

自动开关机与自动保存参数程序设计与开发文档

龙巍



2019-2-15

CSNS RCS RF

广东东莞散裂中子源

目录

[2 AutoOnOffThread版本控制 2](#_Toc1379471)

[3 AutoOnOffThread控制PV的设计 2](#_Toc1379472)

[3.1 目标1-实现自定义开关量的动作 3](#_Toc1379473)

[3.2 目标2-动作后，读取文件中的内容并保存为数值变量 4](#_Toc1379474)

[3.3 目标3-动作后，发起任务，将上述的执行操作放入任务中执行 6](#_Toc1379475)

[4 程序设计 7](#_Toc1379476)

[4.1 自动开关机过程控制PV模块的程序代码 7](#_Toc1379477)

[4.2 自动开关机过程控制PV的device support源码 9](#_Toc1379478)

[4.3 自动开机函数和自动开机任务的设计 12](#_Toc1379479)

[4.4 自动关机函数和自动关机任务的设计 24](#_Toc1379480)

[5 运行状态的PI参数的自动保存 28](#_Toc1379481)

[5.1 设计目标 28](#_Toc1379482)

[5.2 保存参数函数和保存参数任务的设计 29](#_Toc1379483)

[5.3 保存参数函数saveParms的调用 32](#_Toc1379484)

[6 自动开关机程序的总体设计和框架 34](#_Toc1379485)

[7 参数文件与其对应的环高频系统 35](#_Toc1379486)

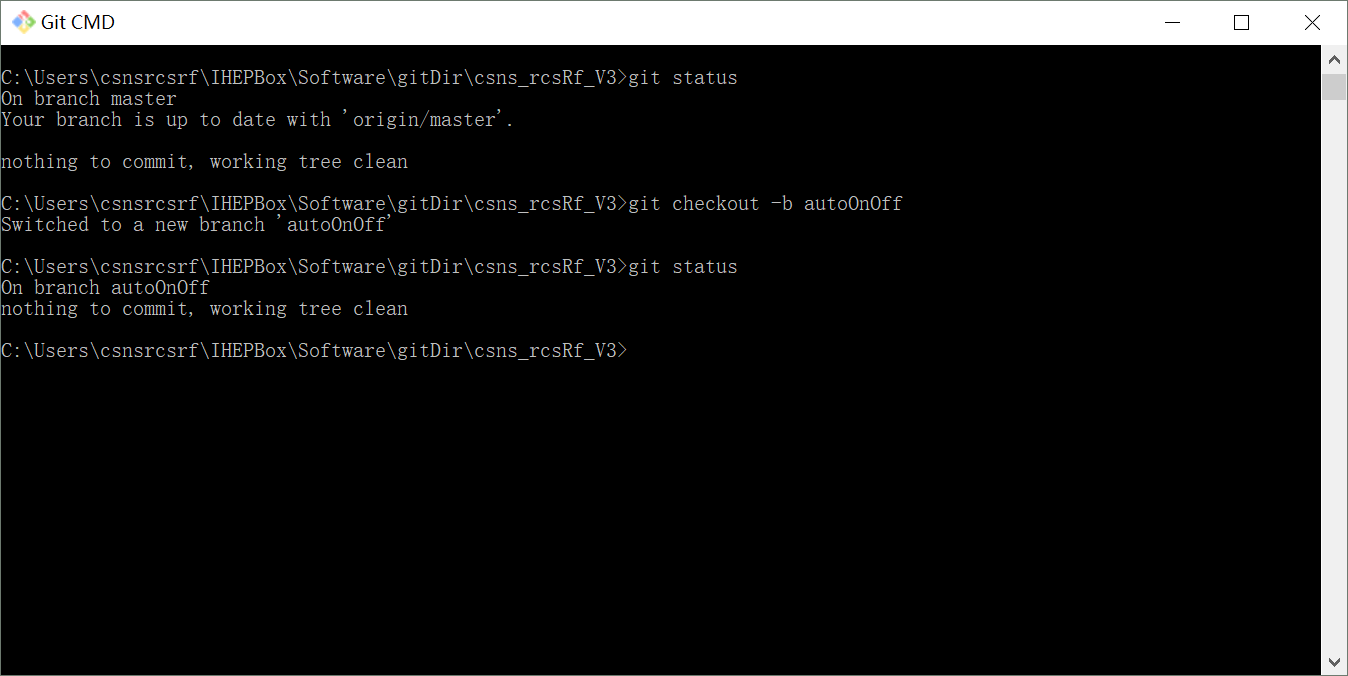
[8 发布代码 36](#_Toc1379487)

# AutoOnOffThread版本控制

CSNS RCS EPICS IOC的版本控制软件为GIT。

在之前的EPICS IOC的基础上新添加一个分支，autoOnOff。

并将当前工作切换到autoOnOff的分支上来。



# AutoOnOffThread控制PV的设计

控制PV的设计，首先通过创建一个测试用的Linux的EPICS IOC来进行调试。

首先创建一个bo类型的PV量，用来自动开关机某套高频系统。

在.db文件中添加如下代码：

record(bo, "auto")

{

field(DTYP,"AutoOnOff")

field(ZNAM,"0")

field(ONAM,"1")

}

在.dbd文件中添加如下代码：

device(bo,CONSTANT,devAutoOnOff,"AutoOnOff")

Copy Base中的软device Support程序文件devBoSoft.c文件，并重命名为devAutoOnOff.c，以此为框架实现上述的自动开关机的代码。

PV auto为一个开关量的PV，常态的值为0，对其写入1时执行自动开关机的操作。

为了方便程序的开发和调试将该目标分为如下的几个小目标。

## 目标1-实现自定义开关量的动作

开关量PV写入1时执行自定义的操作，如print一个字符串；

需要修改如下的结构体的名称为：devAutoOnOff。

struct {

long number;

DEVSUPFUN report;

DEVSUPFUN init;

DEVSUPFUN init\_record;

DEVSUPFUN get\_ioint\_info;

DEVSUPFUN write\_bo;

}devAutoOnOff={

5,

NULL,

NULL,

init\_record,

NULL,

write\_bo

};

修改epicsExportAddress的第二个参数为devAutoOnOff。

epicsExportAddress(dset,devAutoOnOff);

