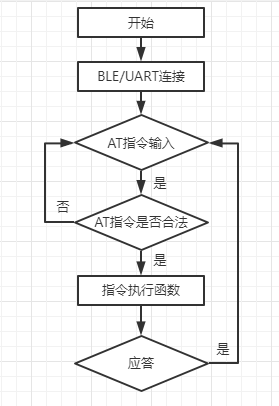
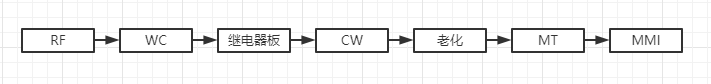
法兰西开关面板ATS设计

一、模块设计：

ATS作为单独一个固件，程序起来时默认就是厂测模式，可通过发送AT指令进入RF模式，老化模式。RF测试就是BLE测量dtm，老化测试主要是做继电器的老化。单板ATS为UART通信，总装ATS为蓝牙通信。如果AT指令被注册过的话，发送指令就会去执行对应的指令函数。例如发送AT^VERSION?就能查询软件版本号。程序流程图如下：



法兰西开关面板的工艺路线流程图如下：



该ATS测试的工位由RF测试，WC测试，继电器板测试，老化测试，CW测试，MT测试，MMI测试组成。

|  |  |
| --- | --- |
| 工位 | 测试项描述 |
| RF测试 | 主要是BLE测试dtm。 |
| WC测试 | 写读BSN，查版本号，设置获取当前模式,，LDOEFUSE电平获取等。 |
| 继电器板测试 | 获取继电器温度，ID1检测，控制继电器开关，继电器引脚控制等。 |
| 老化测试 | 设置获取老化时长, 设置获取老化标志位, 获取老化结果。 |
| CW测试 | 设置获取MAC、SN、PRODUCTID、SECRET、PRODUCT TYPE。 |
| MT测试 | 测试BLE功率。 |
| MMI测试 | 获取按键电平，获取零线检测信号，设置LED开关等。 |

部分AT指令介绍：

|  |  |
| --- | --- |
| AT指令 | 功能描述 |
| AT^BSN=<BSN> | 写入BSN，设置主板单板条码 |
| AT^RELAYON=x | 对应继电器开，x为继电器 可选参数：0 1 2 |
| AT^RELAYOFF=x | 对应继电器关，x为继电器 可选参数：0 1 2 |
| AT^AGINGTIME=x | 设置老化总时长 单位小时 |
| AT^AGING=x | 设置老化标志位 |
| AT^RELAYPIN=x,y,z | 继电器引脚控制 |
| AT^SN=<SN> | 设置SN |
| AT^PRODUCTID=xxxx | 设置PRODUCTID |
| AT^KEY? | 获取按键电平 |

二、API接口

1.1 AppMode GetAppMode(void)

|  |  |
| --- | --- |
| 函数原型 | AppMode GetAppMode(void) |
| 功能描述 | 获取当前模式 |
| 输入参数 | 无 |
| 返回值 | 应用模式，RF模式，厂测模式其中一种 |

函数实现：

/\* 当前模式 \*/

AppMode GetAppMode(void)

{

    unsigned char mode[USE\_MODE\_NVDS\_LEN] = { 0 };

    unsigned int len = USE\_MODE\_NVDS\_LEN;

    unsigned char modeInput[USE\_MODE\_NVDS\_LEN] = { USE\_MODE\_HEAD, USE\_MODE\_MAGIC\_1, USE\_MODE\_MAGIC\_2, FACTORY\_MODE};

    int ret = FactoryNvRead(ID\_USE\_MODE, mode, sizeof(mode), &len);

    /\* 出错进入工厂模式 \*/

    AT\_CHECK\_RETURN(ret == ATS\_ERR\_SUCCESS, FACTORY\_MODE);

    if (memcmp(mode, modeInput, (USE\_MODE\_NVDS\_LEN - 1)) != 0) {

        (void)FactoryNvWrite(ID\_USE\_MODE, modeInput, USE\_MODE\_NVDS\_LEN);

        return FACTORY\_MODE;

