文档版本	V1.0.0
发布日期	2020-04-22

剑池产测工具二次开发指南

芯片开放社区
Open Chip Community

目录

剑池产测工具二次开发指南

1 术语与缩略语

Abbreviations缩略语	Full spelling英文全名	Chinese explanation中文解释
DUT	Device Under Test	待测设备
JTAG	Joint Test Action Group	联合测试行为组织
COM	cluster communication port	串口

2 简介

剑池产测工具是一款由阿里巴巴集团平头哥半导体有限公司开发的生产测试工具（支持Windows7/10 32/64位系统），能够支持多产品、多工位、多路并发的产测需求。
剑池产测工具二次开发涉及到工位配置文件编写和工位程序开发。剑池产测工具通过配置文件实现对工位程序调用，以适配多种DUT产品。

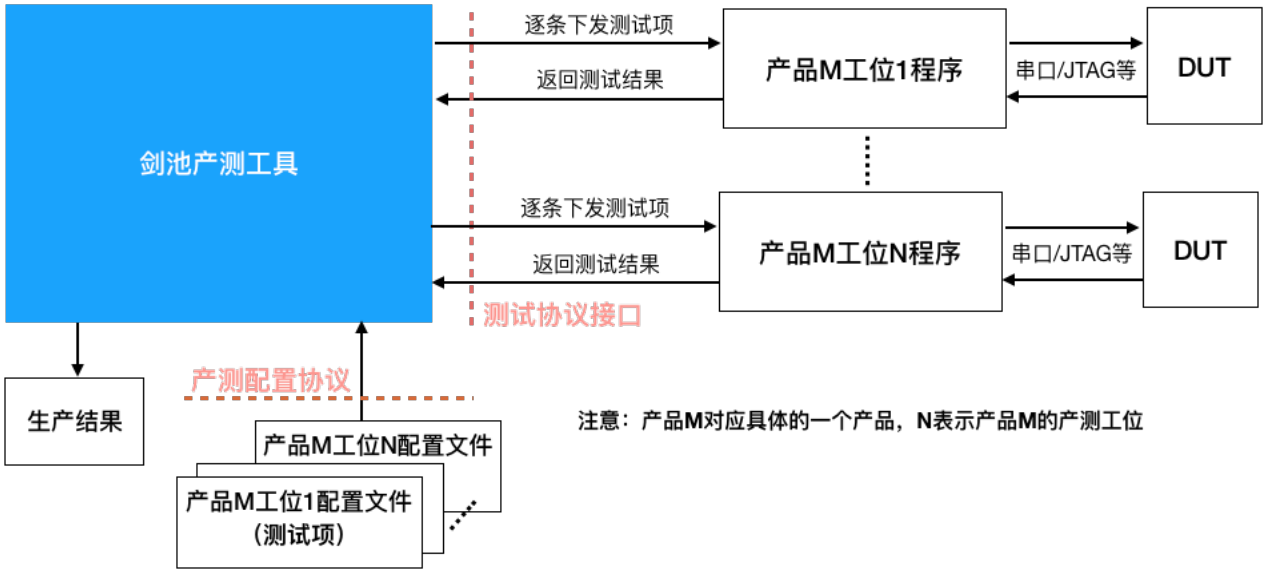
注意：快速上手详见《剑池产测工具二次开发上手手册》。

2.1 主要功能特性

- 可动态配置最多8路的并发产测测试
- 支持工位程序模块化、定制化开发
- 支持手动、自动测试模式切换
- 支持配置产测测试项，以及工位合并和拆分
- 支持在执行测试前启动后台服务程序（如ftp、http等服务）
- 支持生产数据加密

3 剑池产测工具架构

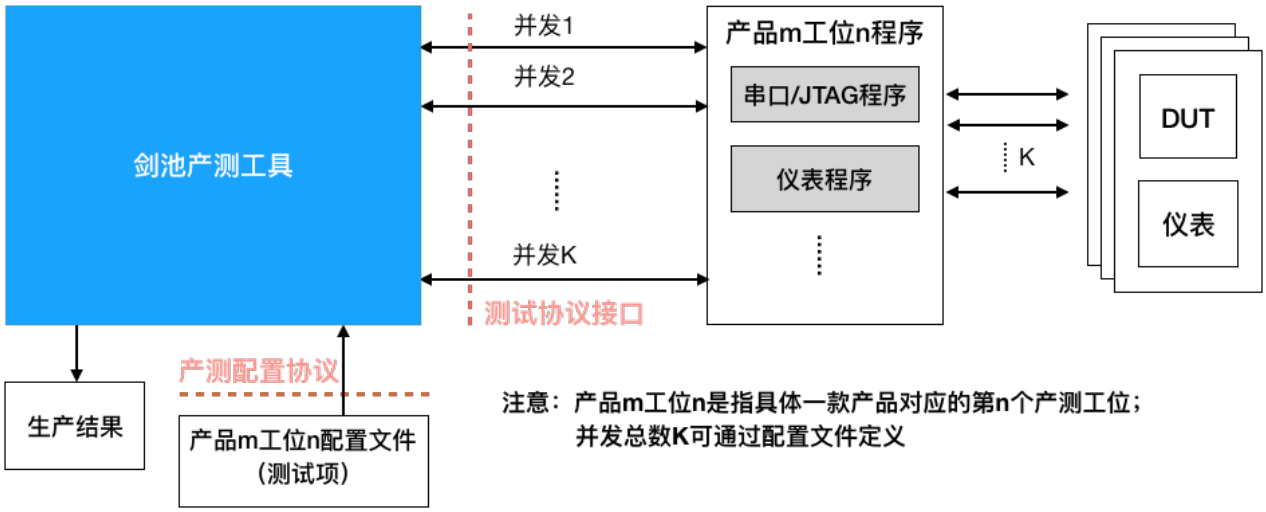
3.1 剑池产测工具整体架构



架构说明：

- 剑池产测工具与外部的接口主要由两部分组成：产测配置协议和测试协议接口。
- 产测配置协议可以按照不同的产品不同的工位配置对应的工位程序。
- 测试协议接口用于交换测试数据内容：包括参数传递和结果返回等。

3.2 产测工位架构



说明：

- 产测工位有一个或多个工位程序。
- 产测工位可以实现多路并发，提高产测效率。
- 工位程序中的接口程序需要按照测试协议接口实现
- 当需要多路并发时，工位程序需要确保多路硬件支持，保证多路之间互不影响。

4 产测配置协议

产测配置协议是剑池产测工具程序和工位配置文件之间的接口约束。

4.1 工位配置文件

工位配置文件必须为加密的zip格式，包含一个factory.json文件和一个可选的csv文件（生产数据文件）。
参考示例如下图：

其中factory.json是一个json格式的文件，demo.csv是生产数据文件。

注意：工位配置文件的生成可参考“剑池产测工具用户使用手册”中的打包工位配置文件章节。

4.2 factory.json文件

factory.json文件必须为有效的json格式文件，其中的内容区分大小写。
注意：factory.json文件以UTF-8编码格式保存。
格式定义如下：

```
{
  "version": "1.0.0",
  "name": "xxxx",
  "fileVersion": "x.y.z",
  "mode": "manual",
  "tasks": 1,
  "program": "xxxx",
  "productInfo": {...},
  "extraProductInfo": {...},
  "unconfigurable": [...],
  "configurable": [...],
  "preparation": [...],
  "cases": [...]
}
```

