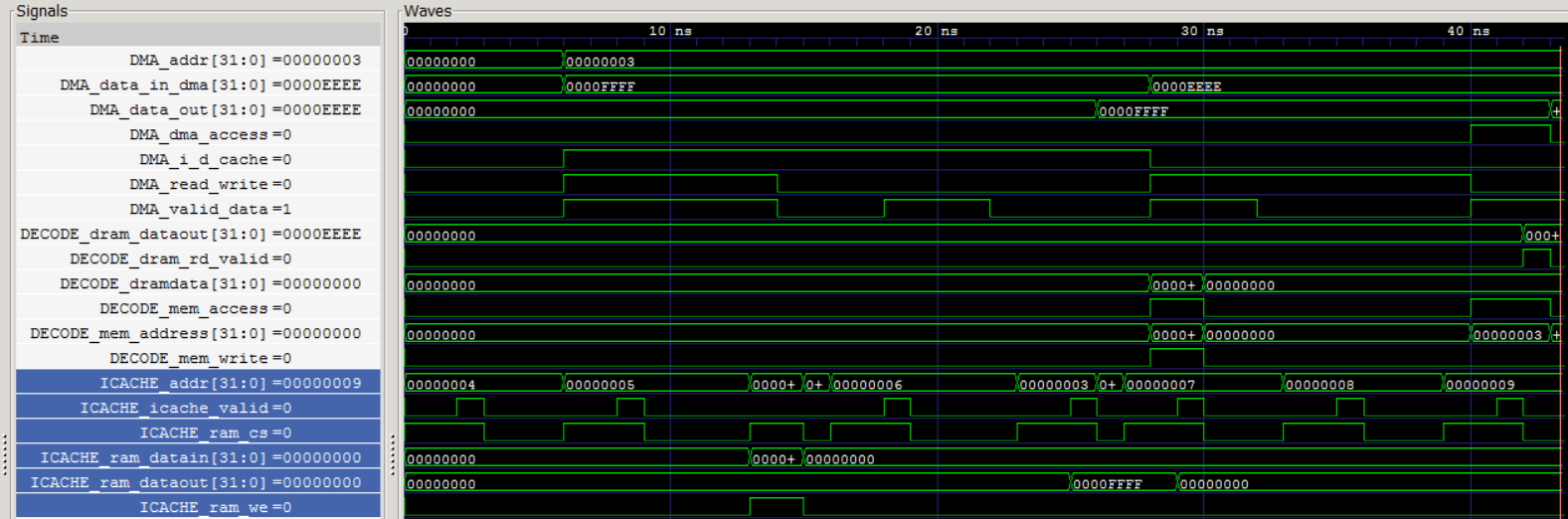


Рис 5.4. Результаты тестирования

**5.5. Модуль интерфейс ввода/вывода - GPIO**

Листинг 5.5. Модульное тестирование GPIO

|  |
| --- |
| #include "directive.h"  #include "systemc.h"  #include "icache.h"  #include "fetch.h"  #include "decode.h"  #include "exec.h"  #include "floating.h"  #include "dcache.h"  #include "pic.h"  #include "GPIO.h"  #include "dma.h"  #include <climits>  #include <cstdlib>  #include <time.h>  #include <limits.h>  int sc\_main(int ac, char \*av[])  {  sc\_core::sc\_report\_handler::set\_actions( "/IEEE\_Std\_1666/deprecated",  sc\_core::SC\_DO\_NOTHING );  sc\_trace\_file \*wf = sc\_create\_vcd\_trace\_file("Processor\_waveform");  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ICACHE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  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> icache\_valid("ICACHE\_VALID") ;  const int delay\_cycles = 2;  // Dump the desired signals  sc\_trace(wf, ram\_cs, "ICACHE\_ram\_cs");  sc\_trace(wf, ram\_we, "ICACHE\_ram\_we");  sc\_trace(wf, addr, "ICACHE\_addr");  sc\_trace(wf, ram\_datain, "ICACHE\_ram\_datain");  sc\_trace(wf, ram\_dataout, "ICACHE\_ram\_dataout");  sc\_trace(wf, icache\_valid, "ICACHE\_icache\_valid");  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Fetch \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // IFU ramdata = ram\_dataout  sc\_signal<unsigned > branch\_target\_address("BRANCH\_TARGET\_ADDRESS") ;  sc\_signal<bool> next\_pc("NEXT\_PC") ;  sc\_signal<bool> branch\_valid("BRANCH\_VALID") ;  sc\_signal<bool> stall\_fetch("STALL\_FETCH") ;  // IFU ram\_cs = ram\_cs  // IFU ram\_we = ram\_we  // IFU address = addr  sc\_signal<unsigned> instruction("INSTRUCTION") ;  sc\_signal<bool> instruction\_valid("INSTRUCTION\_VALID") ;  sc\_signal<bool> busy("busy") ;  sc\_trace(wf, branch\_target\_address, "FETCH\_branch\_target\_address");  sc\_trace(wf, next\_pc, "FETCH\_next\_pc");  sc\_trace(wf, branch\_valid, "FETCH\_branch\_valid");  sc\_trace(wf, stall\_fetch, "FETCH\_stall\_fetch");  sc\_trace(wf, instruction, "FETCH\_instruction");  sc\_trace(wf, instruction\_valid, "FETCH\_instruction\_valid");  sc\_trace(wf, busy, "FETCH\_busy");  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Decode \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // ID instruction = instruction  // ID instruction = instruction\_valid  // ID destreg\_write = out\_valid  // ID destreg\_write\_src = destout  // ID alu\_dataout = dout from EXEC  sc\_signal<signed> dram\_dataout("DRAM\_DATAOUT") ;  sc\_signal<bool> dram\_rd\_valid("DRAM\_RD\_VALID") ;  // ID next\_pc = next\_pc  // ID branch\_valid = branch\_valid  // ID branch\_target\_address = branch\_target\_address  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") ;  // ID stall\_fetch = stall\_fetch  sc\_signal<bool> decode\_valid("DECODE\_VALID") ;  sc\_signal<bool> float\_valid("FLOAT\_VALID") ;  sc\_signal<bool> gpio\_access("GPIO\_ACCESS");  sc\_signal<signed