    }

    return mode[USE\_MODE\_NVDS\_LEN - 1];

}

1.2 static int EnterFactory(void)

|  |  |
| --- | --- |
| 函数原型 | static int EnterFactory(void) |
| 功能描述 | 进入厂测模式 |
| 输入参数 | 无 |
| 返回值 | 成功返回0，返回其他就是失败 |

函数实现：

static int EnterFactory(void)

{

    int ret = FactoryInit();

    AT\_CHECK\_RETURN(ret == ATS\_ERR\_SUCCESS, ATS\_ERR\_FAIL);

    ret = CHIP\_WatchDogInit();

    AT\_CHECK\_RETURN(ret == RET\_OK, ATS\_ERR\_FAIL);

    ret = AgingMain();

    if (ret != ATS\_ERR\_SUCCESS) {

        FACT\_LOG("err %d\n", ret);

    }

    xTaskCreate(AtWorkTask, "at\_task", AT\_TASK\_SIZE, NULL, 0, NULL);

    return ATS\_ERR\_SUCCESS;

}

1.3 static int FactoryInit(void)

|  |  |
| --- | --- |
| 函数原型 | static int FactoryInit(void) |
| 功能描述 | 消息队列和硬件初始化 |
| 输入参数 | 无 |
| 返回值 | 成功返回0，返回其他就是失败 |

函数实现：

/\* 初始化 \*/

static int FactoryInit(void)

{

    InterruptMsgInit();

    int ret = FactoryMsgInit();

    AT\_CHECK\_RETURN(ret == ATS\_ERR\_SUCCESS, ATS\_ERR\_FAIL);

    ret = AtsHwInit();

    AT\_CHECK\_RETURN(ret == ATS\_ERR\_SUCCESS, ATS\_ERR\_FAIL);

#if APP\_DFU\_ENABLE

    DfuInit();

#endif

#if GOODIX\_FAST\_DFU\_ENABLE

    FastDfuInit();

#endif

    return ATS\_ERR\_SUCCESS;

}

1.4 int AgingMain(void)

|  |  |
| --- | --- |
| 函数原型 | int AgingMain(void) |
| 功能描述 | 进入老化任务 |
| 输入参数 | 无 |
| 返回值 | 成功返回0，返回其他就是失败 |

函数实现：

/\* 老化任务 \*/

int AgingMain(void)

{

    NvAgingParam aging = { 0 };

    unsigned int agingLen;

    int ret = FactoryNvRead(ID\_AGING, (unsigned char \*)&aging, sizeof(NvAgingParam), &agingLen);

    AT\_CHECK\_RETURN(ret == ATS\_ERR\_SUCCESS, ATS\_ERR\_FAIL);

    ret = memcpy\_s(&g\_agingCtrl.nv, sizeof(NvAgingParam), &aging, sizeof(NvAgingParam));

    AT\_CHECK\_RETURN(ret == 0, ATS\_ERR\_FAIL);

#ifdef TEST\_LOG

    printf("cyc %u\n",g\_agingCtrl.nv.curCyc);

    printf("min %u\n",g\_agingCtrl.nv.curMin);

    printf("flag %u\n",g\_agingCtrl.nv.flag);

    printf("process %u\n",g\_agingCtrl.nv.process);

    printf("result %u\n",g\_agingCtrl.nv.result);

    printf("tar %u\n",g\_agingCtrl.nv.tarTime);

#endif

    switch (g\_agingCtrl.nv.flag) {

        case AGING\_IDLE: {

            ShowEnterAts();

            break;

        }

        case AGING\_WAIT: {

            ret = AgingInit(g\_agingCtrl.nv.tarTime);

            if (ret != ATS\_ERR\_SUCCESS) {

                ShowEnterAts();

                return ATS\_ERR\_FAIL;

            }

            break;