在write\_bo函数中注释掉不需要的功能代码，并添加如下的代码：

/\*status = dbPutLink(&prec->out,DBR\_USHORT,&prec->val,1);\*/

if (prec->val == 1){

printf("hello\n");

}

prec->val = 0;

wirte\_bo函数会在写该bo PV的时候执行，首先判断VAL的值是否为1，即是否写入了1，然后printf “hello”字符串，并重新将VAL的值置为0。

修改Makefile文件，添加device Support的源代码文件。如：

autoOnOffSupport\_SRCS += devAutoOnOff.c

如之前的.dbd文件没有被嵌套添加到Makefile文件，则也在Makefile中添加.dbd文件。如：

autoOnOff\_DBD += xxx.dbd

make，并运行IOC程序。

输入命令：

-> dbpf auto 1

观察输入是否打印了字符串”hello”。

## 目标2-动作后，读取文件中的内容并保存为数值变量

修改device support文件中的write\_bo函数如下：

static long write\_bo(boRecord \*prec)

{

long status;

int i;

FILE \* fp;

char tmp[20];

double parm;

/\*status = dbPutLink(&prec->out,DBR\_USHORT,&prec->val,1);\*/

if (prec->val == 1){

printf("hello\n");

}

prec->val = 0;

pvs[5] = 2.2;

for(i = 0; i< 10; i++)

{

printf("%f\n",pvs[i]);

}

if((fp = fopen("test.txt","r")) == NULL)

{

printf("can't open file\n");

return(status);

}

while(fscanf(fp,"%s",tmp) == 1)

{

parm = atof(tmp);

printf("in file %f\n",parm);

}

rewind(fp);

if(fclose(fp) != 0)

{

printf("Error in closing file\n");

return(status);

}

if((fp = fopen("test.txt","w")) == NULL)

{

printf("can't open file\n");

return(status);

}

fprintf(fp,"%f\n%f\n",pvs[4],pvs[5]);

if(fclose(fp) != 0)

{

printf("Error in closing file\n");

return(status);

}

return(status);

}

定义了一个10个元素的double数组pvs[10]，其中pvs[5]的值为2.2。将pvs[4]和pvs[5]写入了文件test.txt文件中。

auto PV量写入1时执行如下的操作：

*print “hello”；*

*读取test.txt文件中的内容，并将其转换为double类型的数据，并print；*

*将pvs[4]和pvs[5]写入了文件test.txt文件中。*

## 目标3-动作后，发起任务，将上述的执行操作放入任务中执行

由于Linux和VxWorks在任务的机制上有很大的区别，因此不再通过Linux EPICS IOC进行调试。

基本的思想是8套的环高频的自动开关机的操作，通过8个同样优先级别的VxWorks的任务执行。VxWorks对于同优先级别的任务有两种调度方式：独占资源和时间片轮转。独占资源的调度方式下，当前同级别的任务执行完毕后才能执行下一个同级别的任务。而时间片轮转的调度方式下，当前同级别的任务执行一段时间片的时间，然后下一个同级别的任务执行一段时间片的时间…。

由于每个任务包括一段固定时间的延时，我们希望8个任务能同时完成，为此需要开启同优先级别的时间片轮转的调度方式，

通过如下的函数开启时间片轮转调度方式。

STATUS kernelTimeSlice(int ticks);

由于任务中固定时间的延时为s级的，我们希望每个任务完成的时间误差为0.1s级别，因此将时间片设置为1/60s。

STATUS kernelTimeSlice(sysClkRateGet()/60);

VxWorks推荐的用户任务的优先级别为100~250。

由于自动开关机的任务的优先级别并不是太高，至少没有DMA等驱动任务的优先级别高，但是又高于数据采集任务（优先级别：101~104）。

因此将自动开关机任务的优先级设计为100。

# 程序设计

## 自动开关机过程控制PV模块的程序代码

在rcsRf1.db~rcsRf8.dbd中添加8套环高频系统的自动开机过程控制PV。

如在rcsRf1.db文件中添加如下代码：

record(bo,"rcsRf1:autoOn") {

field(DESC,"3")

field(DTYP,"AutoOn")

field(ZNAM,"0")

field(ONAM,"1")

}

在rcsRf1.db~rcsRf8.dbd中添加8套环高频系统的自动关机过程控制PV。

如在rcsRf1.db文件中添加如下代码：

record(bo,"rcsRf1:autoOff") {

field(DESC,"3")

field(DTYP,"AutoOff")

field(ZNAM,"0")

field(ONAM,"1")

}

其它的rcsRf*N*.db文件添加类似的代码。*N*=1-8。

DESC属性对应着FPGA的板卡号，CardNum。

在devD212.dbd中添加如下的代码：

device(bo,CONSTANT,devAutoOn,"AutoOn")

device(bo,CONSTANT,devAutoOff,"AutoOff")

添加device support的源代码文件devAutoOn.c，源文件的具体代码见3.2章节。

添加device support的源代码文件devAutoOff.c，源文件的具体代码见3.3章节。

修改Makefile文件，添加如下代码：

rcsRfIocSupport\_SRCS += devAutoOn.c

rcsRfIocSupport\_SRCS += devAutoOff.c

在st.cmd文件的开头添加如下代码：

kernelTimeSlice(sysClkRateGet()/60)

该代码用来将VxWorks的同优先级别的任务调度机制设置为时间片轮转的调度方式。

drvD212.h文件中的D212Card结构体中添加元素如下：

int processing;

用来标识该板卡的自动开机程序正在运行中，避免重复发起自动开机程序的任务。

在drvD212.c文件中的板卡初始化配置函数D212Config中对该元素进行初始化为0。

/\*将int processing元素初始化为0，标识当前自动开机程序没有运行\*/

pCard->processing = 0;

并在drvD212.h文件中添加如下的函数声明：

int autoOn(int cardNum);

该函数为自动开机函数。

在drvD212.h文件中添加如下的函数声明：

void autoOnCardNo(int cardNum);

该函数为自动开机任务的执行函数。

在drvD212.h文件中添加如下的函数声明：

int autoOff(int cardNum);

该函数为自动关机函数。

在drvD212.h文件中添加如下的函数声明：

void autoOffCardNo(int cardNum);

