|  |
| --- |
|  |
| 微电网模拟系统设计报告 |
|  |
|  |
|  |
|  |

|  |
| --- |
|  |

**题目：微电网模拟系统**

**摘要**

本文针对微电网模拟系统研究背景，设计了可编程逻辑器件FPGA为控制核心的两个三相逆变器系统。本系统的硬件主要由逆变主电路系统和FPGA控制电路系统构成，包括FPGA控制电路、CC2640的AD采样电路、三相逆变驱动电路、互感器电路、辅助电源电路、调压整流电路、滤波及缓冲电路等。由FPGA控制电路输出六路PWM信号(PWM1-PWM6)来控制逆变器的MOS管通断，通过电流电压互感器对输出进行反馈，再经A/D转换器进行采样，传给FPGA控制电路来调节输出，构成闭环控制系统。本系统软件设计是利用Verilog HDL的FPGA逻辑门、IP核、时钟(DMC)等资源生成SPWM模块、并行通信模块结合TI的CC260的A/D采集和显示模块。最后，将软硬件系统联合调试，经验证，软硬件都达到预期目标，实际效果较好。

**关键字：**微电网模拟系统；FPGA可编程逻辑；三相逆变；SPWM模块

目录

[1 方案论证 1](#_Toc490331593)

[1.1 主控单元的比较与选择 1](#_Toc490331594)

[1.2 SPWM模块的比较与选择 1](#_Toc490331595)

[1.3 驱动模块的比较与选择 1](#_Toc490331596)

[1.4 方案描述 2](#_Toc490331597)

[2 理论分析与计算 2](#_Toc490331598)

[2.1 逆变器提高效率的方法 2](#_Toc490331599)

[2.2 运行模式控制策略 3](#_Toc490331600)

[3 电路与程序设计 3](#_Toc490331601)

[3.1 逆变器主电路与器件选择 3](#_Toc490331602)

[3.1.1 总体系统电路 4](#_Toc490331603)

[3.1.2 逆变电路 4](#_Toc490331604)

[3.1.3 滤波电路 4](#_Toc490331605)

[3.2 控制电路与控制程序 5](#_Toc490331606)

[3.2.1 控制电路 5](#_Toc490331607)

[3.2.2 控制程序 6](#_Toc490331608)

[4 测试方案与测试结果 6](#_Toc490331609)

[4.1 测试方案及测试条件 6](#_Toc490331610)

[4.2 测试结果 7](#_Toc490331611)

[4.3 测试结果分析 7](#_Toc490331612)

[5 总结 7](#_Toc490331613)

[参考文献 8](#_Toc490331614)

[附件 8](#_Toc490331615)

[附1：元器件明细表 8](#_Toc490331616)

[附2：仪器设备清单 8](#_Toc490331617)

[附3：电路图图纸 9](#_Toc490331618)

[附4：PCB图 10](#_Toc490331619)

[附6：程序清单 11](#_Toc490331620)

# 方案论证

## 主控单元的比较与选择

方案一：采用数字信号处理器DSP。传统基于DSP的逆变控制的设计虽然在计算的复杂度和软件的灵活性上有一定优势，但是程序为顺序运行从而导致逆变器控制算法在计算速度上受到很大的限制。

方案二：采用可编程逻辑器件FPGA。基于FPGA的逆变器的并行特点使其非常适合产生SPWM，从而在速度上占很大的优势，适合本题目要求。

综上所述，选择方案二。

## SPWM模块的比较与选择

方案一：采用比较器对正弦波和三角波进行比较得到PWM波，然后送入驱动电路放大再驱动MOSFET。但该方案受运放参数影响较大，调试困难。

方案二：运用可编程逻辑器件FPGA产生PWM通过正弦值查表法来产生SPWM。该方案实现简单，有较强的抗干扰能力。

综上所述，选择方案二。

## 驱动模块的比较与选择

方案一：采用专用驱动芯片IRS2186搭建驱动电路。驱动芯片配合外围电路完成，该方法优点是系统的集成度高，有良好的过载和短路保护功能。。

方案二：采用分立元器件搭建驱动电路。电路中选用高速开关管8050和8550，其反应速度可以达到微秒级，可以避免信号在传输过程中的累加延迟，有利于减少输出波形的失真度。但电路较复杂化，需要额外搭建保护电路。

综上所述，选择方案一。

## 方案描述

本设计的整体方案主要有FPGA控制模块，SPWM模块，驱动模块，A/D采样模块，OLED显示模块和并行通信模块组成。

FPGA

可编程逻辑主控

CC2640

低功耗MCU

并行通信

三相全桥

逆变电路

MOS

驱动电路

LC滤波器

按键设置

显示模块

滤波电路

输入

输出

图1 总体系统方案框架图

# 理论分析与计算

## 逆变器提高效率的方法

逆变器效率提升技术主要集中在两个方面：结构和器件等硬件；控制及调制策略。

结构及器件上的改进，采用软开关技术通过谐振电路，实现功率器件在零电压状态下开通或者关断，从而有效减小换流时MOS管的开关损耗，达到提升逆变器效率的目的。

控制策略的改进，采用电压空间矢量脉宽调制技术通过三相交流电压综合在一起，通过对称排列方式，可获得减小MOS管开关次数的效果，从而能够进一步减小逆变器功率器件的开关损耗。