4.3 version

version定义了当前factory.json使用的产测配置协议的版本。
当前支持的版本为"1.0.0"。

4.4 name

name用于定义工位的名称，比如："工位1"。

4.5 fileVersion

fileVersion用于定义工位文件的版本，内容为用户自定义，会在剑池产测工具界面底部显示，可用于版本控制，比如"0.1.0"。

4.6 mode

mode是定义测试模式，当前可选的模式有"auto"和"manual"两种。其中"auto"为自动模式；"manual"为手动模式。
通过修改该参数实现手动、自动测试模式切换。

4.7 tasks

tasks定义并发的路数，值为[1-8]。当mode为"manual"时，tasks必须为1。
当mode为"auto"时，可动态配置最多8路的多路并发产测测试。

4.8 program

program定义工位程序路径，使用不同的工位程序可以适应不同产品的产测。
工位程序可以使用绝对路径或相对路径，比如"./aesl/aesl.exe"。
注意：相对路径以剑池产测工具安装目录为当前路径；使用路径"\"分割时，需要将其转化为"\\"，比如当相对路径为".\aesl\aesl.exe"时需要转换为".\\aesl\\asel.exe"。

4.9 productInfo和extraProductInfo

productInfo和extraProductInfo都是用于定义生产数据信息，都是可选参数，但是不能同时存在。当有执行的步骤定义burn（烧录/写入生产数据信息）为true时，必须存在一个。
productInfo一般用于首次或只有一次烧录/写入产品信息；而extraProductInfo用于二次烧录/写入产品信息且必须配合cases字段内部的deviceId一起使用。
在模组生产（如WIFI模组）和产品生产（如使用WIFI模组的故事机）分开时，模组生产（如WIFI模组的MAC地址等信息）使用productInfo，而产品生产（如故事机的SN等信息）使用extraProductInfo，其中的deviceId为模组烧录/写入时的唯一标识（如WIFI模组的MAC地址）。
格式定义如下：

```
{
  "file": "xxx.csv",
  "outputItems": [...]
}
```

file用于定义生产数据csv文件，比如"1.csv"；outputItems用于定义csv文件中的哪些列的内容需要导出，outputItems定义的内容必须是1.csv文件中的列名，且必须包括第一列的列名。
注意：

- **列名必须是由英文字母、_和数字组成，且以英文字母或_开头。**
- **列名（不区分大小写）不能是result、reason、starttime、totaltime、taskid、ordernum、seq和burnstate。**

参考示例：

```
{
  "file": "1.csv",
  "outputItems": ["SN"]
}
```

其中的SN为1.csv中的第一列列名
注意：生产数据导出可参考“剑池产测工具用户使用手册”中的生产测试数据导出章节。

4.10 unconfigurable

unconfigurable为可选项，用于定义不可调整的参数。主要用于cases定义中cmd的内容替换。
格式如下：

```
[
  {
    "key": "xxx",
    "name": "xxx",
    "value": "xxx"
  },
  {
    "key": "yyy",
    "name": "yyy",
    "value": "yyy"
  }
]
```

key值必须为全局唯一，name用于标记该键的作用或描述信息，value定义为实际的内容。
注意：

- **key值必须是由英文字母、_和数字组成，且以英文字母或_开头。**
- **key值（不区分大小写）不能是result、reason、starttime、totaltime、taskid、ordernum、seq和burnstate。**
- **value值不能包含空格、$、[、`、~、!、^、|、{、}、'、<、>、&和]字符。**

参考例子：

```
[
  {
    "key": "pn",
    "name": "产品名称",
    "value": "PN000AESL"
  },
  {
    "key": "image",
    "name": "烧录文件",
    "value": "image.hexf"
  }
]
```

在cases定义的cmd中，所有"pn"和"$image$"将会被替换为"PN000AESL"和"image.hexf"；在preparation定义的startCmd启动参数中，所有"pn"和"$image$"将会被替换为"PN000AESL"和"image.hexf"。
注意：可以包含多个key内容；区分大小写。

4.11 configurable

configurable为可选项，用于定义用户可通过剑池产测工具设置界面（参考"剑池产测工具用户使用手册"中的工位配置参数章节）调整的参数。configurable和unconfigurable的区别是，configurable用户可以修改值，而unconfigurable不可修改。
configurable有（key, value）和（key, values）两种形式。
格式如下：

```
[
  {
    "key": "xxx",
    "name": "xxx",
    "value": "xxx"
  },
  {
    "key": "xxx",
    "name": "xxx",
    "values": [...]
  }
]
```

其中name的内容会在设置页面显示；value定义为所有并发都是用相同的值；而values定义为每个并发都用各自的值（values内部的值不能重复）。
注意：

- **key值必须是由英文字母、_和数字组成，且以英文字母或_开头。**
- **key值（不区分大小写）不能是result、reason、starttime、totaltime、taskid、ordernum、seq和burnstate。**
- **value(s)值不能包含空格、$、[、`、~、!、^、|、{、}、'、<、>、&和]字符。**

参考示例如下：

```
[
  {
    "key": "baud",
    "name": "波特率",
    "value": "115200"
  },
  {
    "key": "com",
    "name": "串口",
    "values": ["COM1", "COM2", "COM3", "COM4", "COM5", "COM6", "COM7", "COM8"]
  }
]
```

在cases定义的cmd中，所有"$baud$"会被替换为115200"，所有com"将会被替换为相应并发的值（如第二路替换为"COM2"，第四路替换为"COM4"）。

4.12 preparation

preparation为可选参数，一般用于在执行测试前启动一些服务程序（如ftp、http等服务）。
剑池产测工具与服务程序之间没有接口，只会调用启动和停止服务程序操作。
需要注意的是每次进入到生产测试界面都会执行一遍preparation的启动操作，从生产测试界面离开都会执行一遍preparation的停止操作。如在生产测试界面通过点击设置后进入到设置界面时，剑池产测工具会执行一遍preparation的停止操作；而从设置界面返回，剑池产测工具又会执行一遍preparation的启动操作。
格式如下：

```
[
  {
    "ordernum": 1,
    "name": "xxx",
    "startCmd": "yyy",
    "stopCmd": "zzz",
    "delay": 1000
  },
  ...
]
```

ordernum、name和startCmd为必选参数；而delay和stopCmd为可选参数。

4.12.1 ordernum

ordernum用于定义preparation中执行的顺序（从小到大顺序执行），从1开始编号，可不连续但不可重复（在preparation内）。

4.12.2 name

name用于定义该启动项的名称，当执行该启动项时会显示在剑池产测工具生产测试界面的弹框里面。

4.12.3 startCmd

startCmd用于定义执行该启动项的命令。
startCmd内容包括启动程序和可选的启动参数两部分。两部分之间使用空格分隔；启动程序名称不能包含$、[、`、~、!、^、|、{、}、'、<、>、&和]字符；启动参数可以动态引用在unconfigurable定义的所有key内容，以及在configurable中定义的(key, value)内容。
参考示例如下：
在configurable中定义：

```
{
    "key": "ftpport",
    "name": "ftp端口号",
    "value": "12300"
}
```