该函数为自动关机任务的执行函数。

## 自动开关机过程控制PV的device support源码

复制Base中的devBoSoft.c文件，重命名为devAutoOn.c，并且修改write\_bo函数的代码如下：

static long write\_bo(boRecord \*prec)

{

long status;

int cardNum;

cardNum = atoi(prec->desc); /\*将DESC属性中设置的板卡号转化为整数\*/

if ((getCardStruct(cardNum)->processing) == 0) /\*如果该板卡没有正在运行自动开机程序则调用自动开机的函数，否则不进行任何操作\*/

{

/\*注释掉如下的原代码\*/

/\*status = dbPutLink(&prec->out,DBR\_USHORT,&prec->val,1);\*/

/\*PV的值写入1时执行自动开机的动作\*/

if (prec->val == 1)

{

autoOn(cardNum); /\*调用自动开机的函数\*/

}

}

prec->val = 0;

return(status);

}

因为在write\_bo函数中调用了自动开机函数autoOn，因此需要在文件头include声明该函数的头文件。

#include <drvD212.h>

定义device support的驱动定义的结构体。

struct {

long number;

DEVSUPFUN report;

DEVSUPFUN init;

DEVSUPFUN init\_record;

DEVSUPFUN get\_ioint\_info;

DEVSUPFUN write\_bo;

}devAutoOn={

5,

NULL,

NULL,

init\_record,

NULL,

write\_bo

};

epicsExportAddress(dset,devAutoOn);

复制Base中的devBoSoft.c文件，重命名为devAutoOff.c，并且修改write\_bo函数的代码如下：

static long write\_bo(boRecord \*prec)

{

long status;

int cardNum;

cardNum = atoi(prec->desc); /\*将DESC属性中设置的板卡号转化为整数\*/

/\*注释掉如下的原代码\*/

/\*status = dbPutLink(&prec->out,DBR\_USHORT,&prec->val,1);\*/

/\*PV的值写入1时执行自动关机的动作\*/

if (prec->val == 1)

{

autoOff(cardNum); /\*调用自动关机的函数\*/

}

prec->val = 0;

return(status);

}

因为在write\_bo函数中调用了自动开机函数autoOff，因此需要在文件头include声明该函数的头文件。

#include <drvD212.h>

定义device support的驱动定义的结构体。

struct {

long number;

DEVSUPFUN report;

DEVSUPFUN init;

DEVSUPFUN init\_record;

DEVSUPFUN get\_ioint\_info;

DEVSUPFUN write\_bo;

}devAutoOff={

5,

NULL,

NULL,

init\_record,

NULL,

write\_bo

};

epicsExportAddress(dset,devAutoOff);

## 自动开机函数和自动开机任务的设计

自动开机函数autoOn函数的源代码在drvD212.c文件中。

int autoOn(int cardNum)

{

char taskName[10];

sprintf(taskName,"%s%d","autoOnCardNo",cardNum); /\*在每个自动开机任务名的后面加上板卡号，用来识别开机的系统\*/

/\* 发起自动开机的任务 \*/

if( ERROR == taskSpawn(taskName, 100, VX\_FP\_TASK, 100000, (FUNCPTR) autoOnCardNo, cardNum, 0, 0, 0, 0, 0, 0, 0, 0, 0))

{

printf("Fail to spawn autoOn task!\n");

return ERROR;

}

}

autoOn函数主要的作用是生成一个自动开机的任务。autoOn函数有一个cardNum参数，cardNum的取值范围为0-7，对应8套环高频的射频系统。最多可同时发起8个自动开机的任务。

任务的执行函数为autoOnCardNo，执行具体的开机任务，其参数cardNum用来指定运行开机任务的FPGA板卡号，对应着第几套环高频系统启动开机任务。

自动开机任务执行时会打开“paramCardNo[cardNum]文件”，并读取对应的参数，保存到数组中用来设置自动开机过程中的变量。

具体代码和设计参见源代码和注释说明。

/\*自动开机任务的执行函数\*/

void autoOnCardNo(int cardNum)

{

D212Card \*pCard;

FILE \* fp;

double parms[35]={0}; /\*自动开机所用到的参数为35个参数\*/

char fileName[20];

char charParm[20];

int i = 0;

unsigned int value; /\*value used to set the FPGA reg\*/

pCard = getCardStruct(cardNum); /\*获取对应板卡号的FPGA板卡的资源<结构体>\*/

pCard->processing = 1; /\*将板卡结构体中int processing元素赋值为1，用来标识板卡开始运行自动开机程序\*/

sprintf(fileName,"parmCardNo%d.txt",cardNum); /\*存放参数的文件名为parmCardNo[cardNum].txt\*/

/\*读取文件中的各参数并存入数组中\*/

if((fp = fopen(fileName,"r")) == NULL) /\*打开参数文件\*/

{

printf("can't open file\n");

}

while(fscanf(fp,"%s",charParm) == 1) /\*循环读取文件的每个参数并存入字符串charParm中\*/

{

parms[i] = atof(charParm); /\*将读取到的字符串参数转化为double类型的数据并存入到数组parms中\*/

}

rewind(fp); /\*将文件指针返回到文件的开头\*/

/\*关闭文件\*/

if(fclose(fp) != 0)

{

printf("Error in closing file\n");

}

/\*对FPGA寄存器进行写操作，正式进入自动开机的过程\*/

/\*设置点频频率\*/

set\_Fix\_Frequency (pCard, parms[0]);

/\*设置工作的脉冲周期\*/

set\_Work\_Period (pCard, parms[1]);

/\*设置点频下幅度\*/

set\_AMP (pCard, parms[2]);

/\*设置FF Delay的值\*/

set\_FF\_Delay (pCard, parms[3]);

/\*设置幅度曲线的系数\*/

set\_AMP\_Coefficient (pCard, parms[4]);

/\*设置幅度闭环的P参数\*/

set\_AMP\_P (pCard, parms[5]);

/\*设置幅度闭环的I参数\*/

set\_AMP\_I (pCard, parms[6]);

/\*设置调谐闭环大偏流的值\*/

set\_Bias (pCard, parms[7]);

/\*设置调谐闭环的相位角\*/

set\_Fix\_Tuning\_Angle (pCard, parms[8]);

/\*设置调谐闭环的P值\*/

set\_Tune\_P (pCard, parms[9]);

/\*设置调谐闭环的I值\*/

set\_Tune\_I (pCard, parms[10]);