## 运行模式控制策略

本设计三相逆变器有单独工作模式和并联工作模式。通过分析逆变电源并联基础模型，采样一种基于FPGA的无互连线复合控制方案。

双闭环反馈控制中，设，为电压环的比例系数和反馈系数，，为电流调节器的比例系数和反馈系数，为参考的指令电压。 忽略参数r的影响，经过环路分析， 设 ，可推导出系统传递函数为：

（公式1）

故空载时逆变系统的幅值静差：

（公式2）

可见， 系统的静差可随，和，的增大而不断减小。基于内模原理的重复控制技术，对于给定或具有重复性干扰的系统具有较好的控制效果，有效降低并联电流波形的 THD。结合双闭环和重复控制的并联波形控制方法，解决并联逆变电源的功率分配问题， 不用模式切换即可方便地并联使用。

# 电路与程序设计

## 逆变器主电路与器件选择

本系统器件选择FPGA主控，LC滤波电路，全桥AOTF298L芯片，电压电流互感器，OLED显示屏，以及薄膜按键。

### 总体系统电路

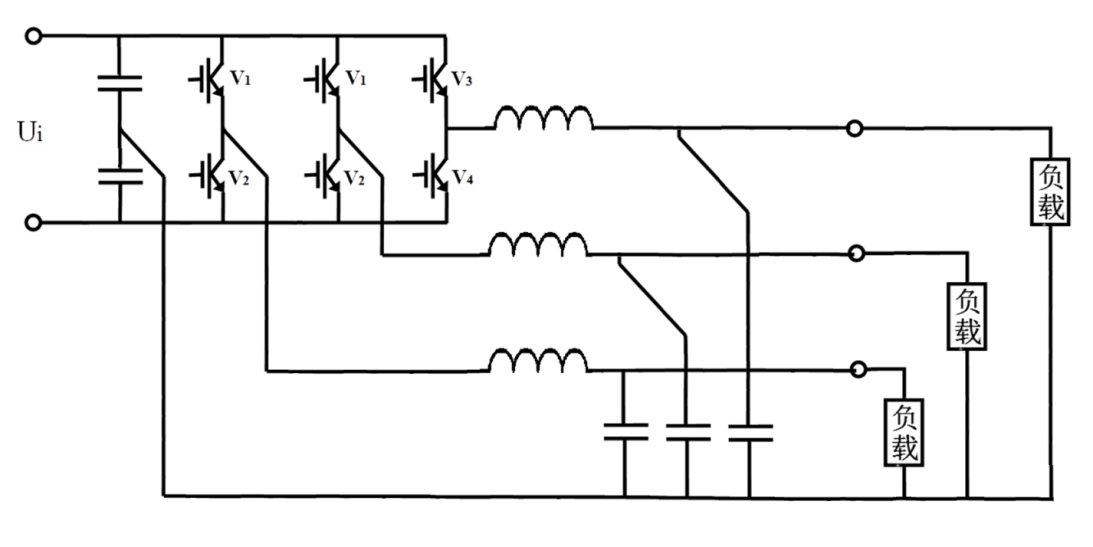


图2 总体系统电路图

### 逆变电路

逆变电路的设计采用全控型MOSFET三相桥式逆变电路。由FPGA控制器产生SPWM到IRS2186芯片搭建的驱动电路，驱动电路控制MOSFET管的通断，逆变输出经过低通滤波器将SPWM波形变换成较稳定的正弦波电压。在此电路中存在布线电感，在开关器件关断的过程中容易出现过尖峰电压，严重时会损坏开关器件，因此需要设置保护电路来抑制尖峰电压，而且设置瞬态电压抑制器和快速二极管组成的钳位电路可将MOSFET关断过程产生的尖峰电压限制在安全范围内，同时可以减少开关损耗。逆变电路图如下：

FPGA

主控制器

三相

SPWM

IRS2186

驱动电路

三相逆变

滤波输出

图3 逆变电路图

### 滤波电路

滤波电路的设计在逆变器的输出中含有逆变器开关频率和开关频率整数倍附近的谐波，如果不滤除这些高频谐波，将会给电路带来谐波污染。因此本设计选择LC滤波电路。逆变全桥输出的是200KHz的SPWM方波，基波为工频50Hz，还含有低次和高次谐波，其中幅值最大的是200KHz的谐波。在实际应用中，忽略电感对负载的分压作用及电容对负载的分流作用，并考虑变压器的电感，经计算及实验调整后，取L =100u H，C=4.7uF/100V。

## 控制电路与控制程序

本系统采用FPGA和CC2640联合控制方案，能更好的实现均流和任意比例电流输出，达到较好并联的效果。(控制电路见附录)