在preparation中定义：

```
{
    "ordernum": 1,
    "name": "ftp服务",
    "startCmd": "demo/ftp.exe start -port $ftpport$",
    "stopCmd": "demo/ftp.exe stop",
    "delay": 1000
}
```

启动程序为"demo/ftp.exe"；启动参数为"start -port $ftpport$"，其中"$ftpport$"会被替换为"12300"。
注意：

- **启动参数不能引用由configurable中定义的(key, values)的内容（如不能引用在4.11例子中定义的com参数）**
- **startCmd与cmd（详见4.13.6.5节）的区别：startCmd第一个部分必须是start程序名，而cmd的程序名由program参数定义；执行startCmd时不会额外添加任何参数，而执行cmd时会添加如"--taskid 1 --timeout 3000"的参数。**

4.12.4 stopCmd

stopCmd为可选的参数，用于定义执行该停止项的命令。如果未定义该参数，剑池产测工具会给进程发送kill信号（SIGTERM）。在实际应用中，一般情况下不需要定义stopCmd参数。
stopCmd内容包括停止程序和可选的停止参数两部分。两部分之间使用空格分隔；停止程序名称不能包含$、[、`、~、!、^、|、{、}、'、<、>、&和]字符；停止参数可以动态引用在unconfigurable定义的所有key内容，以及在configurable中定义的(key, value)内容。
参考示例如下：

```
"stopCmd": "demo/ftp.exe stop"
```

停止程序为"demo/ftp.exe"；停止参数为"stop"。
注意：

- **停止参数不能引用在configurable中定义的(key, values)的内容（如不能引用在4.11例子中定义的com参数）**
- **stopCmd与cmd（详见4.13.6.5节）的区别：stopCmd第一个部分必须是stop程序名，而cmd的程序名由program参数定义；执行stopCmd时不会额外添加任何参数，而执行cmd时会添加如"--taskid 1 --timeout 3000"的参数。**

4.12.5 delay

delay为可选的参数，定义为启动该步骤前需要等待的时间，单位为ms。

4.13 cases

cases用于定义测试项列表、测试顺序。
格式为：

```
[
  {
    "ordernum": 1,
    "name": "xxx",
    "selected": true,
    "program": "xxx",
    "autoOnly": true,
    "steps": [...]
  },
  ...
]
```

4.13.1 ordernum

ordernum用于定义测试执行的顺序（从小到大顺序执行），从1开始编号，可不连续但不可重复（在cases内）。

4.13.2 name

name用于定义该测试项的名称，该名称会显示在剑池产测工具生产测试界面的测试项里面。

4.13.3 selected

selected是可选的bool值参数，用于定义是否执行该测试项。当不存在该参数时，默认为执行该测试项。

4.13.4 program

program为可选参数，定义内容参考4.8节。当该参数在测试项中存在时，会替换外部定义的program值。每个测试项都可以定义program。
实际应用中，可以将一个复杂的工位程序拆分为多个简单的工位程序，配合该参数完成相同的功能；同时也可以将多个功能的工位程序组合，实现工位的合并。

4.13.5 autoOnly

autoOnly为可选的bool值参数。当为true时，该测试项只有在测试模式为"auto"时可以被选中；当为false时，相当于未定义该参数。

4.13.6 steps

steps用于定义测试项中的测试步骤列表及步骤顺序。
格式如下：

```
[
  {
    "seq": 1,
    "name": "xxx",
    "delay": 100,
    "timeout": 3000,
    "cmd": "xxx $yy$ zzz",
    "result": "xxx",
    "selected": true,
    "prompt": "hint message",
    "confirm": true,
    "precondition": true,
    "query": true,
    "condition": true,
    "deviceId": "xx",
    "extra": [...],
    "burn": true,
    "postcondition": false,
  },
  ...
]
```

4.13.6.1 seq

seq定义测试项的步骤顺序（从小到大顺序执行），从1开始编号，可不连续但不可重复（在测试项内）。

4.13.6.2 name

name用于标记该步骤的名称，该名称会显示在剑池产测工具生产测试界面的步骤里面。

4.13.6.3 delay

delay为可选的参数，定义为执行该步骤前需要等待的时间，单位为ms。

4.13.6.4 timeout

timeout定义为执行该步骤最大的时间，单位为ms。当工位程序执行时间超过该值还没有返回结果时，剑池产测工具将认为测试超时，执行失败。
delay定义的等待时间不会被计算到timeout定义的时间内，两个参数相互独立。confirm（详见4.13.6.9）参数和precondition（详见4.13.6.10）参数定义的弹框提示不受timeout限制，即弹框提示不会退出直到用户点击操作。

注意：参数定义的内容会在执行cmd时传给工位程序，如timeout定义为3000（即3000ms），那么执行时会以"--timeout 3000"的方式传递给工位程序（参考5.1节）。

4.13.6.5 cmd

cmd定义为剑池产测工具执行该步骤调用工位程序时需要传递的额外命令参数。cmd内容以空格分隔，以'$'作为特殊字符，其中以两个'$'包括的内容会被动态替换。
剑池产测工具只会动态替换cmd内容并下发给工位程序，而工位程序执行解析参数。
参考例子：

```
"cmd": "flash -P $com$ -F $baud$ -w image.hexf",
"cmd": "ATCMD: AT+MAC=$WIFI_MAC$,$BT_MAC$ -P $com$ -F $baud$",
"cmd": ""
```

命令下发参考"测试协议接口"章节。
注意：

- **两个'$'之间的内容（即需要被动态替换的内容）要求由英文字母、_和数字组成，且以英文字母或_开头。**
- **动态替换需要满足参数已在configurable/unconfigurable定义，或剑池产测工具已经从生产数据中取到，或工位程序已通过extra字段返回结果**
- **cmd和startCmd的区别参见4.12.3节**

4.13.6.6 result

result定义步骤执行成功时的检测内容。剑池产测工具以该内容和工位程序返回的result值进行比对来判断结果。参考"测试协议接口"章节。
参考例子：

```
"result": "SUCCESS"
```

4.13.6.7 selected

selected为可选的bool参数。当为false时，执行测试会跳过该步骤；当不存在该参数时，默认会执行。

4.13.6.8 prompt

prompt为可选参数，定义提示的内容。当执行该步骤时剑池产测工具弹框提示该内容。
当prompt存在且该步骤选中时，对应的该测试项不能定义autoOnly为true；当该测试项也选中时，测试模式不能为"auto"。

4.13.6.9 confirm

confirm为可选的bool参数，定义结果需要用户确认，需要配合prompt一起使用。当为true时，剑池产测工具一旦检测到测试结果成功就会弹框提示prompt内容，等待用户确认结果（有确定和取消两个按钮）。弹框提示不受timeout（详见4.13.6.4）参数限制直到用户点击操作。
常用于led灯等需要用户确认的复杂产测场景测试。

