**《数字逻辑电路与片上系统》**

**实验报告**

**The experimental report of**

**Digital Logic & System on Chip**

|  |  |
| --- | --- |
| 学 号 | 20049200453 |
| 姓 名 | 刘海旭 |
| 任课教师 | 康槿 |
| 实验教师 | 康槿 |
| 实验名称 | 面向通信专业的最小化片上系统设计 |
| 时 间 | 2022-2022第1学期（2021.10-2021.11） |
| 单 位 | 西安电子科技大学通信工程学院 |
| 组 号 |  |
| 组 成 员 |  |

说明

（1）本实验报告为西安电子大学通信工程学院本科卓越工程师班数字逻辑电路与片上系统教学过程使用，包括实验前预习、实验思路整理、实验记录，还包括实验报告内容。

（2）严格遵守本模板格式，不能新增排版格式。

（3）本文档针对第四次实验，将作为实验成绩依据计入期末总评。

目 录

[第1部分 学习备忘录 1](#_Toc85211595)

[2021年10月 1](#_Toc85211596)

[第7周（2021年10月11日-10月17日） 1](#_Toc85211597)

[第8周（2021年 月 日- 月 日） 1](#_Toc85211598)

[2021年11月 1](#_Toc85211599)

[第2部分 项目设计 2](#_Toc85211600)

[1 项目计划 2](#_Toc85211601)

[2 项目需求 2](#_Toc85211602)

[3 系统概述 2](#_Toc85211603)

[第3部分 项目实现 3](#_Toc85211604)

[1.系统设计 3](#_Toc85211605)

[2.模块设计 3](#_Toc85211606)

[3.电路实现及功能仿真 3](#_Toc85211607)

[第4部分 项目测试 4](#_Toc85211608)

[1.系统调试 4](#_Toc85211609)

[2.功能验证 4](#_Toc85211610)

[3.性能分析 4](#_Toc85211611)

[第5部分 优化与思考 5](#_Toc85211612)

[1.模块优化 5](#_Toc85211613)

[2.系统优化 5](#_Toc85211614)

[3.问题与思考 5](#_Toc85211615)

[4.总结心得 5](#_Toc85211616)

# 第1部分 学习备忘录

## 2021年10月

（主要记录用于本实验的学习时间，包括课上课下）

### 第7周（2021年10月11日-10月17日）

|  |  |  |
| --- | --- | --- |
| 日期（星期） | 内容 | 用时 |
| 10.16 | 步骤一、步骤二探究 | 2h |
|  |  |  |

### 第8周（2021年 月 日- 月 日）

|  |  |  |
| --- | --- | --- |
| 日期（星期） | 内容 | 用时 |
| 10．22 | 问题讨论 | 1.5h |
| 10.23 | 指令集、FFT | 5h |
| 10.24 | 指令集、ALU | 5h |
| 10.25 | Alu.PSR | 2h |
|  |  |  |

## 2021年11月

# 第2部分 项目设计

## 1 项目计划

设计FFT运算系统进行数字信号处理

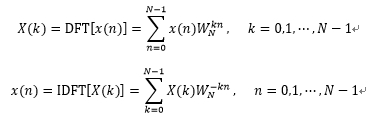
## 2 项目需求

## 3 系统概述

FFT是一种DFT的高效算法，称为快速傅立叶变换（fast Fourier transform）。傅里叶变换是时域—频域变换分析中最基本的方法之一。在数字处理领域应用的离散傅里叶变换(DFT：Discrete Fourier Transform)是许多数字信号处理方法的基础。

FFT是一种DFT的高效算法，称为快速傅立叶变换（fast Fourier transform）。FFT算法可分为按时间抽取算法和按频率抽取算法，先简要介绍FFT的基本原理。从DFT运算开始，说明FFT的基本原理。

DFT的运算为：

[](https://baike.baidu.com/pic/FFT%E5%8E%9F%E7%90%86/8966333/0/5366d0160924ab1832d4ecb237fae6cd7a890b70?fr=lemma&ct=single)

Wn=exp(-j2π/N)

C++实现代码：（来源与网络）

#include<bits/stdc++.h>

using namespace std;

//complex是stl自带的定义复数的容器

typedef complex<double> cp;

#define N 2097153

//pie表示圆周率π

const double pie=acos(-1);

int n;

cp a[N],b[N];

int rev[N],ans[N];

char s1[N],s2[N];