int> dramdata("dramdata");  sc\_trace(wf, dram\_dataout, "DECODE\_dram\_dataout");  sc\_trace(wf, dram\_rd\_valid, "DECODE\_dram\_rd\_valid");  sc\_trace(wf, mem\_access, "DECODE\_mem\_access");  sc\_trace(wf, mem\_address, "DECODE\_mem\_address");  sc\_trace(wf, alu\_op, "DECODE\_alu\_op");  sc\_trace(wf, mem\_write, "DECODE\_mem\_write");  sc\_trace(wf, alu\_src, "DECODE\_alu\_src");  sc\_trace(wf, reg\_write, "DECODE\_reg\_write");  sc\_trace(wf, src\_A, "DECODE\_src\_A");  sc\_trace(wf, src\_B, "DECODE\_src\_B");  sc\_trace(wf, decode\_valid, "DECODE\_decode\_valid");  sc\_trace(wf, float\_valid, "DECODE\_float\_valid");  sc\_trace(wf, gpio\_access, "DECODE\_gpio\_access");  sc\_trace(wf, dramdata, "DECODE\_dramdata");  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* DCACHE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //DECODE sc\_signal<signed int> src\_A("SRC\_A") ;  //DECODE sc\_signal<bool> mem\_access("MEM\_ACCESS") ;  //DECODE sc\_signal<bool> mem\_write("MEM\_WRITE") ;  //DECODE sc\_signal<unsigned > mem\_address("MEM\_ADDRESS") ;  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Execute \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // EXEC in\_valid = decode\_valid  sc\_signal<bool> in\_valid("IN\_VALID") ;  // EXEC opcode = alu\_op  // 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") ;  sc\_trace(wf, in\_valid, "EXEC\_in\_valid");  sc\_trace(wf, c, "EXEC\_c");  sc\_trace(wf, v, "EXEC\_v");  sc\_trace(wf, z, "EXEC\_z");  sc\_trace(wf, dout, "EXEC\_dout");  sc\_trace(wf, out\_valid, "EXEC\_out\_valid");  sc\_trace(wf, destout, "EXEC\_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") ;  sc\_trace(wf, fdout, "FLOATING\_fdout");  sc\_trace(wf, fout\_valid, "FLOATING\_fout\_valid");  sc\_trace(wf, fdestout, "FLOATING\_fdestout");  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PIC \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  sc\_signal<bool> ireq0("IREQ0") ;  sc\_signal<bool> ireq1("IREQ1") ;  sc\_signal<bool> ireq2("IREQ2") ;  sc\_signal<bool> ireq3("IREQ3") ;  sc\_signal<bool> intreq("INTREQ") ;  sc\_signal<unsigned> vectno("VECTNO") ;  sc\_signal<bool> intack\_cpu("INTACK\_CPU") ;  sc\_trace(wf, ireq0, "PIC\_ireq0");  sc\_trace(wf, ireq1, "PIC\_ireq1");  sc\_trace(wf, ireq2, "PIC\_ireq2");  sc\_trace(wf, ireq3, "PIC\_ireq3");  sc\_trace(wf, intreq, "PIC\_intreq");  sc\_trace(wf, vectno, "PIC\_vectno");  sc\_trace(wf, intack\_cpu, "PIC\_intack\_cpu");  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* GPIO \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  sc\_signal<signed> gpio\_data\_in("GPIO\_DATA\_IN"); //input data from gpio  sc\_signal<signed> gpio\_dataout("GPIO\_DATA\_OUT"); //output data from gpio  sc\_trace(wf, gpio\_data\_in, "GPIO\_gpio\_data\_in");  sc\_trace(wf, gpio\_dataout, "GPIO\_gpio\_dataout");    // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* DMA \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  sc\_signal<unsigned > dma\_addr("addr"); // address  