4.13.6.10 precondition

precondition为可选的bool参数，需要配合prompt一起使用。当为true时，剑池产测工具会先弹框提示prompt内容，等待用户执行完某些操作并点击确认（只有确定一个按钮）后，再执行该步骤。弹框提示不受timeout（详见4.13.6.4）参数限制直到用户点击操作。

4.13.6.11 burn

burn为可选的bool参数，当定义为true时，剑池产测工具会从生产数据中取出一条生产信息用于烧录或写入。

4.13.6.12 query

query为可选参数，常用于烧录测试项中检测DUT是否为新的板子。当该参数定义时，工位程序返回结果会包含extra内容，剑池产测工具检测到result成功后，会先查询内部生产数据，如果不存在（即DUT为新的板子）则成功，并继续执行下一个步骤（通常为烧录步骤）；如果存在（即DUT为已烧录的板子）则执行失败。
参考例子：

```
"query": "SN"
```

SN为生产数据的主键列，当查询存在，意味着该DUT已烧录/写入生产数据。
返回结果例子：

```
{"result": "SUCCESS", "extra": [{"SN": "FFFFFFFFFF"}]}
```

注意：

- **query值必须是由英文字母、_和数字组成，且以英文字母或_开头。**
- **query值（不区分大小写）不能是result、reason、starttime、totaltime、taskid、ordernum、seq和burnstate。**

4.13.6.13 condition

condition为可选bool参数，当定义为true时，该步骤执行失败后剑池产测工具会反复执行该步骤。
常与query一起使用，用于在烧录/写入生产数据之前检测DUT是否为新的板子，当执行成功后执行后续的烧录/写入（burn为true）步骤。

4.13.6.14 deviceId

deviceId为可选参数，常用于测试工位。当该参数定义时，工位程序返回结果会包含extra内容。
在实际产测中会分多个工位来完成，第一个工位通常是烧录工位（烧录唯一标识信息到DUT），后续工位是测试工位（从DUT读取唯一标识信息）。测试工位需要先检测板子的信息，再执行具体的测试内容。而deviceId用于标记DUT唯一标识的名称（比如SN）。
参考例子：

```
"deviceId": "SN"
```

SN为生产数据的主键列，用于区分不同的DUT板子。
返回结果例子：

```
{"result": "SUCCESS", "extra": [{"SN": "SN0001"}]}
```

deviceId和query的区别：query会查询生产数据是否存在来决定测试结果，而deviceId只会更新内部数据库并不判断测试结果。
注意：

- **deviceId值必须是由英文字母、_和数字组成，且以英文字母或_开头。**
- **deviceId值（不区分大小写）不能是result、reason、starttime、totaltime、taskid、ordernum、seq和burnstate。**

4.13.6.15 extra

extra为可选参数，用于定义返回结果中extra内部的内容。某些测试步骤除了返回测试结果是失败或成功外，还需要返回额外的信息，比如电流测试等。
参考例子：

```
"extra": ["tx_cur", "rx_cur"]
```

extra定义了两个参数tx_cur和rx_cur。
返回结果例子：

```
{"result": "SUCCESS", "extra": [{"tx_cur": "6.0mA", "rx_cur": "0.51mA"}]}
{"result": "SUCCESS", "extra": [{"tx_cur": "6.0mA"}, {"rx_cur": "0.51mA"}]}
{"result": "SUCCESS", "extra": [{"tx_cur": "6.0mA", "rx_cur": "0.5mA"},
{"tx_cur": "6.1mA", "rx_cur": "0.51mA"}]}
```

extra的返回结果会在导出生产数据中记录。
注意：

- **extra可以返回多个测试结果。**
- **extra值必须是由英文字母、_和数字组成，且以英文字母或_开头。**
- **extra值（不区分大小写）不能是result、reason、starttime、totaltime、taskid、ordernum、seq和burnstate。**

4.13.6.16 postcondition

postcondition为可选的bool参数，用于定义后置条件，常与"auto"测试模式一起使用，用于检测某个状态的变化。比如当烧录完成后需要等待更换DUT，再执行下一轮的测试；当测试内容执行完毕后等待更换新的DUT，再执行测试。
postcondition可用于卡序操作而不需要用户手工参与。

4.13.6.17 部分可选参数的兼容表

	prompt	confirm	precondition	query	condition	deviceId	extra	burn	postcondition
prompt	OK								
confirm	Must	X							
precondition	Must	X	X						
query	OK	X	OK*	OK					
condition	X	X	X	OK	OK				
deviceId	OK	X	OK*	X	OK	OK			
extra	OK	X	OK*	X	X	X	OK		
burn	X	X	X	X	X	X	X	OK	
postcondition	X	X	X	X	X	X	X	X	OK

- 同一字段表示能否独立使用，如prompt&prompt表示prompt可以单独使用
- Must表示必须一起使用，如confirm或precondition，必须和prompt一起使用
- OK表示可以一起使用；OK*表示有条件使用，如query&precondition，是指query&precondition&prompt一起使用
- X表示不可以一起使用
- bool类型值为false等效于不存在该参数

4.14 csv生产数据文件

csv生产数据文件必须为有效的csv格式（以","分割）。数据的列名不能有空格，不能重复。其中第一列为主键列，具有唯一性。
参考示例如下：

	A	B	C
1	SN	SECRET	
2	AA0000000001	secret11111	
3	AA0000000002	secret11112	
4	AA0000000003	secret11113	
5	AA0000000004	secret11114	
6	AA0000000005	secret11115	
7	AA0000000006	secret11116	
8	AA0000000007	secret11117	
9	AA0000000008	secret11118	
10	AA0000000009	secret11119	
11	AA0000000010	secret11120	
12	AA0000000011	secret11121	

其中的SN列为产品的SN，是生产数据的主键列。
注意：

- **数据列名必须是由英文字母、_和数字组成，且以英文字母或_开头。**
- **数据列名（不区分大小写）不能是result、reason、starttime、totaltime、taskid、ordernum、seq和burnstate。**
- **数据列名不能和configurable、unconfigurable、extra中定义的key值相同**

5 测试协议接口

测试协议接口是剑池产测工具和工位程序之间的接口，包括调用格式（下行）、结果返回（上行）和特殊处理。

5.1 调用格式

剑池产测工具调用工位程序时会固定添加taskid和timeout在cmd前面。taskid为并发id号（从1开始编号），timeout（timeout参数详见4.13.6.4节）为配置的超时时间。
格式为：

```
工位程序+"taskid和timeout"+cmd内容
```

参考例子如下：

```
aesl/aesl.exe --taskid 1 --timeout 3000 flash -P COM1 -F 115200 -w image.hexf
```

其中的"aesl/aesl.exe"由program定义，taskid后面的1为第1路并发，timeout后面的3000为超时时间3000ms，“flash -P COM1 -F 115200 -w image.hexf”为cmd的配置内容，传递给工位程序做处理。