        }

        case AGING\_PROCESS: {

            /\* 如果发现上电进入运行态，认为是重启异常，写入错误码 \*/

            ret = OS\_CreateMutex(&g\_agingCtrl.mutex);

            AT\_CHECK\_RETURN(ret == RET\_OK, ATS\_ERR\_FAIL);

            (void)WriteErrMsg(ERR\_REBOOT);

            AgingShowResult(g\_agingCtrl.nv.result);

            break;

        }

        case AGING\_STOP: {

            AgingShowResult(g\_agingCtrl.nv.result);

            break;

        }

        default: {

            break;

        }

    }

    return ATS\_ERR\_SUCCESS;

}

1.5 int FactoryNvWrite(NvId nvId, unsigned char \*buff, unsigned int len)

|  |  |
| --- | --- |
| 函数原型 | int FactoryNvWrite(NvId nvId, unsigned char \*buff, unsigned int len) |
| 功能描述 | 工厂NV写入 |
| 输入参数 | nvId :NV标志位  buff :写到NV区的内容  len :写到NV区内容的长度 |
| 返回值 | 成功返回0，返回其他就是失败 |

函数实现：

/\* 工厂NV 写入 \*/

int FactoryNvWrite(NvId nvId, unsigned char \*buff, unsigned int len)

{

    AT\_CHECK\_RETURN(buff != NULL, ATS\_ERR\_FAIL);

    AT\_CHECK\_RETURN(nvId < ID\_MAX && nvId > ID\_NULL, ATS\_ERR\_FAIL);

    AT\_CHECK\_RETURN(g\_nvCtrl.isInit == 1, ATS\_ERR\_FAIL);

    NvCtrl \*nvCtrl = &g\_nvCtrl;

    AT\_CHECK\_RETURN(nvCtrl->index != NULL, ATS\_ERR\_FAIL);

    int ret = OS\_LockMutex(nvCtrl->mtuex);

    AT\_CHECK\_RETURN(ret == RET\_OK, ATS\_ERR\_FAIL);

    NvItemIndex \*nvIndex = &nvCtrl->index[nvId - 1];

    if (nvIndex == NULL || nvIndex->id != nvId || len > nvIndex->bufLen) {

        OS\_UnLockMutex(nvCtrl->mtuex);

        return ATS\_ERR\_FAIL;

    }

    NvStoreHook \*store = nvCtrl->store;

    if (store == NULL) {

        OS\_UnLockMutex(nvCtrl->mtuex);

        return ATS\_ERR\_FAIL;

    }

    unsigned char i;

    for (i = PART\_FLASH; i < PART\_ALL; i++) {

        /\* 加密长度增加 \*/

        store[i].buf.len = (nvIndex->encrypt == 1 ?

            (nvIndex->bufLen + sizeof(NvEncrypt) + g\_encryptType[FAC\_ENCRYPT\_VERSION - 1].addLen) : nvIndex->bufLen);

        store[i].buf.data = (unsigned char \*)OS\_Malloc(sizeof(unsigned char) \* store[i].buf.len);

        NV\_CHECK\_BREAK(store[i].buf.data != NULL);

        (void)memset\_s(store[i].buf.data, store[i].buf.len, 0, store[i].buf.len);

        if (nvIndex->encrypt == 1) {

            ret = NvEncryptData(store[i].buf.data, store[i].buf.len, buff, len);

            NV\_CHECK\_BREAK(ret == ATS\_ERR\_SUCCESS);

        } else {

            ret = memcpy\_s(store[i].buf.data, store[i].buf.len, buff, len);

            NV\_CHECK\_BREAK(ret == 0);

        }

        ret = store[i].write(nvId, store[i].buf.data, store[i].buf.len);

        NV\_CHECK\_BREAK(ret == ATS\_ERR\_SUCCESS);

    }

    for (i = PART\_FLASH; i < PART\_ALL; i++) {

        NV\_CHECK\_CONTINUE(store[i].buf.data != NULL);