### 控制程序

开始

装载正弦值

按键？

SPWM输出

结束

调幅调频

图 4 主程序程序流程图

## 测试方案与测试结果

第一步：将四通道示波器的三个探针接在单片机输出PWM的引脚；

第二步：记录输出三相波形数据；

第三步：改变单片机输出SPWM的频率，返回第一步操作，直到调出50HZ的SPWM波测试完。

## 测试结果

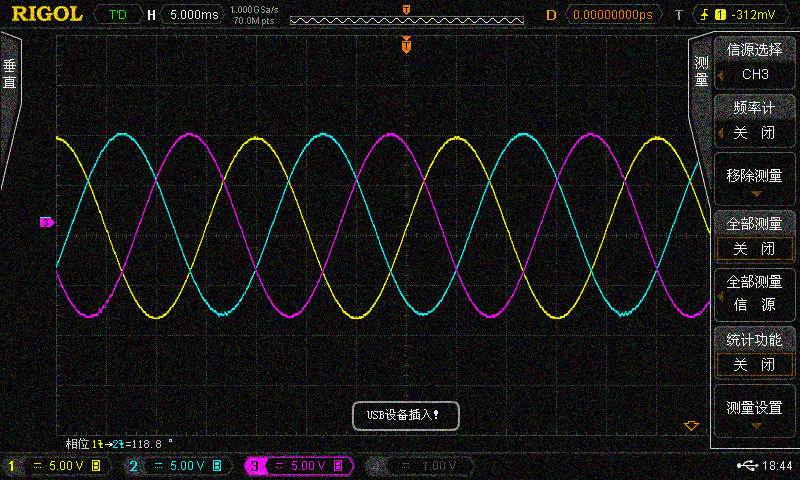


图 5 三相输出波形图

## 测试结果分析

结果分析由数据表明，三相逆变电源输出每路相位相差120度的频率可调的正弦波，电压有效值24V，电流最大输出3A。基本实现所有功能，满足题目要求。

# 总结

通过比赛，大大提高了我们的创新精神，动手能力，团队协作和竞争意识。充分发挥团队合作精神，工作进展很顺利。我们在比赛中做到精益求精，在完成基本功能之后，又向发挥部分进发，最后完成了所有的基本功能和发挥部分，较好的达到了题目要求的各项指标。

