ДОДАТОК А

Графічний матеріал атестаційної роботи

Харківський університет радіоелектроники

Кафедра електронних обчислювальних машин

1

Методи функціонального моделювання електронних систем

Керівник:

проф. Горбачов В.О.

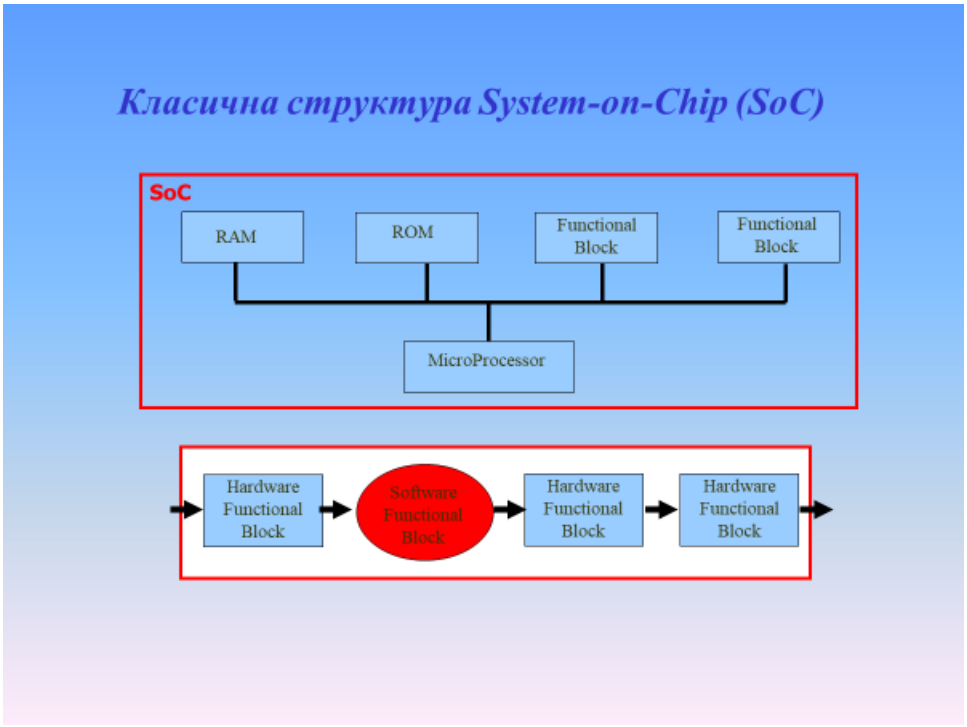
Бобровник Анатолій Анатолійович
група КСМзм 19 - 1

2

Мета дослідження: **ефективний метод верифікації складної електронної системи проектування**.

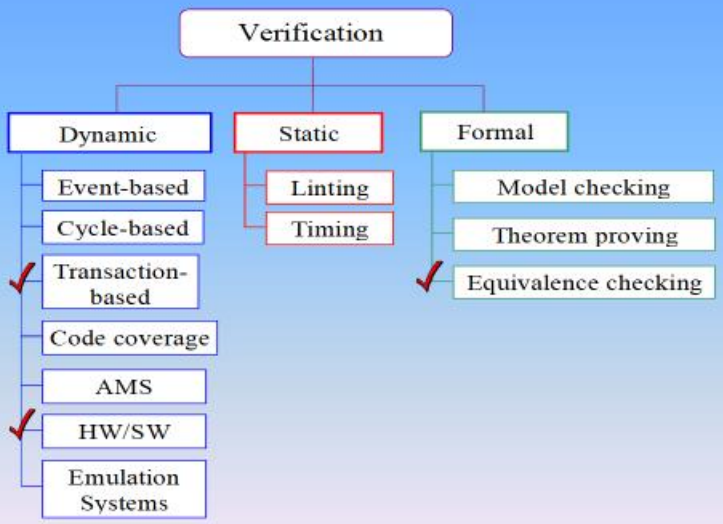
Наступні **завдання** даної дипломної роботи:

1. Огляд сучасних технологій проектування та верифікації
2. Моделювання моделі SoC на високому рівні абстракції
3. Збір даних та підготовка до подальшого аналізу
4. Створення транзакційного методу аналіза даних та оцінки його ефективності
5. Тестування та аналіз результатів

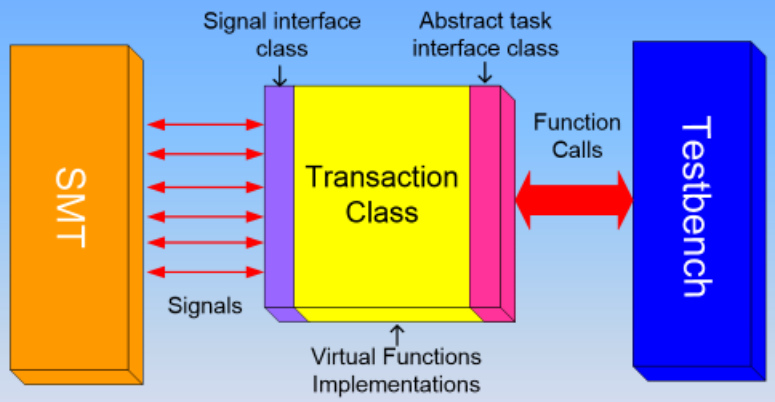

Класична структура System-on-Chip (SoC)
SoC
RAM
ROM
Functional Block
Functional Block
MicroProcessor
Hardware Functional Block
Software Functional Block
Hardware Functional Block
Hardware Functional Block


Моделювання повної системи
Створення віртуальних прототипів
Рання перевірка програмного забезпечення
Архітектурне дослідження
Аналіз ефективності
Транзакційна верифікація
Аналіз енергоспоживання
Специфікація замовника
Віртуальне прототипування
HW/SW Розмежування
ESL
Розробка обладнання
Розробка програмного забезпечення
Тест SoC
Інтеграція системи та перевірка
Виготовлення чіпів