//读入优化

int read(){

int sum=0,f=1;

char ch=getchar();

while(ch>'9'||ch<'0'){if(ch=='-')f=-1;ch=getchar();}

while(ch>='0'&&ch<='9'){sum=(sum<<3)+(sum<<1)+ch-'0';ch=getchar();}

return sum\*f;

}

//初始化每个位置最终到达的位置

{

int len=1<<k;

for(int i=0;i<len;i++)

rev[i]=(rev[i>>1]>>1)|((i&1)<<(k-1));

}

//a表示要操作的系数，n表示序列长度

//若flag为1，则表示FFT，为-1则为IFFT(需要求倒数）

void fft(cp \*a,int n,int flag){

for(int i=0;i<n;i++)

{

//i小于rev[i]时才交换，防止同一个元素交换两次，回到它原来的位置。

if(i<rev[i])swap(a[i],a[rev[i]]);

}

for(int h=1;h<n;h\*=2)//h是准备合并序列的长度的二分之一

{

cp wn=exp(cp(0,flag\*pie/h));//求单位根w\_n^1

for(int j=0;j<n;j+=h\*2)//j表示合并到了哪一位

{

cp w(1,0);

for(int k=j;k<j+h;k++)//只扫左半部分，得到右半部分的答案

{

cp x=a[k];

cp y=w\*a[k+h];

a[k]=x+y; //这两步是蝴蝶变换

a[k+h]=x-y;

w\*=wn; //求w\_n^k

}

}

}

//判断是否是FFT还是IFFT

if(flag==-1)

for(int i=0;i<n;i++)

a[i]/=n;

}

int main(){

n=read();

scanf("%s%s",s1,s2);

//读入的数的每一位看成多项式的一项，保存在复数的实部

for(int i=0;i<n;i++)a[i]=(double)(s1[n-i-1]-'0');

for(int i=0;i<n;i++)b[i]=(double)(s2[n-i-1]-'0');

//k表示转化成二进制的位数

int k=1,s=2;

while((1<<k)<2\*n-1)k++,s<<=1;

init(k);

//FFT 把a的系数表示转化为点值表示

fft(a,s,1);

//FFT 把b的系数表示转化为点值表示

fft(b,s,1);

//FFT 两个多项式的点值表示相乘

for(int i=0;i<s;i++)

a[i]\*=b[i];

//IFFT 把这个点值表示转化为系数表示

fft(a,s,-1);

//保存答案的每一位(注意进位）

for(int i=0;i<s;i++)

{

//取实数四舍五入，此时虚数部分应当为0或由于浮点误差接近0

ans[i]+=(int)(a[i].real()+0.5);

ans[i+1]+=ans[i]/10;

ans[i]%=10;

}

while(!ans[s]&&s>-1)s--;

if(s==-1)printf("0");

else

for(int i=s;i>=0;i--)

printf("%d",ans[i]);

return 0;

}

指令集设计：



以上是参考指令集

其他指令需要进一步分析

2021.10.15添加：



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 9 | 0 | 1 | 0 | 1 | Rd |  | Rn |  |  |
| 10 | 0 | 1 | 1 | 0 | Rd |  | Rn |  | Rd Rd\*Rn 乘法 |
| 11 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 起始码/截止码 |
| 12 | 1 | 0 | 0 | 0 | Rd |  | Rn |  | (PSR,Rd) Rd-Rn 减法运算 |
| 13 | 1 | 0 | 0 | 1 | Rd |  | Rn |  | (PSR,Rd) Rd-Rn-1 借位减法运算 |
| 14 | 1 | 0 | 1 | 0 | Rd |  | Rn |  | Rd-Rn =0 判断寄存器值是否相等 |
| 15 | 1 | 0 | 1 | 1 | Rd |  | Rn |  | [Rd]-[Rn]=0 判断存储器值是否相等 |
| 16 | 1 | 1 | 0 | 1 | 1 | 0 | Rn |  | Rn【3:0】 Rn[0:3] 数据位次颠倒 |

# 第3部分 项目实现

## 1.系统设计

## 2.模块设计

ALU: `timescale 1ns / 1ps

module alu\_add\_1( //一位加法器;

reg A,//加数

reg B,//被加数

input Cin,

output reg Cout,

output reg sum

);

//get input reg value;

reg T0,T1,T2;//mark

always @(\*)

begin

sum=(A ^ B) ^ Cin;

T0=A & Cin;

T1=B & Cin;

T2=A & B;

Cout<=(T0|T1)|T2;

end

endmodule

//多位加法器;

module alu\_add\_8(

input [7:0] add1,

input [7:0] add2,

output reg [8:0] Sum

);

reg [8:0] cin;//??????;

cin<=9'b0;

alu\_add\_1 u\_alu\_add\_10{

.A(add1[0]),

.B(add2[0]),

.Cin(cin[0]),//进位标志;