/\*设置调谐闭环的I1值\*/

set\_Tune\_I\_1 (pCard, parms[11]);

/\*设置调谐闭环的I2值\*/

set\_Tune\_I\_2 (pCard, parms[12]);

/\*设置调谐闭环的I3值\*/

set\_Tune\_I\_3 (pCard, parms[13]);

/\*设置栅极调谐闭环的小偏流值\*/

set\_Front\_Bias (pCard, parms[14]);

/\*设置栅极调谐闭环的相位角\*/

set\_Front\_Fix\_Tuning\_Angle (pCard, parms[15]);

/\*设置栅极调谐闭环的P值\*/

set\_Front\_Tune\_P (pCard, parms[16]);

/\*设置栅极调谐的I值\*/

set\_Front\_Tune\_I (pCard, parms[17]);

/\*设置相位闭环的初始相位\*/

set\_Initial\_Phase (pCard, parms[18]);

/\*设置相位闭环的P值\*/

set\_phase\_p (pCard, parms[19]);

/\*设置相位闭环的I值\*/

set\_phase\_i (pCard, parms[20]);

pCard->processing = 10; /\*标识自动开机过程参数初始化完成\*/

/\*切换到扫频模式\*/

set\_point\_Sweep(pCard);

/\*切换到脉冲工作模式\*/

set\_Sweep\_Option (pCard);

/\*切换sg到脉冲模式\*/

set\_SG\_Mode (pCard);

/\*开幅度闭环\*/

clear\_AMP\_Option (pCard);

/\*关幅度闭环前馈功能\*/

clear\_AMP\_FF\_Option (pCard);

/\*关幅度闭环前馈表计算功能\*/

clear\_AMP\_Modify\_Option (pCard);

/\*开调谐闭环\*/

clear\_Tune\_Option (pCard);

/\*关调谐闭环前馈功能\*/

clear\_Tune\_FF\_Option (pCard);

/\*关调谐闭环前馈表计算功能\*/

clear\_Tune\_Modify\_Option (pCard);

/\*开栅极调谐闭环\*/

clear\_Front\_Tune\_Option (pCard);

/\*开相位闭环\*/

clear\_Phase\_Option (pCard);

/\*关相位闭环前馈功能\*/

clear\_Phase\_FF\_Option (pCard);

/\*关相位闭环前馈表计算功能\*/

clear\_Phase\_Modify\_Option (pCard);

/\*复位中断\*/

set\_RFReset\_Option (pCard);

/\*复位驱动\*/

set\_Drv\_Option (pCard);

/\*复位中断\*/

set\_RFReset\_Option (pCard);

/\*复位驱动\*/

set\_Drv\_Option (pCard);

pCard->processing = 20; /\*标识自动开机过程开关量初始化完成\*/

/\*启动幅度闭环前馈表计算功能\*/

set\_AMP\_Modify\_Option (pCard);

/\*启动调谐闭环前馈表计算功能\*/

set\_Tune\_Modify\_Option (pCard);

/\*启动相位闭环前馈表计算功能\*/

set\_Phase\_Modify\_Option (pCard);

/\*延时0.5s\*/

/\*任务主动放弃CPU资源进入延时态，此时其他同级别的任务可以获取CPU资源并运行\*/

taskDelay(sysClkRateGet()/2);

/\*复位中断\*/

set\_RFReset\_Option (pCard);

/\*复位驱动\*/

set\_Drv\_Option (pCard);

/\*复位中断\*/

set\_RFReset\_Option (pCard);

/\*复位驱动\*/

set\_Drv\_Option (pCard);

pCard->processing = 30; /\*标识自动开机过程前馈计算启动完成\*/

/\*升点频幅度值\*/

set\_AMP (pCard, parms[21]);

/\*延时0.5s\*/

taskDelay(sysClkRateGet()/2);

/\*闭调谐闭环\*/

set\_Tune\_Option (pCard);

/\*延时0.5s\*/

taskDelay(sysClkRateGet()/2);

/\*闭栅极调谐闭环\*/

set\_Front\_Tune\_Option (pCard);

pCard->processing = 40; /\*标识调谐闭环和栅极调谐闭环完成\*/

/\*延时0.5s\*/

taskDelay(sysClkRateGet()/2);

/\*加调谐闭环前馈功能\*/

set\_Tune\_FF\_Option (pCard);

/\*延时5s\*/

taskDelay(sysClkRateGet() \* 5);

/\*关调谐闭环前馈表计算功能，固定前馈表的值\*/

clear\_Tune\_Modify\_Option (pCard);

pCard->processing = 60; /\*标识调谐闭环前馈修正完成\*/

/\*调整栅极调谐闭环的I值\*/

set\_Front\_Tune\_I (pCard, parms[22]);

/\*延时0.5s\*/

taskDelay(sysClkRateGet()/2);

/\*闭幅度闭环\*/

set\_AMP\_Option (pCard);

/\*延时0.5s\*/

taskDelay(sysClkRateGet()/2);

/\*调整调谐闭环I的值\*/

set\_Tune\_I (pCard, parms[23]);

/\*调整调谐闭环I1的值\*/

set\_Tune\_I\_1 (pCard, parms[24]);

/\*调整调谐闭环I2的值\*/

set\_Tune\_I\_2 (pCard, parms[25]);

/\*调整调谐闭环I3的值\*/

set\_Tune\_I\_3 (pCard, parms[26]);

pCard->processing = 70; /\*标识幅度闭环、调谐闭环和栅极调谐闭环I值调整完成\*/

/\*延时0.5s\*/

taskDelay(sysClkRateGet()/2);

/\*加幅度闭环前馈功能\*/

set\_AMP\_FF\_Option (pCard);

/\*延时0.5s\*/

taskDelay(sysClkRateGet()/2);

/\*升扫频幅度曲线系数\*/

set\_AMP\_Coefficient (pCard, parms[27]);

/\*延时0.5s\*/

taskDelay(sysClkRateGet()/2);

/\*升扫频幅度曲线系数\*/

set\_AMP\_Coefficient (pCard, parms[28]);

/\*延时0.5s\*/

taskDelay(sysClkRateGet()/2);