# 参考文献

[1] 邱关源.电路[M].北京：高等教育出版社，2003

[2] 华成英，童诗白.模拟电子技术基础[M].北京：高等教育出版社，2006

[3] 李练兵，光伏发电并网逆变技术：化学工业出版社，2016

[4] 路秋生，中大功率开关变换器：机械工业出版社，2017

# 附件

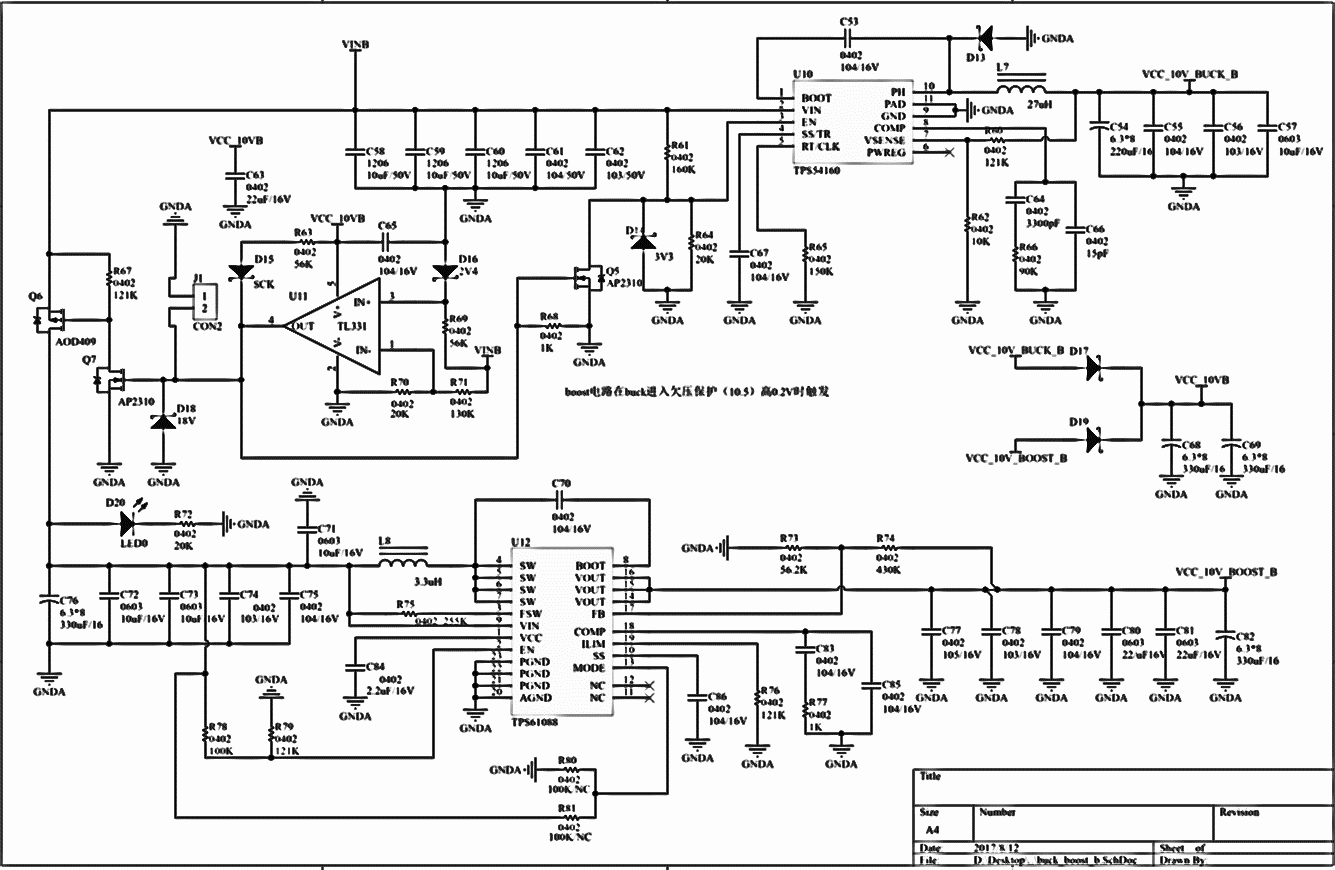
## 附1：元器件明细表

|  |  |  |
| --- | --- | --- |
| 序号 | 元器件 | 数量 |
| 1 | 可编程逻辑器件FPGA | 2 |
| 2 | TLC272 | 1 |
| 3 | MOS管 | 12 |
| 4 | 散热片 | 2 |
| 5 | 电容 | 若干 |
| 6 | INA282 | 10 |
| 7 | 电阻 | 若干 |

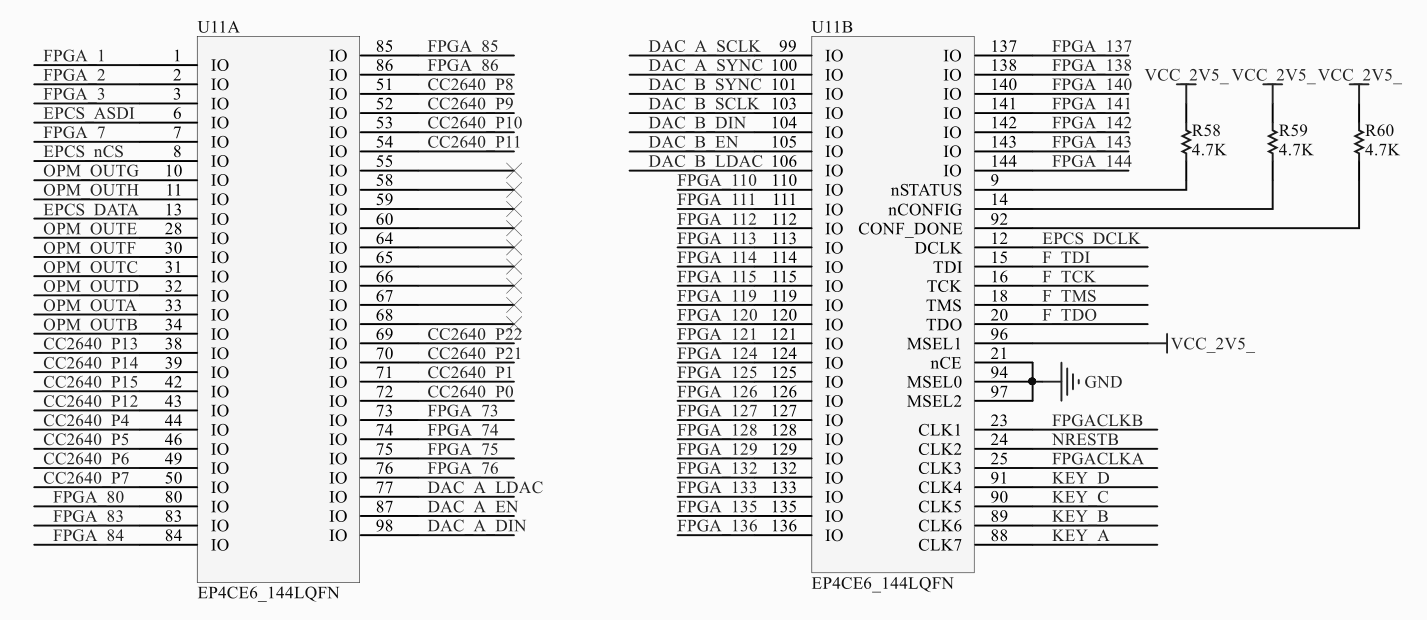
## 附2：仪器设备清单

|  |  |  |
| --- | --- | --- |
| 序号 | 仪器 | 数量 |
| 1 | 四位半数字万用表 | 4 |
| 2 | 数字电桥 | 1 |
| 3 | 直流稳压电源 | 1 |
| 4 | 函数信发生器 | 1 |
| 5 | 500MHz 4GSa/s数字示波器 | 1 |