sc\_signal<signed> dma\_data\_in\_dma("data\_in\_dma"); // input data  sc\_signal<bool> dma\_i\_d\_cache("i\_d\_cache"); // icache or dcache  sc\_signal<bool> dma\_valid\_data("valid\_data"); // input data valid  sc\_signal<bool> dma\_read\_write("read\_write"); // read or write  sc\_signal<signed> dma\_data\_out("data\_out"); // data from mem  sc\_signal<bool> dma\_access("dma\_access");  sc\_trace(wf, dma\_addr, "DMA\_addr");  sc\_trace(wf, dma\_data\_in\_dma, "DMA\_data\_in\_dma");  sc\_trace(wf, dma\_i\_d\_cache, "DMA\_i\_d\_cache");  sc\_trace(wf, dma\_valid\_data, "DMA\_valid\_data");  sc\_trace(wf, dma\_read\_write, "DMA\_read\_write");  sc\_trace(wf, dma\_data\_out, "DMA\_data\_out");  sc\_trace(wf, dma\_access, "DMA\_dma\_access");  ////////////////////////////////////////////////////////////////////////////  // MAIN PROGRAM  ////////////////////////////////////////////////////////////////////////////  sc\_clock clk("Clock", 1, SC\_NS, 0.5, 0.0, SC\_NS);  sc\_trace(wf, clk, "CLK");  fetch IFU("FETCH\_BLOCK");  IFU.init\_param(delay\_cycles);  IFU << ram\_dataout << branch\_target\_address << next\_pc << branch\_valid  << stall\_fetch << intreq << vectno << icache\_valid << ram\_cs << ram\_we  << addr << instruction << instruction\_valid << intack\_cpu << busy << clk;  decode IDU("DECODE\_BLOCK");  IDU << instruction << instruction\_valid << out\_valid << destout << dout  << dram\_dataout << dram\_rd\_valid << fdout << fout\_valid << fdestout  << next\_pc << branch\_valid << branch\_target\_address << mem\_access  << mem\_address << alu\_op << mem\_write << alu\_src << src\_A << src\_B  << decode\_valid << float\_valid << gpio\_access << dramdata << dma\_access << ireq0 <<clk;  exec IEU("EXEC\_BLOCK");  IEU << decode\_valid << alu\_op << src\_A << src\_B << alu\_src << c << v << z  << dout << out\_valid << destout << clk;  floating FPU("FLOAT\_BLOCK");  FPU << float\_valid << alu\_op << src\_A << src\_B << alu\_src << fdout  << fout\_valid << fdestout << clk;  icache ICACHE("ICACHE\_BLOCK");  ICACHE.init\_param(delay\_cycles);  ICACHE << ram\_datain << ram\_cs << ram\_we << addr << ram\_dataout  << icache\_valid << clk;  dcache DCACHE("DCACHE\_BLOCK");  DCACHE.init\_param(delay\_cycles);  DCACHE << dramdata << mem\_access << mem\_write << mem\_address << dram\_dataout  << dram\_rd\_valid << clk;  gpio GPIO("GPIO\_BLOCK");  GPIO.init\_param(delay\_cycles);  GPIO << src\_A << gpio\_data\_in << gpio\_access << mem\_write << dram\_dataout << gpio\_dataout  << dram\_rd\_valid << clk;  time\_t tbuffer = time(NULL);  gpio\_data\_in = 0xff;  //6ns  sc\_start(6, SC\_NS);  src\_A=0xeeee;  gpio\_access=1;  mem\_write=1;  cout << "@" << sc\_time\_stamp() << " Write to GPIO\n" << endl;  sc\_start(6, SC\_NS);  //8ns  assert(gpio\_dataout.read() == 0xeeee);  gpio\_access=1;  mem\_write=0;  sc\_start(6, SC\_NS);  assert(dram\_dataout.read() == 0xff);  cout << "Time for simulation = " << (time(NULL) - tbuffer) << endl;  sc\_close\_vcd\_trace\_file(wf);    return 0;  } |