/\*升扫频幅度曲线系数\*/

set\_AMP\_Coefficient (pCard, parms[29]);

pCard->processing = 90; /\*标识加幅度闭环前馈功能和升幅度系数完成\*/

/\*最后调整栅极调谐闭环I值\*/

set\_Front\_Tune\_I (pCard, parms[30]);

/\*最后调整调谐闭环I3值\*/

set\_Tune\_I\_3 (pCard, parms[31]);

/\*延时0.5s\*/

taskDelay(sysClkRateGet()/2);

/\*再次计算调谐前馈表的值\*/

set\_Tune\_Modify\_Option (pCard);

/\*延时0.5s\*/

taskDelay(sysClkRateGet()/2);

/\*闭相位闭环\*/

set\_Phase\_Option (pCard);

/\*延时0.5s\*/

taskDelay(sysClkRateGet()/2);

/\*加相位闭环前馈功能\*/

set\_Phase\_FF\_Option (pCard);

/\*设置腔间相位\*/

set\_Initial\_Ref\_Phase (pCard, parms[32]);

/\*设置FF DELAY的值\*/

set\_FF\_Delay (pCard, parms[33]);

/\*设置RBF DELAY的值\*/

set\_RBF\_Delay (pCard, parms[34]);

pCard->processing = 100; /\*最后的栅极调谐闭环I值和调谐闭环I3值调整完成、相位闭环完成和腔间相位以及电缆延时补偿设置完成\*/

/\*延时1s\*/

taskDelay(sysClkRateGet());

pCard->processing = 0; /\*标识自动开机任务的完成\*/

}

## 自动关机函数和自动关机任务的设计

自动开机函数autoOff函数的源代码在drvD212.c文件中。

int autoOff(int cardNum)

{

char taskName[10];

sprintf(taskName,"%s%d","autoOffCardNo",cardNum); /\*在每个自动关机任务名的后面加上板卡号，用来识别关机的系统\*/

/\* 发起自动关机的任务 \*/

if( ERROR == taskSpawn(taskName, 99, VX\_FP\_TASK, 100000, (FUNCPTR) autoOffCardNo, cardNum, 0, 0, 0, 0, 0, 0, 0, 0, 0))

{

printf("Fail to spawn autoOff task!\n");

return ERROR;

}

}

autoOff函数主要的作用是生成一个自动关机的任务。autoOff函数有一个cardNum参数，cardNum的取值范围为0-7，对应8套环高频的射频系统。最多可同时发起8个自动关机的任务。

我们希望自动关机的任务的优先级别高于自动开机任务的优先级别，因此将自动关机任务的优先级别设置为99。

任务的执行函数为autoOffCardNo，执行具体的关机任务，其参数cardNum用来指定运行关机任务的FPGA板卡号，对应着第几套环高频系统启动关机任务。

具体代码和设计参见源代码和注释说明。

void autoOffCardNo(int cardNum)

{

D212Card \*pCard;

char taskName[50];

int taskId;

pCard = getCardStruct(cardNum); /\*获取对应板卡号的FPGA板卡的资源<结构体>\*/

/\*如果对应板卡上的自动开机任务正在运行则删除该任务\*/

if(pCard->processing != 0)

{

sprintf(taskName, "%s%d","autoOnCardNo",cardNum);

taskId = taskNameToId(taskName);

if(taskId != taskIdSelf() && taskId != ERROR)

{

taskDelete(taskId);

}

}

/\*如果此时没有正在运行的自动开机任务，需要在自动关机任务进入延时状态时禁止运行自动开机任务\*/

if(pCard->processing == 0)

{

pCard->processing = 200;

}

/\*开相位闭环\*/

clear\_Phase\_Option (pCard);

/\*延时0.5s\*/

taskDelay(sysClkRateGet()/2);

/\*开栅极调谐闭环\*/

clear\_Front\_Tune\_Option (pCard);

/\*延时0.5s\*/

taskDelay(sysClkRateGet()/2);

/\*点频下幅度降为0\*/

set\_AMP (pCard, 0);

/\*停止计算幅度闭环前馈表的值\*/

clear\_AMP\_Modify\_Option (pCard);

/\*停止计算调谐闭环前馈表的值\*/

clear\_Tune\_Modify\_Option (pCard);

/\*停止计算相位闭环前馈表的值\*/

clear\_Phase\_Modify\_Option (pCard);

/\*关调谐闭环前馈功能\*/

clear\_Tune\_FF\_Option (pCard);

/\*关幅度闭环前馈功能\*/

clear\_AMP\_FF\_Option (pCard);

/\*关相位闭环前馈功能\*/

clear\_Phase\_FF\_Option (pCard);

pCard->processing = 0; /\*此时可以允许自动开机任务的运行了\*/

}

# 运行状态的PI参数的自动保存

## 设计目标

EPICS官方给出的AutoSave and Restore模块能够实现在IOC重启后自动恢复IOC运行时的PV的值。

但这个模块并不适用于散裂环高频低电平控制EPICS IOC中的部分变量。

在环高频低电平控制EPICS IOC启动后其幅度闭环、调谐闭环、栅极调谐闭环和相位闭环的PI参数将在用户启动开机任务后，在开机任务的过程中将发生变化，因此不能简单的恢复到原来运行状态的PI参数的值。

此外由于环高频低电平系统控制回路的复杂性，环高频系统开机时，为了运行的稳定性和处于保护高频设备的考虑，扫频幅度系数也是分阶段上升的，并不是一次加到位的。

鉴于上述的原因，开发了自己的PV值的保存和恢复的程序，作为AutoSave and Restore模块的补充。以便灵活的满足环高频低电平EPICS IOC中部分PV量，相对复杂PV值的保存和恢复的要求。

需要进行特殊的PV值保存和恢复的PV量，对应在文件中的参数和设置函数为：

parms[23]——set\_Tune\_I

parms[24]——set\_Tune\_I\_1

parms[25]——set\_Tune\_I\_2

parms[29]——set\_AMP\_Coefficient

parms[30]——set\_Front\_Tune\_I

parms[31]——set\_Tune\_I\_3

设置函数会在PV的处理过程中调用一个saveParms的函数，该函数有3个输入参数。

第一个参数为文件中参数的序号[index]。用来告诉函数保存的PV值为文件中的第几个参数。

第二个参数为cardNum，即FPGA的板卡号，用来查找每套系统对应的参数文件。

参数文件的名称为“parmCardNo[cardNum]”。

第三个参数为PV的值，用来更新文件中对应的参数的值。

saveParms函数会生成一个saveParmsCardNo[cardNum]的任务，该任务具有较低的优先级别，低于之前的自动开关机程序以及其它的数据采集任务等，这样可以不影响系统其他较为重要功能的运行。