## 附3：电路图图纸

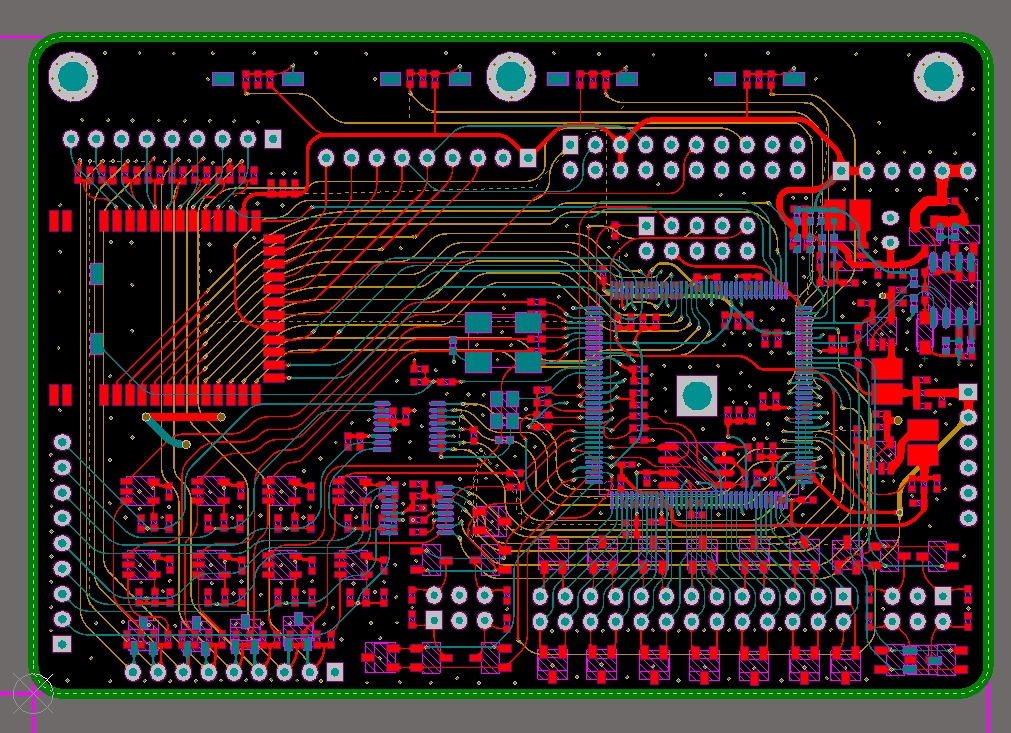


附图 1 过压保护电路图

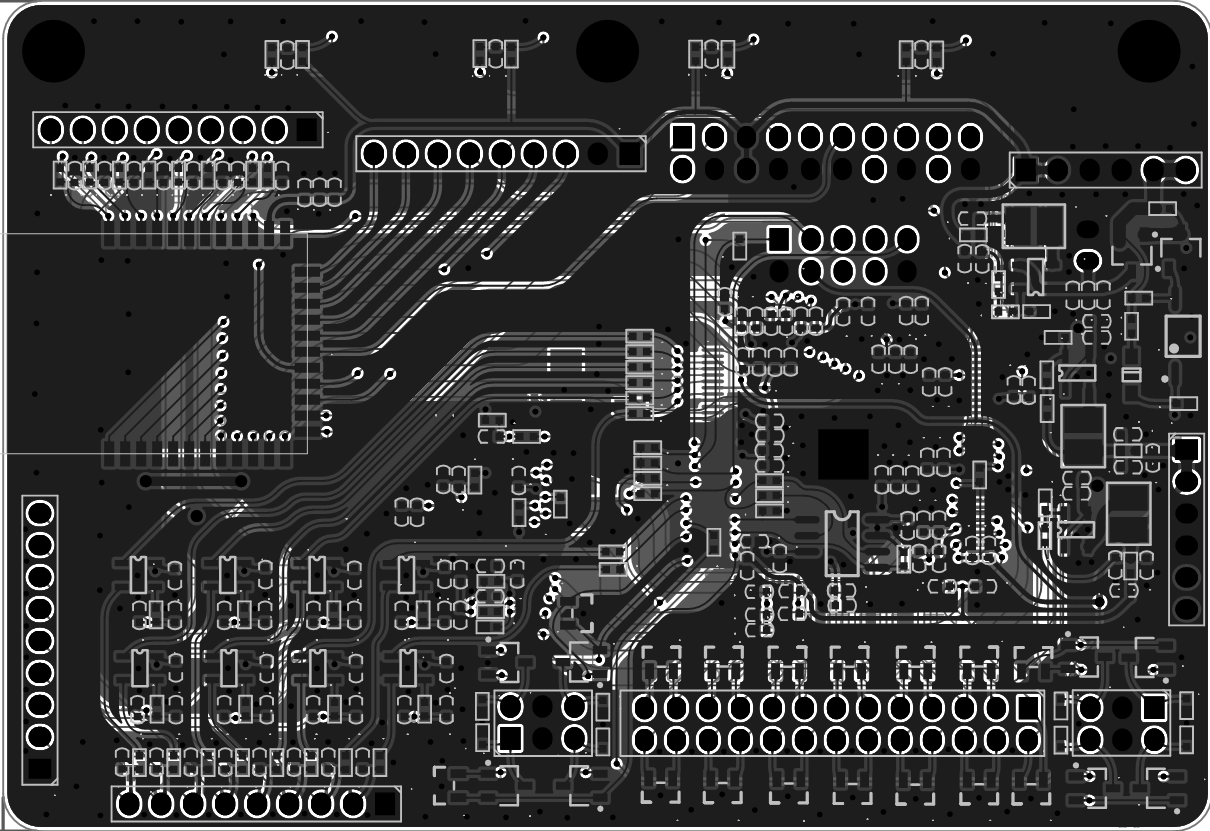


附图 2 主控制电路图

## 附4：PCB图



附图 3 主控板PCB正面图



附图 4 主控板PCB背面图

## 附6：程序清单

FPGA代码：

module sanxiangA

(clk,ah,al,bh,bl,ch,cl,hafeh,hafel,led,key\_in,key\_in1,key\_out,data\_in,

control\_jk,din,sclk,sync,enable,ldac,v\_back,flag\_out,flag\_in);

input clk;//输入时钟50M

input[1:0] data\_in;

input v\_back;//电压采样反馈

input[4:0] key\_in1;//映射按键 key[4]是并联控制信号

input[4:0] key\_in; //按键输入

output[4:0] key\_out; //按键输出

wire[4:0] key\_out;//

assign key\_out[3:0]=key\_in[3:0];

assign key\_out[4]=key\_temp;

reg key\_temp;

output control\_jk;//并联继电器控制

reg control\_jk=0;//0为继电器断开

input flag\_in;

output flag\_out;

wire flag\_out;

assign flag\_out=flag\_a;

output hafeh;//50%pwm高管

output hafel;//50%pwm低管

output[3:0] led;

parameter[25:0] pwm\_timer\_full = 26'd1000;//pwm总计数1000次 即1000档

reg[9:0] pwm\_timer\_high = 10'd450;//pwm最高档位 不应大于pwm\_timer\_full 用于调幅

reg[9:0] pwm\_hold\_a = 10'd10;//pwm高电平所用计数数量

reg[9:0] pwm\_hold\_b = 10'd10;//pwm高电平所用计数数量

reg[9:0] pwm\_hold\_c = 10'd10;//pwm高电平所用计数数量

reg[25:0] count\_pwm\_count\_a=26'd0;//a通道的pwm计数

reg[25:0] count\_pwm\_count\_b=26'd0;//b通道的pwm计数

reg[25:0] count\_pwm\_count\_c=26'd0;//c通道的pwm计数