5.2 结果返回

工位程序通过标准输出口stdout返回测试结果以及打印信息。
测试结果为json格式（result、reason和extra为关键字，其中的reason和extra可选），如下：

```
{"result":"xxx", "reason":"xxx", "extra":[自定义对象列表]}
```

参考例子如下：

```
{"result":"SUCCESS"}
{"result":"FAIL", "reason":"BLE disconnect"}
{"result":"SUCCESS", "extra":[{"SN":"SN00001"}]}
{"result":"FAIL", "extra":[{"rx_cur":"2.5mA", "tx_cur":"6.0mA"},
{"rx_cur":"2.51mA", "tx_cur":"6.01mA"}]}
```

其中的SN、rx_cur、tx_cur在工位配置文件中定义。

注意：中文字符需要以UTF-8编码，否则会显示乱码；用户自定义打印信息请不要包含字符"{"和"}"。

5.3 特殊处理

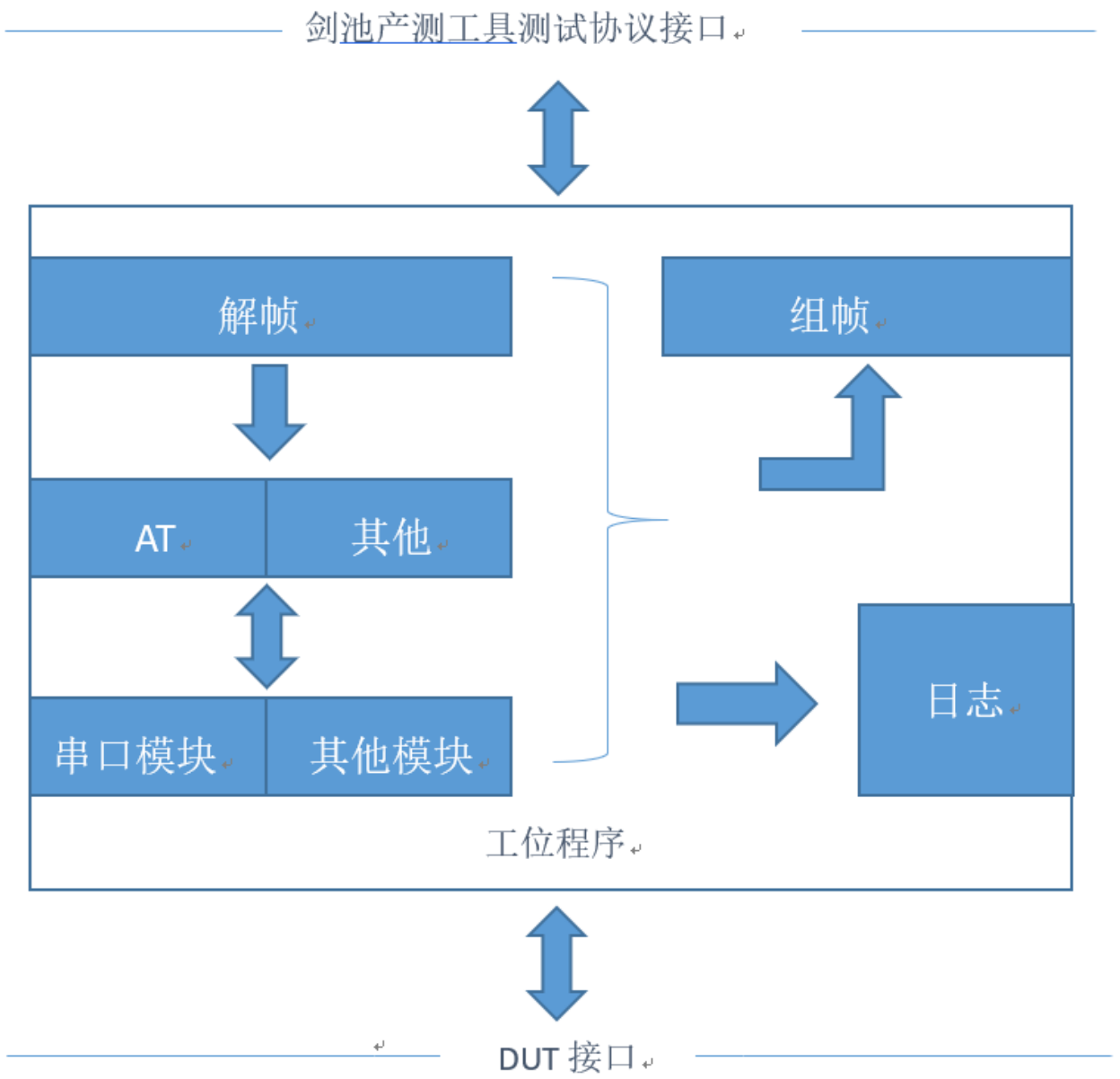
剑池产测工具在检测到结果或者超时都会发送kill信号（SIGTERM），为防止工位程序异常退出，工位程序需要确保执行完成之后再上报结果，或和配置文件约定好超时时间。

6 工位程序开发

工位程序对接剑池产测工具和适配DUT。针对特定功能需求，如仪表控制，复位等，工位程序可单独封装成模块供调用。
本章以HEAD_ProductionExample例程为例，介绍工位程序开发，用户可通过《剑池产测工具二次开发上手手册》4.2.1章节下载，以下是代码目录说明。

文件/文件夹	说明
ComAtCommand	串口驱动及AT指令封装文件夹
DispenseCall	分发功能函数文件夹
dist	编译打包文件夹
log	日志模块文件夹
modules	模块化相关代码示例
Output	输出接口文件夹（对接剑池产测工具）
sound_verify	用户定义文件夹
SoundConnect	用户定义文件夹
Make.bat	编译脚本文件
LICENSE	license文件
T-HEAD_ProductionExample.py	程序入口文件
version	版本信息文件
DUTSimulator.exe	DUT模拟程序，用来演示目的
OpenDUTSimulator.bat	启动模拟DUT程序脚本文件
run.bat	启动工位程序运行脚本

6.1 工位程序架构



6.2 测试协议接口

测试协议接口分为上行接口和下行接口。接口规范详见第5章，接口内容由配置文件定义。

6.2.1 下行接口

下行接口为剑池产测工具对工位程序下发控制的接口，格式如5.1章节所示。

6.2.2 上行接口

上行接口为工位程序上报给剑池产测工具的接口，格式如图5.2章节所示。

6.3 解帧和组帧

解帧函数：

```
paracheck()
parasplit()
```

- paracheck函数实现数据有效性检测。
- parasplit函数实现解析。

组帧函数：

```
format_output(result, reason=None, extitle=None, extra=None)
```

参数值分别对应如下"xxx"的值：

```
{"result":"xxx", "reason":"xxx", "extra":[{"xxx":"xxx"}]}
```