.Cout(cin[1]),

.sum(Sum[0])};

genvar i=0;

generate

for(i=1; i<8; i=i+1) begin: adder\_gen

alu\_add\_1 u\_alu\_add\_1(

.A(add1[i]),

.B(add2[i]),

.Cin(cin[i]),//进位标志;

.Cout(cin[i+1]),

.sum(Sum[i]));

end

endgenerate

Sum[8]<=cin[8];

end

//赋值立即数;

module assign\_value(

input [3:0] imm,

output reg [7:0] Rd

);

begin Rd=imm[3:0];

end

endmodule

//空操作;

module nop;

#20//a clock period

endmodule

//寄存器赋值;

//寄存器转移数据;

module reg\_reg\_transfer(

input [7:0] cmd,

inout [7:0] cpu\_reg0,

inout [7:0] cpu\_reg1,

inout [7:0] cpu\_reg2,

inout [7:0] cpu\_reg3

);

begin

case(cmd[3:2])

2'b00:case(cmd[1:0])

2'b00:cpu\_reg0<=cpu\_reg0;

2'b01:cpu\_reg0<=cpu\_reg1;

2'b10:cpu\_reg0<=cpu\_reg2;

2'b11:cpu\_reg0<=cpu\_reg3;

endcase

2'b01:case(cmd[1:0])

2'b00:cpu\_reg1<=cpu\_reg0;

2'b01:cpu\_reg1<=cpu\_reg1;

2'b10:cpu\_reg1<=cpu\_reg2;

2'b11:cpu\_reg1<=cpu\_reg3;

endcase

2'b10:case(cmd[1:0])

2'b00:cpu\_reg2<=cpu\_reg0;

2'b01:cpu\_reg2<=cpu\_reg1;

2'b10:cpu\_reg2<=cpu\_reg2;

2'b11:cpu\_reg2<=cpu\_reg3;

endcase

2'b11:case(cmd[1:0])

2'b00:cpu\_reg3<=cpu\_reg0;

2'b01:cpu\_reg3<=cpu\_reg1;

2'b10:cpu\_reg3<=cpu\_reg2;

2'b11:cpu\_reg3<=cpu\_reg3;

endcase

endcase

end

endmodule

//存储器数据转移寄存器;

module mem\_reg\_transfer(

input [7:0] cmd,

inout [7:0] cpu\_reg0,

inout [7:0] cpu\_reg1,

inout [7:0] cpu\_reg2,

inout [7:0] cpu\_reg3,

inout memory

);

//根据寄存器数据取址;

//下次完成；

endmodule

//寄存器数据转移存储器;

module reg\_mem\_transfer(

input [7:0] cmd,

inout [7:0] cpu\_reg0,

inout [7:0] cpu\_reg1,

inout [7:0] cpu\_reg2,

inout [7:0] cpu\_reg3,

inout memory

);

//根据寄存器数据取址;

//下次完成；

endmodule

//带返回跳转;

module BL;

endmodule

//相等时跳转;

module BL\_CMP;

endmodule

//乘法器;

寄存器调用：

module cpu\_reg( //give reg a value and output reg value;

input rst;

input clk;

input [7:0] cmd;//命令;

input [7:0] cpu\_reg0;

input [7:0] cpu\_reg1;

input [7:0] cpu\_reg2;

input [7:0] cpu\_reg3;

output reg [7:0] PC\_reg;//程序计数器;

output reg [7:0] LR\_reg;//程序连接器;

output reg [7:0] out\_Rd;//output reg value

output reg [7:0] out\_Rn;

);

out\_Rd<=8'b0;

out\_Rn<=8'b0;

PC\_reg<=8'b0000;

LR\_reg<=8'b0000;

reg [3:0] th4\_cmd;//得到命令前四位;

always @(posedge clk or negedge rst)

begin

if(!rst) begin

out\_Rd<=8'b0;

out\_Rn<=8'b0;

PC\_reg<=8'b0000;

LR\_reg<=8'b0000;

end

th4\_cmd[3:0]=cmd[7:4];

else if(th4\_cmd[3:0]!=4'b0 && th4\_cmd[3:0]!=4'b1111) begin

if(th4\_cmd[3]&th4\_cmd[2]&(~th4\_cmd[1]|~th4\_cmd[0])) begin

case(cmd[1:0])

2'b00:out\_Rn<=cpu\_reg0;out\_Rd<=8'bz;