        OS\_Free(store[i].buf.data);

        store[i].buf.data = NULL;

        store[i].buf.len = 0;

    }

    (void)OS\_UnLockMutex(nvCtrl->mtuex);

    return ret;

}

1.6 int FactoryNvRead(NvId nvId, unsigned char \*buff, unsigned int inLen, unsigned int \*outLen)

|  |  |
| --- | --- |
| 函数原型 | int FactoryNvRead(NvId nvId, unsigned char \*buff, unsigned int inLen, unsigned int \*outLen) |
| 功能描述 | 工厂NV读取 |
| 输入参数 | nvId :NV标志位  buff :读到NV区的内容  inLen:写到NV区内容的长度  outLen:读到NV区内容的长度 |
| 返回值 | 成功返回0，返回其他就是失败 |

函数实现：

/\* 工厂NV 读取 \*/

int FactoryNvRead(NvId nvId, unsigned char \*buff, unsigned int inLen, unsigned int \*outLen)

{

    AT\_CHECK\_RETURN(buff != NULL, ATS\_ERR\_FAIL);

    AT\_CHECK\_RETURN(outLen != NULL, ATS\_ERR\_FAIL);

    AT\_CHECK\_RETURN(nvId < ID\_MAX && nvId > ID\_NULL, ATS\_ERR\_FAIL);

    AT\_CHECK\_RETURN(g\_nvCtrl.isInit == 1, ATS\_ERR\_FAIL);

    NvCtrl \*nvCtrl = &g\_nvCtrl;

    AT\_CHECK\_RETURN(nvCtrl->index != NULL, ATS\_ERR\_FAIL);

    int ret = OS\_LockMutex(nvCtrl->mtuex);

    AT\_CHECK\_RETURN(ret == RET\_OK, ATS\_ERR\_FAIL);

    NvItemIndex \*nvIndex = &nvCtrl->index[nvId - 1];

    if (nvIndex == NULL || nvIndex->id != nvId || inLen < nvIndex->bufLen) {

        OS\_UnLockMutex(nvCtrl->mtuex);

        return ATS\_ERR\_FAIL;

    }

    NvStoreHook \*store = nvCtrl->store;

    if (store == NULL) {

        OS\_UnLockMutex(nvCtrl->mtuex);

        return ATS\_ERR\_FAIL;

    }

    NvPart use = PART\_NONE;

    unsigned char i;

    for (i = PART\_FLASH; i < PART\_ALL; i++) {

        FACT\_LOG("read part %u\n", i);

        if (nvIndex->encrypt == 1) {

            ret = ReadFormEncrypt(nvId, &store[i], outLen);

        } else {

            ret = ReadFromUnEncrypt(nvId, &store[i], outLen);

        }

        use = (ret == ATS\_ERR\_SUCCESS ? (use + (NvPart)i) : use);

    }

    /\* 如果有异常数据进行修复处理 \*/

    use = NvRepairData(nvId, use, store);

    if (use != PART\_NONE) {

        if (nvIndex->encrypt == 1) {

            ret = NvDecryptData(buff, inLen, store[use].buf.data, store[use].buf.len);

        } else {

            ret = memcpy\_s(buff, inLen, store[use].buf.data, store[use].buf.len);

        }

        use = (ret == ATS\_ERR\_SUCCESS ? use : PART\_NONE);

    }

    \*outLen = (((ret == ATS\_ERR\_SUCCESS) && (use != PART\_NONE)) ? nvIndex->bufLen : 0);

    for (i = PART\_FLASH; i < PART\_ALL; i++) {

        NV\_CHECK\_CONTINUE(store[i].buf.data != NULL);

        OS\_Free(store[i].buf.data);

        store[i].buf.data = NULL;

        store[i].buf.len = 0;

    }

    OS\_UnLockMutex(nvCtrl->mtuex);

    return (use == PART\_NONE ? ATS\_ERR\_FAIL : ATS\_ERR\_SUCCESS);

}