6.4 通用功能封装

功能封装按类型分为AT封装(纯AT指令)、非AT指令封装、组合封装(AT和其他指令配合)。三种类型里AT指令通用性较强，AT指令格式如6.4.1章节介绍。
AT通用功能配置如下,其中参数名CALLMETHOD代表AT类型,有SERIALNORMAL，SERIALWRITE，SERIALLOOPGET三种类型，参数值SERIALWRITE 对应标准AT下发，SERIALNORMAL对应标准AT回读数据，SERIALLOOPGET对应标准AT循环读取，这三种AT指令类型只需配置即可：

```
 'SERIALNORMAL': [[u'SERCOM', u'SERBAUD', u'ATCMD', u'SERDELAY'],
'funcatexecuteread'],
 'SERIALWRITE': [[u'SERCOM', u'SERBAUD', u'ATCMD', u'SERDELAY'],
'funcatexecutewrite'],
 'SERIALLOOPGET': [[u'SERCOM', u'SERBAUD', u'ATCMD', u'SERDELAY',
u'LOOPTIMES'], 'funcatexecuteloopget'],
```

6.4.1 AT指令格式

AT协议采用基于ASCII码格式(指令或返回后面固定加<\r\n>)
AT请求指令(上位机下发)

类型	指令格式	描述
AT测试查询指令	AT	查询是否处于AT模式
查询参数指令	AT+<CMD>=?	用于查询设置指令的参数以及取值范围
查询指令	AT+<CMD>?	返回参数的当前值
设置指令	AT+<CMD>=<params>	设置用户自定义的参数值
执行指令	AT+<CMD>	用于执行受内部控制的功能

AT对应查询参数指令和查询指令响应格式：
    +<CMD>:<response>
    <STATUS>
AT对应设置指令和执行指令响应格式：
    <STATUS>

域	描述
<CMD>	命令字符串，可以使用'_'来分割
<params>	指令参数列表值，使用","分割
+<CMD>:	当响应测试命令和查询命令时，必须已该格式开头，如"+Sleep: mode"
<response>	响应的内容
<STATUS>	状态值：以OK或ERROR开头（注意是大写），后面加可选的原因，以":"分割

注意：<>只是用于标识

6.4.2 标准AT下发

通用AT指令下发以'OK'返回，例如写MAC地址AT+MAC=11:22:33:55:66:88不需添加commands列表，判断是否在位指令AT也不需添加commands列表，只需修改接口命令即可。
如配置工位程序原始格式如下：

```
T-HEAD_ProductionExample.exe --taskid 0 --timeout 3000 CALLMETHOD: SERIALWRITE
SERCOM: COM3 SERBAUD: 115200 ATCMD: AT+MAC=11:22:33:55:66:88 SERDELAY: 1000
```

6.4.3 标准AT回读数据

标准回读数据以+<CMD>：字段开始，要求回复格式按标准格式返回：例如读取版本号命令"AT+SWVER?"，回读数据+SWVER:V1.0.1，上位机解析字段V1.0.1作为版本号上报给剑池产测工具，标准回读数据接口命令如下：

```
T-HEAD_ProductionExample.exe --taskid 0 --timeout 3000 CALLMETHOD:
SERIALNORMAL SERCOM: COM3 SERBAUD: 115200 ATCMD: AT+SWVER? SERDELAY: 1000
```

6.4.4 标准AT循环读取

循环读取测试前常用于检测板子是否在位，测试完毕后用于检测板子是否拿走。实现在位检测与卡序的目的。
循环读取CALLMETHOD字段为SERIALLOOPGET，LOOPTIMES代表读取次数，接口格式如下。

```
T-HEAD_ProductionExample.exe --taskid 0 --timeout 3000 CALLMETHOD:
SERIALLOOPGET SERCOM: COM3 SERBAUD: 115200 ATCMD: AT SERDELAY: 1000
LOOPTIMES:5
```

6.5 用户增加功能

增加功能包括非标准AT和其他功能，其他功能可以是非AT或者AT加其他功能的组合。

6.5.1 非标准AT功能

针对非标准AT类型，用户需增加处理函数处理。
需用户添加列表和解析函数，如下commands列表没有添加处理函数默认执行通用AT指令下发(非标准AT指令CALLMETHOD字段值是SERIALWRITE)。如下所示设置接收模式AT+RXMODE,查询复位信息AT+RESET?都是非标准回读数据。需单独添加处理函数解析下位机返回字段。
接口格式如下：

```
T-HEAD_ProductionExample.exe --taskid 0 --timeout 3000 CALLMETHOD: SERIALWRITE
SERCOM: COM3 SERBAUD: 115200 ATCMD: AT+RXMODE SERDELAY: 1000
```

处理函数列表：

```
commands = {
  u'AT+RXMODE': self.getrxmodefunc,
  u'AT+RESET?': self.getresetfunc
}
```

6.5.2 其他功能

其他功能需要用户在COMMANDS_TABLE添加解析字段，并添加处理函数。如下所示，添加'COMBINE'字段，传参名是'SOUNDFUNC','AFECFG','FILEGET','SOUNDDELAY','SERCOM','SERBAUD','ATCMD','SERDELAY'：

```
'COMBINE': [[u'SOUNDFUNC', u'AFECFG', u'FILEGET', u'SOUNDDELAY', u'SERCOM',
u'SERBAUD', u'ATCMD', u'SERDELAY'], 'funcatexecutecombine']
```

参数值由程序框架解析出来，作为参数传入函数'funcatexecutecombine'执行。

6.6 驱动模块

驱动模块实现DUT或仪表等接口对接,驱动模块具有下发指令和回读数据的功能，一般封装成单独类。如串口驱动模块以ComThread封装成单独的类(工程目录下ComThread.py文件)，具备发送数据，接收数据，打开串口，关闭串口等函数接口供调用。

```
class ComThread
```

6.7 日志保存

日志模块提供了日志写入接口函数用于写入指定文件数据(工程目录下log文件夹)。

```
Print_Log(line, data)
```

日志文件以record_x.log命名，其中x代表实际通道，在多路测试时使用。
如同时进行八路测试，按传参line值不同会生成record_1.log 到 record_8.log 数据，并以时间戳记录保存。