任务具体的执行函数为saveParmsCardNo，其也有三个参数。saveParms函数在生成任务的同时将index，cardNum和val三个参数传递给saveParmsCardNo这个任务的执行函数。

添加saveParms函数和saveParmsCardNo任务执行函数在drvD212.h中的声明如下：

int saveParms(int index, int cardNum, double val); /\*声明保存参数的函数\*/

void saveParmsCardNo(int index, int cardNum, int val2int) /\*声明保存参数任务的执行函数\*/

由于每个板卡中的参数保存在同一个文件中，因此同一板卡的每个saveParmsCardNo任务将会对同一个保存参数的文件进行写操作，为了避免资源抢占引起的对文件的错误的操作，对每块板卡的saveParmsCardNo引入互斥信号量，以避免多个任务同时对同一个文件进行操作。

在drvD212.h文件中的D212Card结构体中添加元素如下：

SEM\_ID semSaveParm; /\*该互斥信号，用来多个saveParmsCardNo的任务的通讯，避免对同一文件资源的抢占，造成文件的错误操作\*/;

该信号量在板卡初始化函数D212Config中创建。

/\*创建semSaveParm互斥信号量，用来多个saveParmsCardNo的任务的通讯，避免对同一文件资源的抢占，造成文件的错误操作\*/

semSaveParm = semMCreate(SEM\_Q\_FIFO | SEM\_DELETE\_SAFE);

## 保存参数函数和保存参数任务的设计

保存参数函数saveParms函数的源代码在drvD212.c文件中。

具体代码和设计参见源代码和注释说明。

int saveParms(int index, int cardNum, double val)

{

char taskName[10];

int val2int;

sprintf(taskName,"%s%d","saveParmsCardNo",cardNum); /\*在每个保存参数任务名的后面加上板卡号，用来识别保存的是第几套高频系统的参数\*/

/\*因为传递给任务的参数只能为整数，因此将其保留小数点后3位，再乘上1000\*/

/\*在保存参数任务中需要将该值转换为double类型再除以1000\*/

if(0 < val < 100)

{

val2int = (int)(val \* 1000);

/\* 发起保存参数的任务 \*/

/\*保存参数的任务的具体的执行函数为saveParmsCardNo\*/

/\*任务的优先级别设置为较低的优先级别，低于之前的自动开关机程序以及其它的数据采集任务等\*/

/\*这样可以不影响系统其他较为重要功能的运行\*/

if( ERROR == taskSpawn(taskName, 150, VX\_FP\_TASK, 100000, (FUNCPTR) saveParmsCardNo, index, cardNum, val2int, 0, 0, 0, 0, 0, 0, 0))

{

printf("Fail to spawn saveParms task!\n");

return ERROR;

}

}

else

{

return ERROR;

}

return OK;

}

任务具体的执行函数为saveParmsCardNo，其源代码在drvD212.c中定义，其设计参见源代码和注释。

void saveParmsCardNo(int index, int cardNum, int val2int)

{

D212Card \*pCard;

double val;

FILE \* fp;

char fileName[20];

char tmp[20];

int i;

pCard = getCardStruct(cardNum); /\*获取对应板卡号的FPGA板卡的资源<结构体>\*/

semTake(pCard->semSaveParm, WAIT\_FOREVER);

val = ((double) val2int) / 1000; /\*将之前saveParms函数传递过来的PV的值恢复为double类型，并保留小数点后3位精度\*/

sprintf(fileName,"parmCardNo%d.txt",cardNum); /\*存放参数的文件名为parmCardNo[cardNum].txt\*/

/\*以可读写的方式打开参数文件\*/

if((fp = fopen(fileName,"r+")) == NULL)

{

printf("can't open file\n");

}

/\*fscanf函数读取参数文件，并且移动文件指针fp到第index个参数之前（不包括换行符"\n"）\*/

for(i=0;i<index;i++)

{

fscanf(fp,"%s",tmp);

}

/\*fprintf将fp指针处的内容覆写替换为val的值，由于fp位于"/n"换行符之前，所以替换文件内容时在前面加上换行符\*/

fprintf(fp,"\n%f",val);

/\*将fp指针返回到文件的开头\*/

rewind(fp);

/\*关闭文件\*/

if(fclose(fp) != 0)

{

printf("Error in closing file\n");

}

/\*释放互斥信号量\*/

semGive(pCard->semSaveParm);

}

## 保存参数函数saveParms的调用

需要进行特殊的PV值保存和恢复的PV量，对应在文件中的参数和设置函数为：

parms[23]——set\_Tune\_I

parms[24]——set\_Tune\_I\_1

parms[25]——set\_Tune\_I\_2

parms[29]——set\_AMP\_Coefficient

parms[30]——set\_Front\_Tune\_I

parms[31]——set\_Tune\_I\_3

在上述的设置函数中添加saveParms函数的调用代码。

由于我们不希望在开关机的过程中进行参数的保存，因此之前先查询板卡资源结构体中的int processing元素的值，如果其不为0，则表示自动开关机的任务正在运行，此时不会调用saveParms函数；如果其为0，则表示没有自动开关机的任务正在运行，此时可以调用saveParms函数。

如在设置函数set\_Tune\_I中添加如下的代码（紫色部分）。

void set\_Tune\_I (D212Card\* pCard, float tuneI)

{

unsigned int value;

value = (unsigned int)(tuneI \* CALC\_Tune\_I\_Set\_MUL + CALC\_Tune\_I\_Set\_ADD);

FPGA\_REG\_WRITE32(pCard->fpgaAddr, REG\_Tune\_I\_Set, value);

/\*调用保存参数的函数saveParms\*/

/\*使用PV值覆写序号为23的文件参数\*/

if(pCard->processing != 0)