Таксономія верифікаційних технологій



Моделювання рівня транзакцій



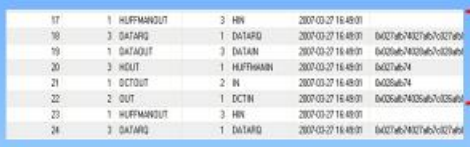
Перевірка еквівалентності

Проблема верифікації

Прискорення процесу верифікації

TLM-FP Метод

1. Початкова TDB



2. New TDMB

Transaction ID	Items
1	abcdef
2	abe
3	cef
4	acdf
5	cef

3. Items count

Items	Support
{a}	3
{b}	2
{c}	4
{d}	2
{e}	4
{f}	4

4. Упорядковані items

Transaction ID	Items
1	fecadb
2	eab
3	cef
4	fcad
5	cef

5. FP-деревое

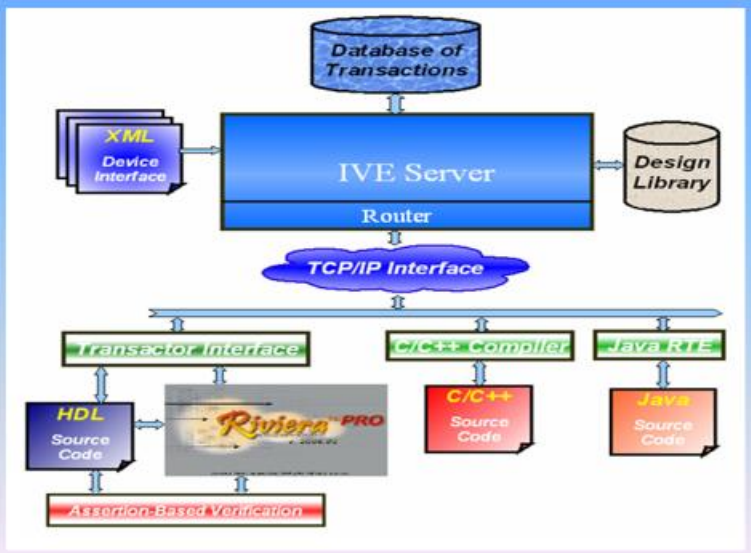
6. FP-growth

f–c-a-d : 2
c-a-d : 2
f–a-d : 2
a-d : 2
f–c-d : 2
c-d : 2
f-d : 2
d : 2

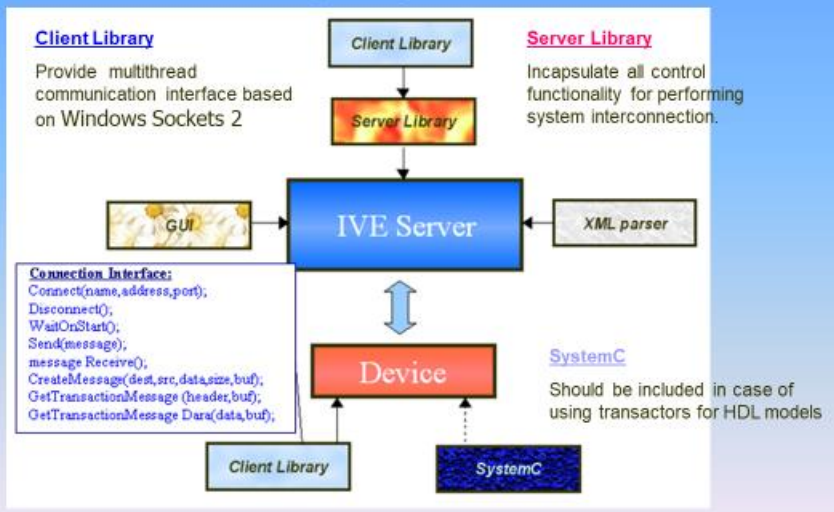
c-a : 2
e-a : 2
f-a : 2
b : 2
a-b :2
f-c-a :2
a : 3
e-c : 3