7 工位程序模块扩展

工位程序模块扩展主要是以功能块为封装，使用时可直接调用。
本节以继电器复位程序及电流测量仪表程序为示例。

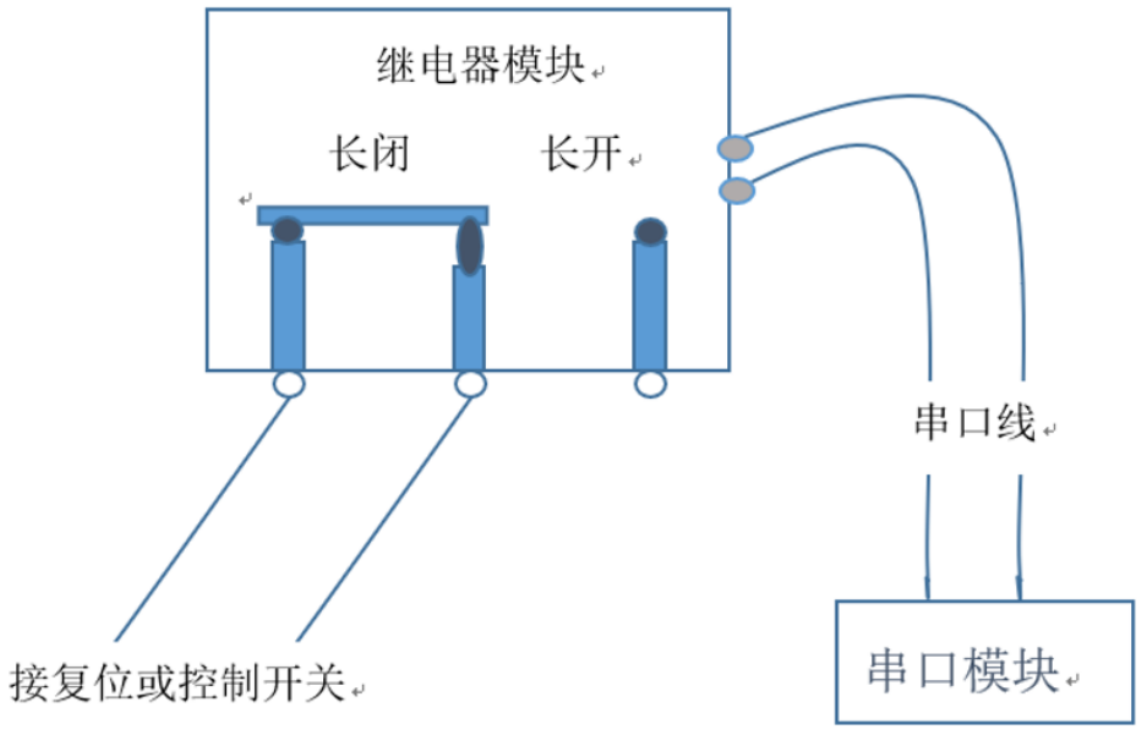
7.1 继电器复位模块

继电器复位模块支持艾尔赛USB继电器(LCUS-1)的型号，继电器复位模块如下所示：

关于LCUS-1继电器的更多内容详见该继电器的使用说明。
该模块已集成到剑池产测工具中，程序在剑池产测工具安装目录下的T-Head_CH340_Relay文件夹。
注意：在使用该模块化功能之前需要安装对应的驱动。

7.1.1 继电器复位硬件连接

复位程序通过串口模块控制继电器，弹片和长闭连接时，复位线连接。弹片和长闭连接时，复位线断开，程序通过控制继电器模块长开还是长闭达到复位或断电目的。



7.1.2 使用方法

使用方法以继电器串口号为COM3示例，若实际对应的串口号不同，请做相应的修改。

- 长开

```
T-Head_CH340_Relay.exe --taskid 1 --timeout 3000 COM3 wakeupon
```

- 长闭

```
T-Head_CH340_Relay.exe --taskid 1 --timeout 3000 COM3 wakeupoff
```

- 重启（执行顺序：长开，延时，长闭）

```
T-Head_CH340_Relay.exe --taskid 1 --timeout 3000 COM3 reset
```

7.1.3 继电器工位配置文件演示

```
{
    "version": "1.0.0",
    "name": "继电器复位演示",
    "fileVersion": "0.0.1",
    "mode": "manual",
    "program": "T-Head_CH340_Relay/T-Head_CH340_Relay.exe",
    "tasks": 1,
    "configurable": [
        {
            "key": "COMRELAY",
            "name": "继电器串口",
            "value": "COM3"
        }
    ],
    "cases": [
        {
            "ordernum": 6,
            "name": "模组复位",
            "program": "T-Head_CH340_Relay/T-Head_CH340_Relay.exe",
            "steps": [
                {
                    "seq": 1,
                    "name": "模组进入复位",
                    "cmd": "$COMRELAY$ wakeupon",
                    "timeout": 2000,
                    "result": "SUCCESS"
                },
                {
                    "seq": 2,
                    "name": "模组退出复位",
                    "cmd": "$COMRELAY$ wakeupoff",
                    "timeout": 3000,
                    "delay": 3000,
                    "result": "SUCCESS"
                }
            ]
        }
    ]
```

```
}
```

7.1.4 继电器复位程序

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

'''CH340 Relay Controler'''

import time
import sys
import serial


# format json output
def format_output(result, reason=None, extitle=None, extra=None):
    '''format_output with (result, reason, extitle, extra)'''
    message = '{'
    message += '"result":"' + str(result.encode('unicode_escape').decode('utf-
8')) + '"'
    if reason is not None:  # Fail case: add reason
        message += ',"reason":"' +
str(reason.encode('unicode_escape').decode('utf-8')) + '"'
    if extra is not None:
        message += ',"extra":[{"' +
str(extitle.encode('unicode_escape').decode('utf-8')) + '":"'
        message += str(extra.encode('unicode_escape').decode('utf-8')) + '"}]'
    message += '}'
    print(message)


def wakeupon(comport):
    '''wakeup turn on'''
    try:
        wakeser = serial.Serial(comport, 9600, timeout=1, write_timeout=0.5)

        wakeser.write(b'\xa0\x01\x01\xa2')
        time.sleep(0.1)
        wakeser.close()
        return True
    except serial.SerialException as ex:
        print(ex)
        return False


def wakeupoff(comport):
    '''wakeup turn off'''
    try:
```

```
        wakeser = serial.Serial(comport, 9600, timeout=1, write_timeout=0.5)

        wakeser.write(b'\xa0\x01\x00\xa1')
        time.sleep(0.1)
        wakeser.close()
        return True
    except serial.SerialException as ex:
        print(ex)
        return False


def reset(comport):
    try:
        wakeser = serial.Serial(comport, 9600, timeout=1, write_timeout=0.5)
        wakeser.write(b'\xa0\x01\x01\xa2')  # on
        time.sleep(5)
        wakeser.write(b'\xa0\x01\x00\xa1')  # off
        time.sleep(0.1)
        wakeser.close()
        return True
    except serial.SerialException as ex:
        print(ex)
        return False


##############################################################################
# <comport> <command>
def check_func_resp(comport, command):
    commands = {
        u'wakeupon': wakeupon,
        u'wakeupoff': wakeupoff,
        u'reset': reset
    }
    command_match = False
    for key_word in commands:
        if key_word in command.lower():
            response = commands[key_word](comport)
            command_match = True
            break
    if not command_match:
        format_output("FAIL", "命令不支持:" + str(command))
        return False
    if not response:
        #  print (key_word + ' execute err')
        format_output("FAIL", "执行失败:" + str(command))
        return False

    format_output("SUCCESS")
    return True
```

```
if __name__ == '__main__':