parameter[25:0] spwm\_timer\_1hz = 26'd25\_000;//spwm计数 计数值为25’d24\_999\_999时为1HZ 实质为半周期

reg[7:0] spwm\_f = 8'd50;//spwm频率

reg[25:0] count\_spwm = 26'd49\_999;//spwm计数 即一周期pwm波的总数量 用于设置spwm波的频率 最大为25’d24\_999\_999 且count\_spwm= spwm\_timer\_1hz / spwm\_f

reg[25:0] count=26'd0;

reg[3:0] led;

reg flag=0;//为0时是单通道 为1时为并联

reg flag\_a=1;//为1时为上半波 为0时为下班波

reg flag\_b=1;

reg flag\_c=1;

reg[25:0] data\_tatol=26'd500;//总数据量

reg[15:0] datax[0:500];//偏转值数据库

always@(posedge clk)

begin

channel<=D;

count\_spwm<=spwm\_timer\_1hz/spwm\_f;

if(flag\_a==1)begin pwm\_hold\_a<=10'd500+(datax[count\_pwm\_count\_a\*data\_tatol/count\_spwm]\*pwm\_timer\_high/pwm\_timer\_full); end

if(flag\_b==1)begin pwm\_hold\_b<=10'd500+(datax[count\_pwm\_count\_b\*data\_tatol/count\_spwm]\*pwm\_timer\_high/pwm\_timer\_full); end

if(flag\_c==1)begin pwm\_hold\_c<=10'd500+(datax[count\_pwm\_count\_c\*data\_tatol/count\_spwm]\*pwm\_timer\_high/pwm\_timer\_full); end

if(flag\_a==0)begin pwm\_hold\_a<=10'd500-(datax[count\_pwm\_count\_a\*data\_tatol/count\_spwm]\*pwm\_timer\_high/pwm\_timer\_full); end

if(flag\_b==0)begin pwm\_hold\_b<=10'd500-(datax[count\_pwm\_count\_b\*data\_tatol/count\_spwm]\*pwm\_timer\_high/pwm\_timer\_full); end