**Результаты выполнения теста:**

GTKWave:

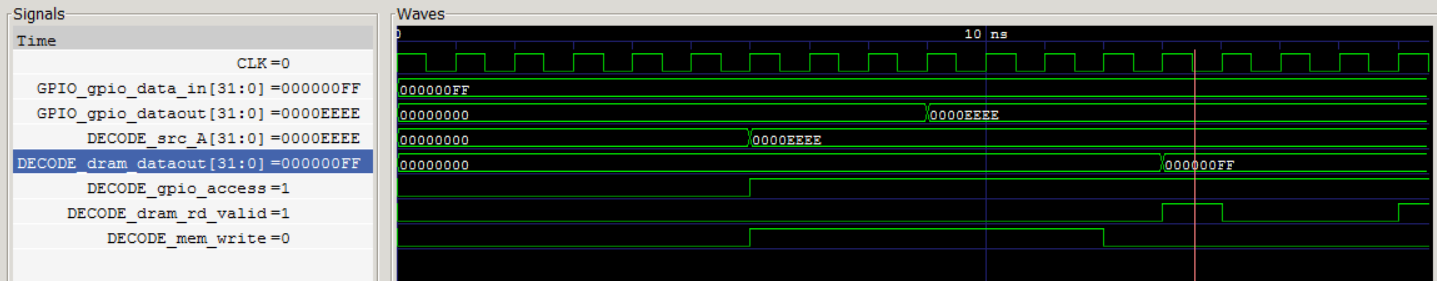


Рис 5.5. Результаты тестирования

**5.5. Модули fetch, exec, decode**

Листинг 5.5. Модульное тестирование

|  |
| --- |
|  |

**Результаты выполнения теста:**

Консоль:

|  |
| --- |
| SystemC 2.3.1-Accellera --- Feb 18 2017 01:17:28  Copyright (c) 1996-2014 by all Contributors,  ALL RIGHTS RESERVED  \*\* ALERT \*\* ID: initialize Architectural Registers  \*\* ALERT \*\* DCU: initialize Data Cache  Info: (I702) default timescale unit used for tracing: 1 ps (Processor\_waveform.vcd)  ------------------------  ICU: fetching mem[4]  ICU: (1012000) at CSIM 2 ns  ------------------------  ------------------------  IFU : mem=0x1012000  IFU : pc= 4 at CSIM 3 ns  ------------------------  -------------------------------  ID: R0= R1(=4)+R2(=5)  : at CSIM 5 ns  -------------------------------  -------------------------------  ALU : op= 3 A= 4 B= 5  ALU : R= 9-> R0 at CSIM 7 ns  -------------------------------  -------------------------------  ID: R0=0x9(9) fr ALU at CSIM 8 ns  -------------------------------  ------------------------  ICU: fetching mem[5]  ICU: (7345000) at CSIM 8 ns  ------------------------  ------------------------  IFU : mem=0x7345000  IFU : pc= 5 at CSIM 9 ns  ------------------------  -------------------------------  ID: R3=R4(=2)\*R5(=5)  : at CSIM 11 ns  -------------------------------  -------------------------------  ALU : op= 5 A= 2 B= 5  ALU : R= 10-> R3 at CSIM 13 ns  -------------------------------  -------------------------------  ID: R3=0xa(10) fr ALU at CSIM 14 ns  -------------------------------  ------------------------  ICU: fetching mem[6]  ICU: (ffffffff) at CSIM 14 ns  ------------------------  ------------------------  IFU : mem=0xffffffff  IFU : pc= 6 at CSIM 15 ns  ------------------------  -------------------------------  ID: - SHUTDOWN - at CSIM 17 ns  ID: - PLEASE WAIT ...... -  -------------------------------  ////////////////////////////////////////////////////////////////////////////////  Info: /OSCI/SystemC: Simulation stopped by user.  Time for simulation = 0 |

GTKWave:

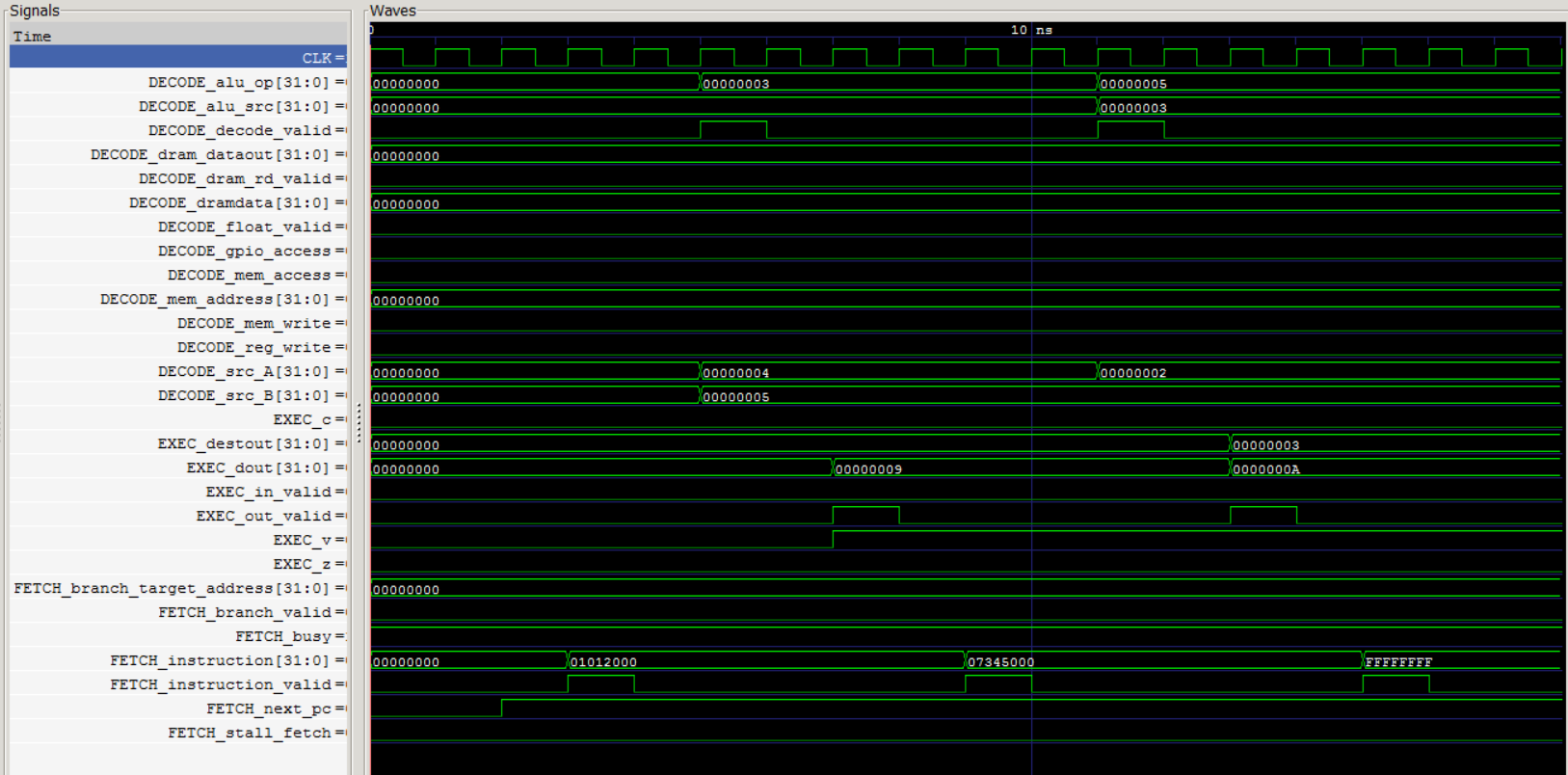


Рис 5.5. Результаты тестирования

**Раздел 6. Создание программы в машинных кодах для реализации заданных алгоритмов. Демонстрация работоспособности процессора. Оценка эффективности выполнения алгоритмов.**

1. **Сортировка** **(для массивов размером до 1024 слов)**

В качестве алгоритма был взят алгоритм сортировки методом выбора:

Листинг 6.1. Реализация алгоритма сортировки методом выбора на языке Java

|  |
| --- |
| int out, in, min;  for(out=0; out<nElems-1; out++)//Внешний цикл  {  min = out; //Минимум  for(in = out + 1; in<nElems; in++) //Внутренний цикл  if(a[in]<a[min]) //Если значение min больше  min=in; //значит, найден новый минимум  swap(out, min) // Поменять их местами  } |

Регистры:

R0 – количество элементов

R1 – начальный адрес массива в памяти (счетчик out)

R2 – конечный адрес массива-1 (NElems-1)

R3 – min=out

R4 – счетчик in

R5 – a[in]

R6 – a[min]

R7 – для swap a[out]

R8 – для swap a[min]

Программа:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Команда | Код | Описание |
| 4 | movi R0, #1000 | 0xf10003e8 | n=1000 |
| 5 | movi R1, #0 | 0xf1100000 |  |
| 6 | subi R2, R0, 1 | 0x05200001 | NElems-1 |
| 7 | mov R3, R1 | 0x0f310000 | min=out |
| 8 | addi R4, R1, 1 | 0x02410001 |  |
| 9 | ld R5, R4, 0 | 0x4d540000 | a[in] |
| a | ld R6, R3, 0 | 0x4d630000 | a[min] |
| b | bgt R5, R6 | 0x1256000d | если a[in]>a[min] пропуск присваивания(переход на d) |
| c | mov R3, R4 | 0x0f340000 | min=in |
| d | addi R4, R4, 1 | 0x02440001 | in++ |
| e | blt R4, R0 | 0x14400009 |  |
| f | ld R7, R1, 0000 | 0x4d710000 | R7=a[out] |
| 10 | ld R8, R3, 0000 | 0x4d830000 | R8=a[in] |
| 11 | wr R7, R3, 0000 | 0x4e730000 | swap a[out] a[min] |
| 12 | wr R8, R1, 0000 | 0x4e810000 |
| 13 | addi R1, R1, 0000 | 0x02110001 | out++ |
| 14 | blt R1, R2 | 0x14120007 |  |