2'b01:out\_Rn<=cpu\_reg1;out\_Rd<=8'bz;

2'b10:out\_Rn<=cpu\_reg2;out\_Rd<=8'bz;

2'b11:out\_Rn<=cpu\_reg3;out\_Rd<=8'bz;

endcase

end

else begin

case(cmd[3:2])

2'b00:out\_Rd<=cpu\_reg0;

2'b01:out\_Rd<=cpu\_reg1;

2'b10:out\_Rd<=cpu\_reg2;

2'b11:out\_Rd<=cpu\_reg3;

endcase

case(cmd[1:0])

2'b00:out\_Rn<=cpu\_reg0;

2'b01:out\_Rn<=cpu\_reg1;

2'b10:out\_Rn<=cpu\_reg2;

2'b11:out\_Rn<=cpu\_reg3;

endcase

end

end

end

endmodule

PSR：

module cmd\_PSR(//命令控制器;

input clk,//50MHz输入;

input rst\_n,//低电平复位;

input [127:0][7:0] cmd\_code,//命令码;

input [7:0] cpu\_reg0,

input [7:0] cpu\_reg1,

input [7:0] cpu\_reg2,

input [7:0] cpu\_reg3

);

wire [7:0] cmd;

wire [7:0] Rd;

wire [7:0] Rn;

reg flag;//命令读取开始信号;

parameter s;//程序调度器;

parameter i=0;//可选参数;

integer [7:0] IA=0;//命令地址;

always @(posedge clk or negedge rst\_n) begin

if(!rst\_n)begin

cmd<=8'b0;

flag<=0;

end

else begin

cmd[7:0]=cmd\_code[LR][7:0];

IA<=IA+1;

if(cmd[7:0]==8'b01110111 && flag==0)

flag<=1;

else if(cmd[7:0]==8'b01110111 && flag==1)

flag<=0;

end

end

//程序调度器;

always @(posedge clk or negedge rst\_n) begin

if(~cmd[7]&cmd[6]&~cmd[5])begin//加法;

if(~cmd[4])

s<=4;

end

else if(~cmd[7]&~cmd[6]&~cmd[5]&~cmd[4]) //赋值立即数;

s<=0;

else if(cmd[7]&cmd[6]&cmd[5]&cmd[4]) //空操作;

s<=15;

else if(~cmd[7]&~cmd[6])begin //寄存器内存赋值;

if(~cmd[5]&cmd[4]);

s<=1;

if(cmd[5]&~cmd[4])

s<=2;

if(cmd[5]&cmd[4])

s<=3;

end

else if(cmd[7]&cmd[6]&~cmd[4]) begin //跳转;

if(~cmd[5]) //条件跳转;

s<=12;

if(cmd[5]) //无条件跳转;

s<=14;

else if(~cmd[7]&cmd[6]&cmd[5]&~cmd[4]) //乘法;

s<=6;

else if(cmd[7]&cmd[6]&~cmd[5]&cmd[4]) //????;

s<=13;

cpu\_reg u\_cpu\_reg(

.clk(clk),

.rst(rst\_n),

.cmd(cmd[7:0]),

.out\_Rd(Rd[7:0]),

.out\_Rn(Rn[7:0])

);

generate

if(s==4)begin:adder\_8

alu\_add\_8 u\_alu\_add\_8(

.add1(Rd[7:0]),

.add2(Rn[7:0]),

.sum({cpu\_CPSR,cpu\_reg0[7:0]));

end

else if(s==0)begin:assignment

assign\_value u\_assign\_value(

.imm(cmd[3:0]),

.Rd(cpu\_reg0[7:0])

);

end

else if(s==15)begin:nopnop

nop u\_nop;

end

else if(s==1)begin:transfer\_reg\_reg

reg\_reg\_transfer u\_reg\_reg\_transfer(

.cms(cmd[7:0]),

.cpu\_reg0(cpu\_reg0[7:0]),

.cpu\_reg1(cpu\_reg1[7:0]),

.cpu\_reg2(cpu\_reg2[7:0]),

.cpu\_reg3(cpu\_reg3[7:0])

);

end

else if(s==2)begin:transfer\_mem\_reg

end

else if(s==3)begin:transfer\_reg\_mem

end

endgenerate

endmodule

## 3.电路实现及功能仿真

# 第4部分 项目测试

## 1.系统调试

## 2.功能验证

## 3.性能分析

# 第5部分 优化与思考

## 1.模块优化

## 2.系统优化

## 3.问题与思考

## 4.总结心得