{

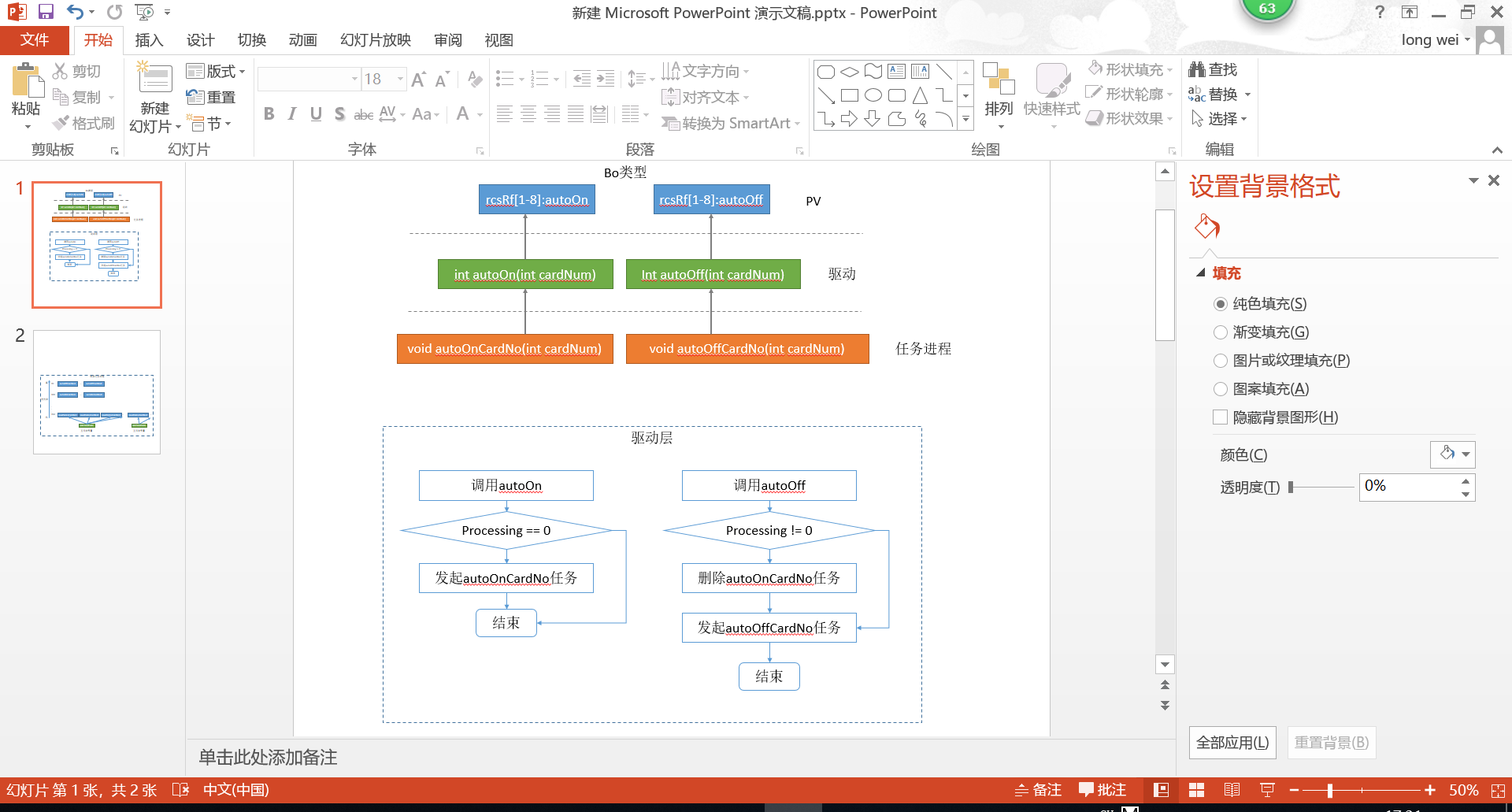
saveParms(23, pCard->cardNum, tuneI);