    print(str(sys.argv))
    if len(sys.argv) < 7:
        print("参数异常:")
        format_output("FAIL", "参数异常")
    else:
        try:
            check_func_resp(sys.argv[5], sys.argv[6])
        except Exception as ex:
            print(str(ex))
            format_output("FAIL", "参数异常")
```

7.2 dm3058电流测量模块

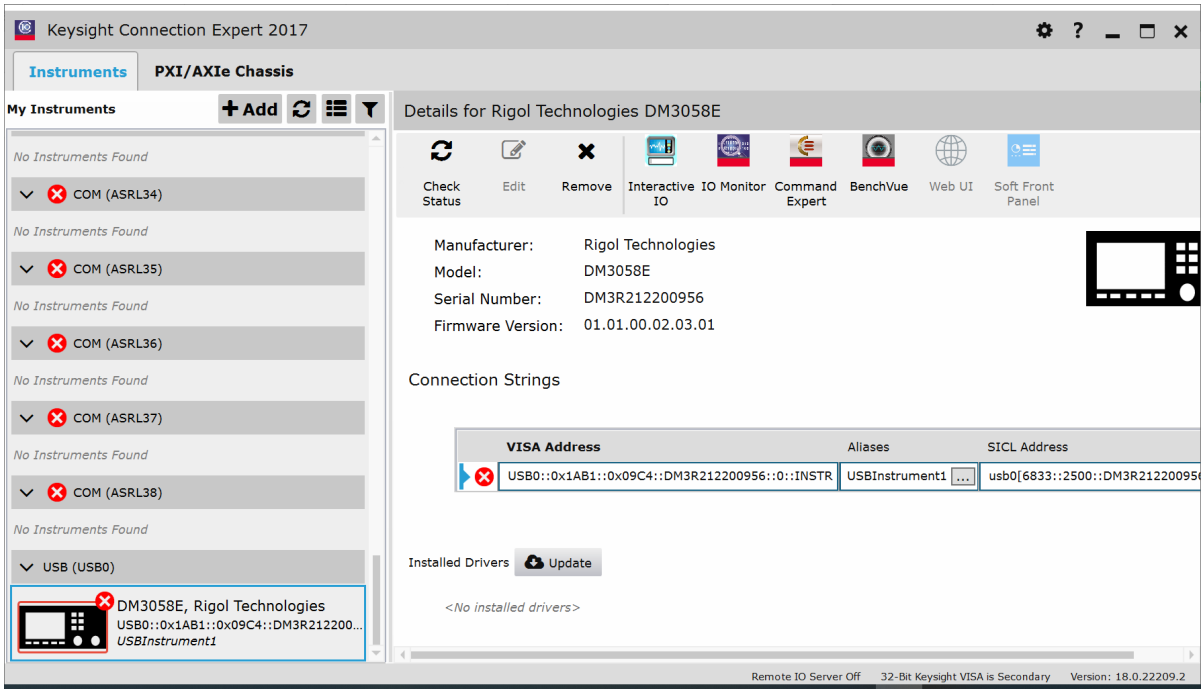
dm3058为RIGOL生产的程控万用表，可以程控测量电流，测量之前需要安装IOLibSuite仪表驱动程序，本例程介绍使用方法。

注意：该模块未集成到剑池产测工具中。

7.2.1 使用方法

```
dm3058.exe --taskid 1 --timeout 3000 instrid 10
```

- instrid是电源id，类似于"USB0::0x1AB1::0x09C4::DM3R212200956::0::INSTR"这种格式，安装IOLibSuite并打开，VISA Adress即为电源id（仪表连接成功显示对号，否则无法控制仪表），如下图所示。



- 10是测量时间，单位ms。

7.2.2 电源测量工位配置文件演示

```
{
    "version": "1.0.0",
    "name": "功耗测量工位",
    "fileVersion": "1.0",
    "mode": "manual",
    "program": "dm3058/dm3058.exe",
    "tasks": 1,
    "configurable": [
        {
            "key": "POWERID",
            "name": "电源ID",
            "value": "USB0::0x1AB1::0x09C4::DM3R212200956::0::INSTR"
        },
        {
            "key": "TIME",
            "name": "时间",
            "value": "2"
        }
    ],
    "cases": [
        {
            "ordernum": 6,
            "name": "POWER",
            "steps": [
                {
                    "seq": 1,
                    "deviceId": "CURRENT",
                    "name": "测试",
                    "cmd": "$POWERID$ $TIME$",
                    "timeout": 4000,
                    "result": "SUCCESS"
                }
            ]
        }
    ]
}
```

需要注意的是，在实际使用中通常会在执行该项前已通过烧录命令或deviceId取得板子唯一标识，这时需要将27行内容修改为：

```
"extra": ["CURRENT"]
```

7.2.3 电流测量程序

```
#!/usr/bin/env python
```

```
# -*- coding:utf-8 -*-

'''Current Measurement for DM3058'''

import time
import pyvisa
import sys

# format json output
def format_output(result, reason=None, extitle=None, extra=None):
    '''format_output with (result, reason, extitle, extra)'''
    message = '{'
    message += '"result":"' + str(result.encode('unicode_escape').decode('utf-
8')) + '"'
    if reason is not None: # Fail case: add reason
        message += ',"reason":"' +
str(reason.encode('unicode_escape').decode('utf-8')) + '"'
    if extra is not None:
        message += ',"extra":[{"' +
str(extitle.encode('unicode_escape').decode('utf-8')) +'":"'
        message += str(extra.encode('unicode_escape').decode('utf-8')) + '"}]'
    message += '}'
    print(message)

def serial_and_get_current(idset, powerdelay, powrecom = 0):
    calc = 0.0
    if idset == '':
        return calc
    if powerdelay in ('', '0'):
        print('not set power delay')
        powerdelay = 1000
    try:
        resource = pyvisa.ResourceManager()
        inst = resource.open_resource(str(idset))

        inst.write("cmdset rigol")
        inst.write(":function:current:DC")
        inst.write(":MEASure:CURRent:DC 2")
        inst.write(":MEASure:CURRent:DC:RANGe?")

        time.sleep(0.1)
        ret = inst.read()
        if int(ret) != 2:
            print('Err:Range Not 0-20ma ,' + (ret))

        inst.write(":measure AUTO")
        inst.write(":calculate:function AVERAGE ")
        time.sleep(float(powerdelay) / 1000.0)
        inst.write(":calculate:statistic:average?")
```

```
        get_current = float(inst.read())

        inst.write(":calculate:statistic:count?")

        times = inst.read()
        if times == 0:
            print('err times:' + str(times))

        inst.write(":calculate:function none")
        inst.write("*cls")

        resource.close()
        caldata = float(get_current * 1000000.0 - float(powrecom))
        #print('read current is :' + str(caldata) + 'ua')
        calc = (str(caldata))

    except Exception as ex:
        print(str(ex))
        calc = 0.0
        return False, calc
    return True, calc

def getcurrent(setid, testtime):
    result, getpower=serial_and_get_current(setid, testtime)
 #'USB0::0x1AB1::0x09C4::DM3R212200956::0::INSTR', '1000'

    if result is True:
        format_output("SUCCESS", None, "CURRENT", str(getpower) + ' ua')
        return True
    else:
        format_output("FAIL", "GET CURRENT ERROR")
        return False

if __name__ == '__main__' :
    print(str(sys.argv))
    try:
        setid = sys.argv[5]
        testtime = sys.argv[6]

        getcurrent(setid, testtime)

    except Exception as ex:
        print(str(ex))
        format_output("FAIL","parameter error:",None)
```