Результат работы программы:

На вход процессору был предоставлен массив из 1000 элементов со случайными значениями.

Процессор закончил сортировку за 15066815 ns.

Время, затраченное на симуляцию, = 371 сек

1. **Умножение матриц (для матриц с каждой размерностью до 1024 слов)**

Алгоритм:

a[n][m]

b[m][p]

q[n][p] = 0

FOR i = 0 TO n-1 STEP 1

FOR j = 0 TO p-1 STEP 1

FOR k = 0 TO m-1 STEP 1

q[i][j] = q[i][j] + a[i][k]\*b[k][j]

Регистры:

R0 – записываем n

R1 – записываем p

R2 – записываем m

R3 – нулевой регистр

R4 – i

R5 – j

R6 – k

R7 – q[i][j]

R8 – [i][j]

R9 – [i][k]

R10 – [k][j]

R11 – a[i][k]

R12 – b[k][j]

R13 – mul a[i][k]\*b[k][j]

Программа:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Команда | Код | Описание |
| 4 | movi R3, #0000 | 0xf1300000 |  |
| 5 | movi R0, # n | 0xf100000a |  |
| 6 | movi R1, # p | 0xf1100009 |  |
| 7 | movi R2, # m | 0xf120000f |  |
| 8 | movi R4, 0 | 0xf1400000 |  |
| 9 | movi R5, 0 | 0xf1500000 |  |
| a | movi R7, #0 | 0xf1700000 |  |
| b | mul R8, R4, R1 | 0x07841000 |  |
| c | add R8, R8, R5 | 0x01885000 |  |
| d | movi R6, 0 | 0xf1600000 |  |
| e | mul R9, R4, R2 | 0x07942000 | Расчет [i][k] – R9 |
| f | add R9, R9, R6 | 0x01996000 |
| 10 | mul R10, R6, R1 | 0x07a61000 | Расчет [k][j] – R10 |
| 11 | add R10, R6, R1 | 0x01aa5000 |
| 12 | ld R11, R9, 0000 | 0x4db90000 | Загрузка a[i][k] |
| 13 | ld R12, R10, 00C8 | 0x4dca00c8 | Загрузка b[k][j] смещение 200 |
| 14 | mul R13, R11, R12 | 0x07dbc000 | a[i][k]\* b[k][j] |
| 15 | add R7, R7, R13 | 0x0177d000 | q[i][j]=q[i][j]+…. |
| 16 | addi R6, R6, 1 | 0x02660001 | (R6) k++ |
| 17 | blt R6, R2 | 0x1462000e | если R6<R2 прыжок |
| 18 | wr R7, R8, 400 | 0x4e780190 | сохранить q[i][j] |
| 19 | addi R5, R5, 1 | 0x02550001 | (R5) j++ |
| 1a | blt R5, R1 | 0x1451000a | если R5<R1 прыжок |
| 1b | addi R4, R4, 1 | 0x02440001 | (R4) i++ |
| 1c | blt R4, R0 | 0x14400009 | если R4<R0 прыжок |
| 1d |  | 0x00000000 |  |
| 1e |  | 0xffffffff |  |

Результат работы программы:

На вход процессору были предоставлены две матрицы размерами: 10x15 and 15x9. Результатом умножения матриц является матрица 10x9.

Процессор закончил умножение матриц за 85001 ns.