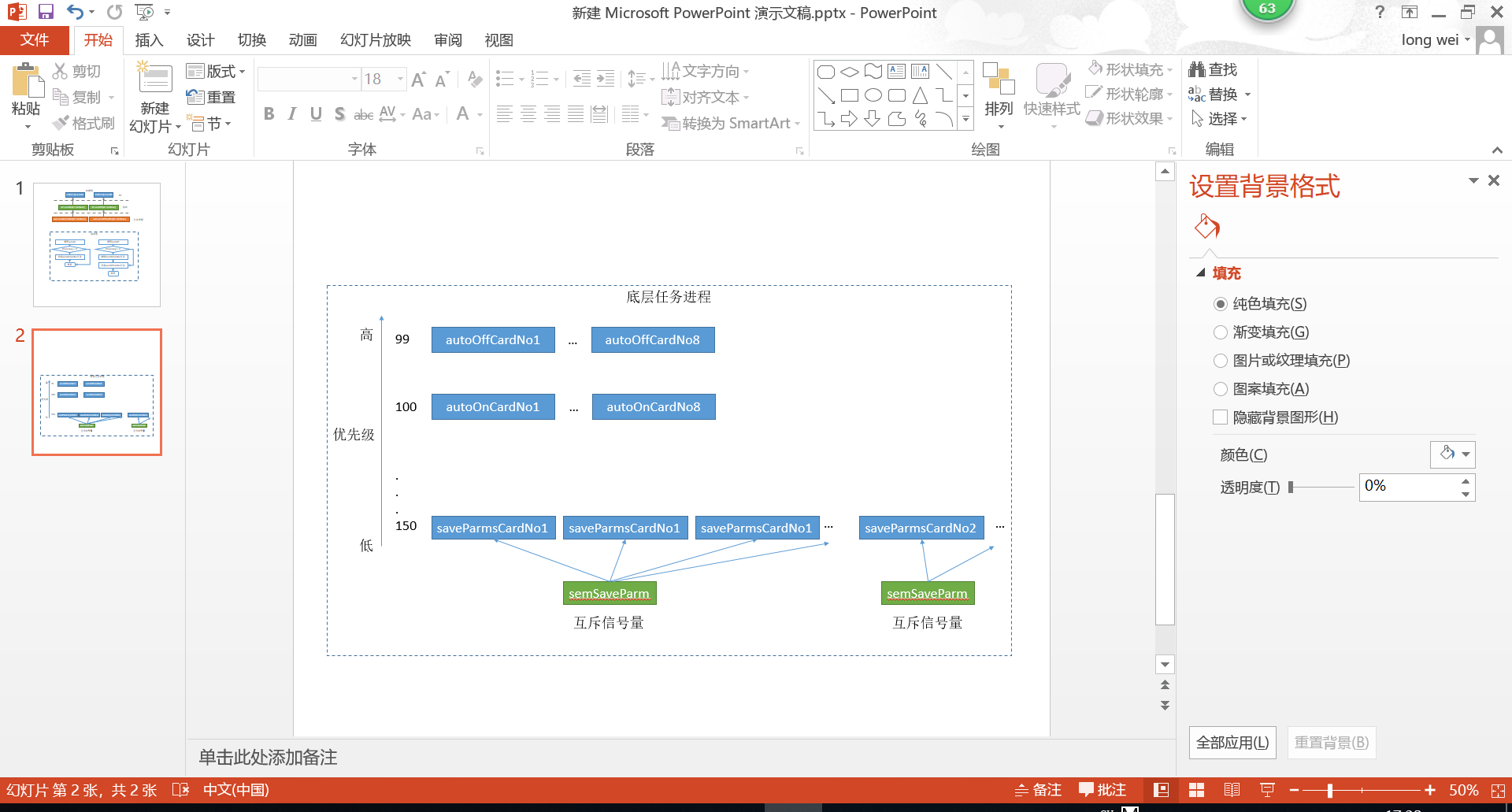
}

/\*调用保存参数的函数saveParms——结束\*/

}

# 自动开关机程序的总体设计和框架





# 参数文件与其对应的环高频系统

环高频系统共由8套独立的射频系统组成，每套系统对应的低电平控制回路由一块FPGA板卡实现，共有8块FPGA板卡。8块FPGA板卡通过CPCI总线由系统槽的CPU板卡同一管理。CPU板卡为每块FPGA板卡分配一个cardNum，用来标识和寻址FPGA板卡及其内部的寄存器。

对应的射频系统和cardNum见下表。

|  |  |
| --- | --- |
| 射频系统 | cardNum |
| 第一套（rcsRf1） | 3 |
| 第二套（rcsRf2） | 2 |
| 第三套（rcsRf3） | 1 |
| 第四套（rcsRf4） | 0 |
| 第五套（rcsRf5） | 4 |
| 第六套（rcsRf6） | 7 |
| 第七套（rcsRf7） | 6 |
| 第八套（rcsRf8） | 5 |

因此每套射频系统对应的参数文件见下表。

|  |  |
| --- | --- |
| 射频系统 | 参数文件名 |
| 第一套（rcsRf1） | parmCardNo3.txt |
| 第二套（rcsRf2） | parmCardNo2.txt |
| 第三套（rcsRf3） | parmCardNo1.txt |
| 第四套（rcsRf4） | parmCardNo0.txt |
| 第五套（rcsRf5） | parmCardNo4.txt |
| 第六套（rcsRf6） | parmCardNo7.txt |
| 第七套（rcsRf7） | parmCardNo6.txt |
| 第八套（rcsRf8） | parmCardNo5.txt |

各系统对应的参数如下：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 参数名 | rcsRf1 | rcsRf2 | rcsRf3 | rcsRf4 | rcsRf5 | rcsRf6 | rcsRf7 | rcsRf8 |
| Freq | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Period | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 |
| Amp | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| FFDelay | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| AmpCoe | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |
| AmpP | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| AmpI | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 |
| Bias | 2500 | 4600 | 2500 | 2500 | 2500 | 2500 | 3500 | 2500 |
| FixTuneA | 170 | 170 | 170 | 170 | 170 | 170 | 170 | 170 |
| TuneP | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| TuneI | 0.04 | 0.04 | 0.04 | 0.04 | 0.04 | 0.04 | 0.04 | 0.04 |
| TuneI1 | 0.04 | 0.04 | 0.04 | 0.04 | 0.04 | 0.04 | 0.04 | 0.04 |
| TuneI2 | 0.04 | 0.04 | 0.04 | 0.04 | 0.04 | 0.04 | 0.04 | 0.04 |
| TuneI3 | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 |
| FrontBias | 2000 | 2000 | 2000 | 2000 | 2000 | 2000 | 2000 | 2000 |
| FrFixTuneA | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| FrTuneP | 8 | 8 | 8 | 8 | 8 | 8 | 6 | 8 |
| FrTuneI | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 |
| InitialPhase | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| PhaseP | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| PhaseI | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 |
| Amp UP | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| 2nd FrTuneI | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 |
| 2nd TuneI | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 |
| 2nd TuneI1 | 0.1 | 0.01 | 0.16 | 0.1 | 0.1 | 0.14 | 0.1 | 0.1 |
| 2nd TuneI2 | 0.3 | 0.3 | 0.4 | 0.4 | 0.4 | 0.3 | 0.4 | 0.4 |
| 2nd TuneI3 | 0.16 | 0.16 | 0.16 | 0.16 | 0.16 | 0.16 | 0.16 | 0.16 |
| 2nd AmpCo | 140 | 140 | 140 | 140 | 140 | 140 | 140 | 140 |
| 3rd AmpCo | 200 | 200 | 200 | 200 | 200 | 200 | 200 | 200 |
| 4th AmpCo | 224 | 224 | 224 | 224 | 224 | 224 | 224 | 224 |
| 3rd FrTuneI | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 |
| 3rd TuneI3 | 0.2 | 0.16 | 0.2 | 0.2 | 0.2 | 0.16 | 0.2 | 0.2 |
| RefPhase | 0 | 109 | 107 | 131 | 140 | 150 | 178 | 290 |
| FFDelay | 26 | 26 | 26 | 26 | 26 | 26 | 26 | 26 |
| RBFDelay | 58 | 58 | 47 | 49 | 47 | 42 | 49 | 57 |

其中高亮为蓝色的为需要进行特殊的保存与恢复的系统参数。

# 发布代码

在git命令窗口运行命令

git add –all

增加所有的新添加的文件到git工程中。

git commit –a

标记此次的更新为：

2019-02-18 add parmCardNo[N].txt file for autoOn and save Restore (Customed) system.

git

运行命令

git push

发布代码到github。

代码地址为: <https://github.com/LongWei83/csns_rcsRf_V3.git>

可以通过git clone下载该代码，分支为autoOnOff分支。