if(flag\_c==0)begin pwm\_hold\_c<=10'd500-(datax[count\_pwm\_count\_c\*data\_tatol/count\_spwm]\*pwm\_timer\_high/pwm\_timer\_full); end

led<=4'b1110;

count<=count+26'd1;

if(count==26'd1) begin ah<=1;bh<=1;ch<=1;al<=0;bl<=0;cl<=0; end

if(count==pwm\_hold\_a) begin ah<=0; end

if(count==pwm\_hold\_b) begin bh<=0; end

if(count==pwm\_hold\_c) begin ch<=0; end

if((count==pwm\_hold\_a+10'd5)&(pwm\_hold\_a<pwm\_timer\_full-10'd5)) begin al<=1; end

if((count==pwm\_hold\_b+10'd5)&(pwm\_hold\_a<pwm\_timer\_full-10'd5)) begin bl<=1; end

if((count==pwm\_hold\_c+10'd5)&(pwm\_hold\_a<pwm\_timer\_full-10'd5)) begin cl<=1; end

if(count==pwm\_timer\_full-10'd5) begin al<=0;bl<=0;cl<=0; end

if(count==pwm\_timer\_full) begin count<=26'd0;key\_temp<=1;

else begin pwm\_timer\_high<=pwm\_timer\_high-10'd1; end end //反馈电压监控

count\_pwm\_count\_a<=count\_pwm\_count\_a+26'd1;

count\_pwm\_count\_b<=count\_pwm\_count\_b+26'd1;

count\_pwm\_count\_c<=count\_pwm\_count\_c+26'd1;

end

if(pwm\_hold\_a==0) begin ah<=0; end

if(pwm\_hold\_b==0) begin bh<=0; end

if(pwm\_hold\_c==0) begin ch<=0; end

if((count\_pwm\_count\_a>count\_spwm)&&(count==26'd0)) begin count\_pwm\_count\_a<=8'd0;flag\_a<=~flag\_a;key\_temp<=0; end

if((count\_pwm\_count\_a==count\_spwm\*2/3)&&(count==26'd0)) begin count\_pwm\_count\_b<=8'd0;flag\_b<=~flag\_b;end

if((count\_pwm\_count\_b==count\_spwm\*2/3)&&(count==26'd0)) begin count\_pwm\_count\_c<=8'd0;flag\_c<=~flag\_c;end

/\*以下为按键监控\*/

if ( flag\_key[0] )begin if(spwm\_f>=8'd100)begin spwm\_f<=8'd100; end else begin spwm\_f<=spwm\_f+8'd1; end end

if ( flag\_key[1] )begin if(spwm\_f<=8'd0) begin spwm\_f<=8'd0; end else begin spwm\_f<=spwm\_f-8'd1; end end

// if ( flag\_key[2] )begin if(data>=16'h3998)begin data<=16'h3998; end else begin data<=data+16'd1; end end

// if ( flag\_key[3] )begin if(data==16'h666) begin data<=16'h666; end else begin data<=data+16'd1; end end

if ( flag\_key[2] )begin if(pwm\_timer\_high>=10'd475)begin pwm\_timer\_high<=10'd475; end else begin pwm\_timer\_high<=pwm\_timer\_high+10'd20; end end

if ( flag\_key[3] )begin if(pwm\_timer\_high<=10'd25)begin pwm\_timer\_high<=10'd25; end else begin pwm\_timer\_high<=pwm\_timer\_high-10'd20; end end

if ( flag\_key[4] )begin

begin flag\_a<=flag\_in;count\_pwm\_count\_a<=8'd0;count<=26'd0; end

end

if ( flag\_key1[0] )begin if(spwm\_f>=8'd100) begin spwm\_f<=8'd100; end else begin spwm\_f<=spwm\_f+8'd1; end end

if ( flag\_key1[1] )begin if(spwm\_f<=8'd0) begin spwm\_f<=8'd0; end else begin spwm\_f<=spwm\_f-8'd1; end end

// if ( flag\_key[2] )begin if(data>=16'h3998)begin data<=16'h3998; end else begin data<=data+16'd1; end end

// if ( flag\_key[3] )begin if(data==16'h666) begin data<=16'h666; end else begin data<=data+16'd1; end end

if ( flag\_key1[2] )begin if(pwm\_timer\_high>=10'd480)begin pwm\_timer\_high<=10'd475; end else begin pwm\_timer\_high<=pwm\_timer\_high+10'd10; end end

if ( flag\_key1[3] )begin if(pwm\_timer\_high<=10'd25)begin pwm\_timer\_high<=10'd25; end else begin pwm\_timer\_high<=pwm\_timer\_high-10'd10; end end

if ( flag\_key1[4] )begin

begin flag\_a<=flag\_in;count\_pwm\_count\_a<=8'd0;count<=26'd0; end

//以下为外部来着2640的电压控制

// data<=16'h3fff;channel<=D;

// case(data\_in)