c : 4
f -e : 3
e : 4
f : 4
f -c : 4
f -e-c :4

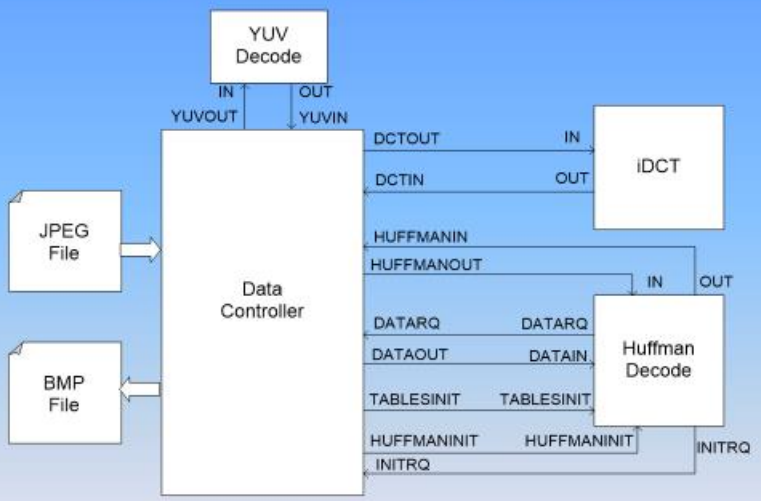
Архітектура середовища моделювання



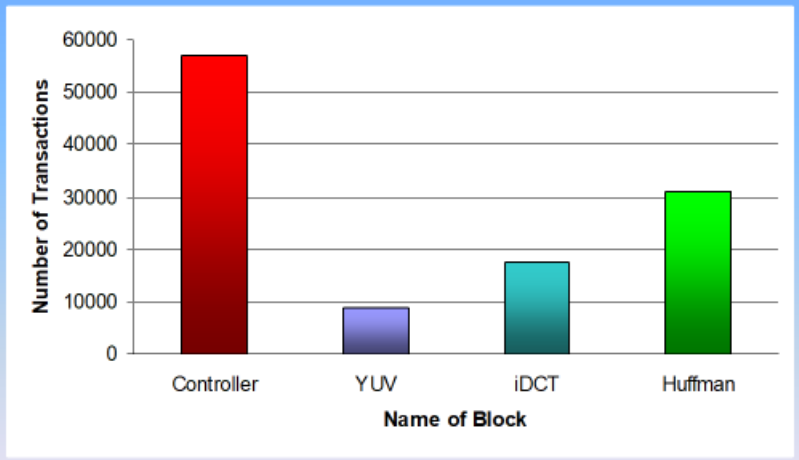
Експериментальне програмне забезпечення: базова структура проекту IVE



Приклад JPEG - схема прототипу



Приклад JPEG - Статистика транзакцій



ВИСНОВКИ

До використаного середовища моделювання на системному рівні запропоновані ключові особливості:

1. **Метод TLM-FP був використаний для аналізу транзакційних даних.**

2. **Загальна кількість транзакцій для сканування була визначена:**

ДОДАТОК Б

Зразок моделі транзактора

/tlm/HWComponent/my_component.sv:

```
/** Memory module */
`timescale 1 ns/1 ps
module m (clk,
        data_in,
        addr,
        read_en,
        write_en,
        data_out);

    parameter DATA_WIDTH = 4;
parameter ADDR_WIDTH = 4;

    output clk;
input[DATA_WIDTH - 1 : 0] data_in;
input[ADDR_WIDTH - 1 : 0] addr;
input read_en;
input write_en;
output[DATA_WIDTH - 1 : 0] data_out;
reg[DATA_WIDTH - 1 : 0] data_out;
reg [ADDR_WIDTH - 1 : 0] data [DATA_WIDTH - 1 : 0];
reg clk1 = 0;

//////////////ASSERTIONS///////////////
property read_check;
      @(posedge clk) read_en and !write_en;
endproperty

property write_check;
      @(posedge clk) !read_en and write_en;
endproperty
      always @(posedge clk )
      if (read_en && !write_en)
      begin//READ
            read_asrt: assert property (read_check);
            data_out = data[addr];
            $display("Time=%0d  ns  =>  Read:%b"  ,$time,  data[addr]);

      end
      else if (write_en && !read_en)
      begin//WRITE
            write_asrt:assert property(write_check);
            data[addr] = data_in;
            $display("Time=%0d ns => Write:%b" ,$time, data_in);
      end
      else
            $display("WAIT!!!");

//clock generator
alwaysclk1 = #100 ~clk1;
assign clk = clk1;

//TRANSACTOR INSTANCE
tester Test (clk,
             data_in,
             addr,
```

```
              read_en,
              write_en,
              data_out);


endmodule
###############TRANSACTOR MODEL################

/tlm/Tester/tester.h:
#ifndef __TESTER_H__
#define __TESTER_H__

#include <systemc.h>
#include "test.h"
#include "transactor.h"

SC_MODULE(tester)
{
  public :
            sc_in<sc_logic > clk;
      sc_out<sc_lv<4> > data_in;
      sc_out<sc_lv<4> > addr;
      sc_out<sc_logic > read_en;
      sc_out<sc_logic > write_en;
      sc_in<sc_lv<4> > data_out;

      test *ptst;
      transactor *ptrans;

SC_CTOR(tester) :
            clk("clk"),
            data_in("data_in"),
            addr("addr"),
            read_en("read_en"),
            write_en("write_en"),
            data_out("data_out")
 {
      ptst = new test("tst");
      ptrans = new transactor("trans");

      ptrans->clk(clk);
      ptrans->data_in(data_in);
      ptrans->addr(addr);
      ptrans->read_en(read_en);
      ptrans->write_en(write_en);
      ptrans->data_out(data_out);
      ptst->transactor_interface.bind(*ptrans);
    }

    ~tester()
    {
      delete ptst;
      delete ptrans;
    }

};

SC_MODULE_EXPORT(tester);

#endif //__TESTER_H__

/tlm/Tester/transactor.h:
```

```
#ifndef __TRANSACTOR_H__
#define __TRANSACTOR_H__

#include <systemc.h>

class transactor_signal_interface : public sc_module
{
  public:
            sc_in<sc_logic > clk;
      sc_out<sc_lv<4> > data_in;
      sc_out<sc_lv<4> > addr;
      sc_out<sc_logic > read_en;
      sc_out<sc_logic > write_en;
      sc_in<sc_lv<4> > data_out;
};


class transactor_task_interface : virtual public sc_interface
{
  public:
      virtual int read    (int addr) = 0;
      virtual void write  (int addr, int val) = 0;
      virtual void terminate() = 0;
};


class transactor : public transactor_signal_interface,
                   public transactor_task_interface
{
  public:
    SC_CTOR(transactor)
    {
    }

      virtual int read (int addr);
      virtual void write (int addr, int val);
      virtual void terminate();
};

#endif //__TRANSACTOR_H__


/tlm/Tester/transactor.cpp:

#include "transactor.h"
#include <systemc.h>


int transactor::read (int _addr)
{
addr.write( _addr );

read_en.write( SC_LOGIC_1 );
write_en.write( SC_LOGIC_0 );

wait(clk.posedge_event());       // wait for rising edge

return data_out.read().to_int();
}


void transactor::write (int _addr, int _val)
{
addr.write( _addr );
```

```
data_in.write( _val );
read_en.write( SC_LOGIC_0 );
write_en.write( SC_LOGIC_1 );

wait(clk.posedge_event());       // wait for rising edge

}

void transactor::terminate()
{
      sc_stop();
}


/tlm/Tester/test.h:

#ifndef __TEST_H__
#define __TEST_H__

#include <systemc.h>
#include <scv.h>
#include "transactor.h"

class test : public sc_module
{
  public:
    sc_port<transactor_task_interface> transactor_interface;


    SC_CTOR(test)
    {
      SC_THREAD(main);
    }

    void main();


};

#endif //__TEST_H__

/tlm/Tester/test.cpp:

#include "test.h"
#include "ClientLib.h"

const char* WRITE = "WRITE";
const int DELAY = 10;

void test::main()
{
  int res = 0;
  char buf_out[255];
  const char* buf_in = NULL;
  char* str_ptr;
  DataStorage newData;

if(!Connect("MEMORY","127.0.0.1",12))
 cout<<"Error: Connecting failed!";
  else
  {
//waiting on sim start
while(!WaitOnStart())
            Sleep(DELAY);
```

```
for(int count=0; count<4; count++)
{
      /** receive data */
      while(1)
      {
            buf_in = (char*)Receive();
            if(!buf_in)
                  Sleep(DELAY);
            else
                  break;
      }
      if(buf_in)
      {

            memcpy(&newData, buf_in, sizeof(newData));
            if(!newData.size)
                  cout<<"Bad storage size!\n";

            newData.data = new char[newData.data_size];
memcpy(newData.data,
buf_in+sizeof(newData), newData.data_size);
str_ptr = strstr(newData.to_port, (char*)WRITE);

            if(str_ptr)
            {
                  res = *((int*)newData.data);
                  transactor_interface->write(count, res);
                  cout << "##" << res<<endl;
            }

            delete[] newData.data;
            res = transactor_interface->read(count);

/** Send data */
CreateMessage(
"CONTROLLER",
"IN",
"MEMORY",
"READ",
(void*)&res, sizeof(res),buf_out);
            Send(buf_out);
      }
      Sleep(DELAY);
}
Disconnect();
  }

  cout << "End of the test! res:" << res << endl;
  transactor_interface->terminate();          // stop simulation

}
```