Время, затраченное на симуляцию, = 4 сек

1. **Медианный фильтр для двумерного массива**

Программа:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Команда | Код | Описание |
| 0 |  | 0xffffffff |  |
| 1 |  | 0xffffffff |  |
| 2 |  | 0xffffffff |  |
| 3 |  | 0xffffffff |  |
| 4 | movi R15, #1 | 0xf1f00001 | i=1 |
| 5 | movi R14, #1 | 0xf1e00001 | j=1 |
| 6 | movi R13, #m | 0xf1d0000a | m=10 |
| 7 | movi R12, #n | 0xf1c0000a | n=10 |
| 8 | mov R1, R15 | 0x0f1f0000 | R1=i |
| 9 | subi R1, R1, #1 | 0x05110001 | R1=i-1 |
| 10 | addi R6, R1, #3 | 0x02610003 | R6=i+2 |
| 11 | mov R2, R14 | 0x0f2e0000 | R2=j |
| 12 | subi R2, R2, #1 | 0x05220001 | R2=j-1 |
| 13 | addi R7, R2, #3 | 0x02720003 | R7=j+2 |
| 14 | mul R3, R1, R12 | 0x0731c000 | R3=[R1][R2] |
| 15 | add R3, R3, R2 | 0x01332000 |
| 16 | ld R4, R3, 0000 | 0x4d430000 | загрузка q[i-1][j-1] |
| 17 | wr R4, R5, 0000 | 0x4e450BB8 | R5=R5+1 |
| 18 | addi R5, R5, #1 | 0x02550001 | запись q[i-1][j-1] |
| 19 | addi R2, R2, #1 | 0x02220001 | [j-1]+1 |
| 20 | blt R2, R7 | 0x1427000e | переход |
| 21 | addi R1, R1, 0001 | 0x02110001 | [i-1]+1 |
| 22 | blt R1, R6 | 0x1416000b | переход |
| 23 |  | 0xf1000009 | СОРТИРОВКА |
| 24 |  | 0xf1100000 |
| 25 |  | 0x05200001 |
| 26 |  | 0x0f310000 |
| 27 |  | 0x02410001 |
| 28 |  | 0x4d540BB8 |
| 29 |  | 0x4d630BB8 |
| 30 |  | 0x12560020 |
| 31 |  | 0x0f340000 |
| 32 |  | 0x02440001 |
| 33 |  | 0x1440001C |
| 34 |  | 0x4d710BB8 |
| 35 |  | 0x4d830BB8 |
| 36 |  | 0x4e730BB8 |
| 37 |  | 0x4e810BB8 |
| 38 |  | 0x02110001 |
| 39 |  | 0x1412001a |
| 40 | mul R1, R15, R12 | 0x071fc000 | [i][j] |
| 41 | add R1, R1, R14 | 0x0111e000 |
| 42 | ld R2, R9, 0bbc | 0x4d290bbc |  |
| 43 | wr R2, R1 | 0x4e210000 |  |
| 44 | mov R5, R9 | 0x0f590000 |  |
| 45 | addi R14, R14, 1 | 0x02ee0001 | j++ |
| 46 | blt R14, R13 | 0x14ed0008 | если j<n переход |
| 47 | addi R15, R15, 1 | 0x02ff0001 | i++ |
| 48 | blt R15, R12 | 0x14fc0008 | если i<m переход |
| 49 |  | 0xffffffff |  |

Результат работы программы:

На вход процессору был предоставлен двумерный массив размером: 10x10

Процессор закончил фильтрацию за 34307 ns.

Время, затраченное на симуляцию, = 6 сек

**ВЫВОДЫ**

В ходе данной курсовой работы была разработана процессорная система. Модули процессорной системы были реализованы с использованием языка Systemc. Для системы была разработана и реализована своя система команд. Используя данные команды, разработаны программы сортировки, умножения матриц и медианного фильтра. Корректность работы программ установлена и проведено полное модульное тестирование разработанной процессорной системы.