// 1: begin if(data>=16'h3998)begin data<=16'h3998; end else begin data<=data+16'd1; end end

// 2: begin if(data<=16'h666) begin data<=16'h666; end else begin data<=data+16'd1; end end

// 3: begin data<=data; end

// default: begin data<=16'd12000;channel<=C;end

// endcase

end

/\*以下为50%pwm输出\*/

reg[9:0] count\_hafe=10'd0;

always@(posedge clk)

begin

if(count\_hafe==500) begin hafeh<=0; end

if(count\_hafe==505) begin hafel<=1; end

if(count\_hafe==995) begin hafel<=0; end

if(count\_hafe==1000)begin hafeh<=1;count\_hafe<=count\_hafe+10'd1; end

count\_hafe<=count\_hafe+10'd1;

end

/\*以下为按键设置\*/

reg[19:0] counttimer=20'd0;

reg[4:0] key\_scan;

reg[4:0] key\_scan1;

always@(posedge clk)

begin

if(counttimer ==20'd999\_999) //20ms 扫描一次按键,20ms 计数(50M/50-1=999\_999)

begin

counttimer <= 20'b0; //计数器计到 20ms，计数器清零

key\_scan <= key\_in; //采样按键输入电平

key\_scan1 <= key\_in1; //映射按键输入电平

end

else counttimer <= counttimer + 20'b1; //计数器加 1

end

reg [4:0] key\_scan\_r;

reg [4:0] key\_scan\_r1;

always @(posedge clk)

begin

key\_scan\_r <= key\_scan;

key\_scan\_r1 <= key\_scan1;

end

wire [4:0] flag\_key = key\_scan\_r[4:0] & (~key\_scan[4:0]); //当检测到按键有下降沿变化时，代表该按键被按下，按键有效

wire [4:0] flag\_key1 = key\_scan\_r1[4:0] & (~key\_scan1[4:0]); //当检测到按键有下降沿变化时，代表该按键被按下，按键有效

//以下为dac参考电压控制模块设置

output din;

output sclk;

output sync;

output enable;

output ldac;

reg[1:0] channel=2'd1;

reg[15:0] data=16'd12000;

parameter A=2'd0,B=2'd1,C=2'd2,D=2'd3;

DAC8164\_spi\_control t1(.clk(clk),.channel(channel),.data(data),.din(din),.sclk(sclk),.sync(sync),

.enable(enable),.ldac(ldac));

endmodule

CC2640代码：

#include <xdc/runtime/Error.h>

#include <ti/sysbios/family/arm/cc26xx/Power.h>

#include <ti/sysbios/BIOS.h>

#include "ICall.h"

#include "bcomdef.h"

#include "peripheral.h"

#include "simpleBLEPeripheral.h"

/\* Header files required to enable instruction fetch cache \*/

#include <inc/hw\_memmap.h>

#include <driverlib/vims.h>

#ifndef USE\_DEFAULT\_USER\_CFG

#include "bleUserConfig.h"

// BLE user defined configuration

bleUserCfg\_t user0Cfg = BLE\_USER\_CFG;

#endif // USE\_DEFAULT\_USER\_CFG

/\*Exception handler\*/

void exceptionHandler()

{

volatile uint8\_t i = 1;

while(i){}

}

#ifdef FEATURE\_OAD

#if defined(\_\_IAR\_SYSTEMS\_ICC\_\_)

extern uint32\_t \_\_vector\_table;

#elif defined (\_\_TI\_COMPILER\_VERSION\_\_)

extern uint32\_t ti\_sysbios\_family\_arm\_m3\_Hwi\_resetVectors;

#endif //Compiler

#endif //FEATURE\_OAD

/\*======== main ========\*/

int main()

{

PIN\_init(BoardGpioInitTable);

#ifndef POWER\_SAVING

Power\_setConstraint(Power\_SB\_DISALLOW);

Power\_setConstraint(Power\_IDLE\_PD\_DISALLOW);

#endif // POWER\_SAVING

/\* Initialize ICall module \*/

ICall\_init();

/\* Start tasks of external images - Priority 5 \*/

ICall\_createRemoteTasks();

/\* Kick off profile - Priority 3 \*/

GAPRole\_createTask();

SimpleBLEPeripheral\_createTask();

#ifdef FEATURE\_OAD

{

uint8\_t counter;

uint32\_t \*vectorTable = (uint32\_t\*) 0x20000000;

#if defined(\_\_IAR\_SYSTEMS\_ICC\_\_)

uint32\_t \*flashVectors = &\_\_vector\_table;

#elif defined(\_\_TI\_COMPILER\_VERSION\_\_)

uint32\_t \*flashVectors = &ti\_sysbios\_family\_arm\_m3\_Hwi\_resetVectors;

#endif //Compiler.

// Write image specific interrupt vectors into RAM vector table.

for(counter = 0; counter < 15; ++counter)

{

\*vectorTable++ = \*flashVectors++;

}

}

#endif //FEATURE\_OAD

/\* enable interrupts and start SYS/BIOS \*/

BIOS\_start();

return 0;

}