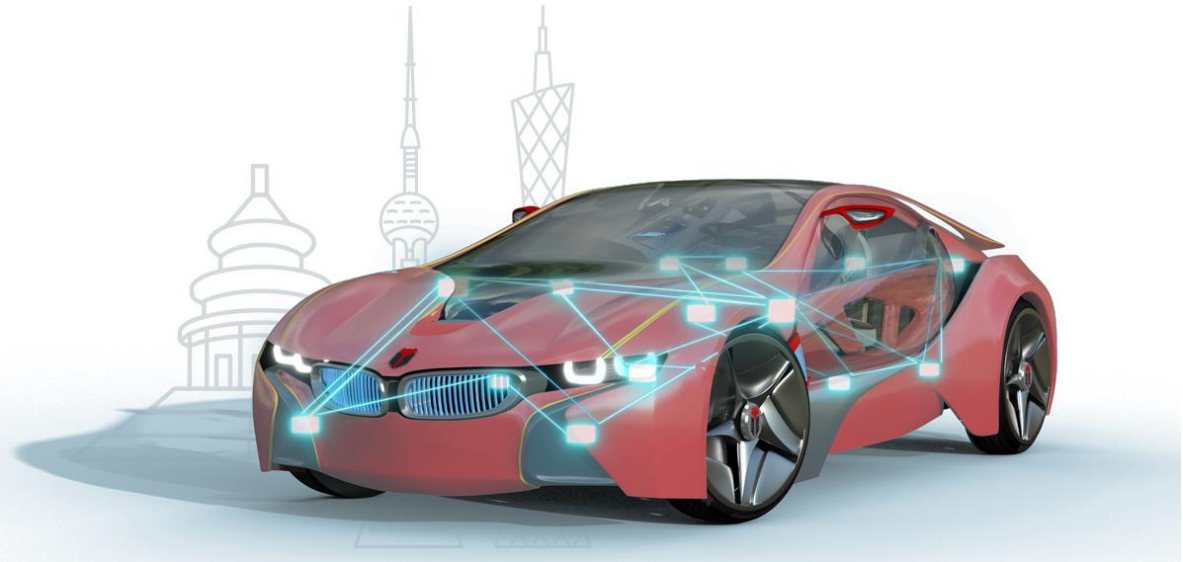
知从嵌入式自动化测试服务手册
ZC EMBEDDED AUTOMATION TESTING SERVICE MANUAL

知从工程服务

ZC Engineering Services



知从嵌入式自动化测试服务手册

ZC EMBEDDED AUTOMATION TESTING SERVICE MANUAL

知从工程服务

ZC Engineering Services

1 概述 OVERVIEW

随着硬件元器件的飞速发展，嵌入式系统的功能越来越强大和复杂。从早期仅具有简单控制功能的系统，发展到如今广泛应用于各种领域且功能多样化的智能系统，如智能家居、智能汽车、工业自动化等。这使得嵌入式软件的规模和复杂度不断增加，对软件的质量和可靠性要求也越来越高，从而导致嵌入式软件测试工作的难度和工作量大幅上升。

With the rapid development of hardware components, the functions of embedded systems are becoming increasingly powerful and complex. From early systems with only simple control functions, it has developed into intelligent systems that are widely used in various fields and have diverse functions, such as smart homes, smart cars, industrial automation, etc. This has led to an increasing scale and complexity of embedded software, as well as higher requirements for software quality and reliability, resulting in a significant increase in the difficulty and workload of embedded software testing work.

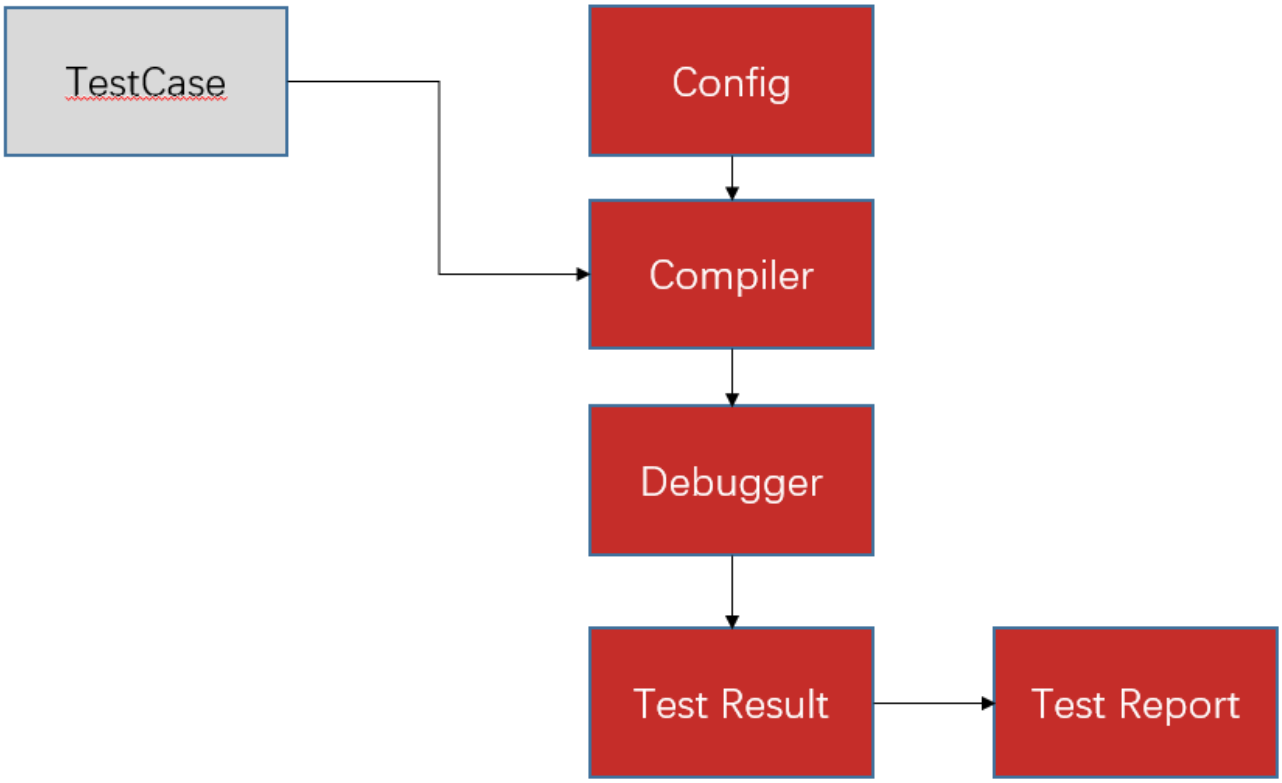
早期的嵌入式系统较为简单，传统的手工测试方法能够在一定程度上满足测试需求。然而，随着嵌入式软件的日益复杂，手工测试的效率低下、容易出错、难以重复执行等问题逐渐凸显，已无法对大规模的嵌入式软件进行全面、准确、高效的测试，无法满足嵌入式软件快速迭代和高质量发展的需求。

Early embedded systems were relatively simple, and traditional manual testing methods were able to meet testing requirements to a certain extent. However, with the increasing complexity of embedded software, problems such as low efficiency, easy errors, and difficulty in repeated execution of manual testing have gradually become prominent. It is no longer possible to conduct comprehensive, accurate, and efficient testing of large-scale embedded software, and cannot meet the needs of rapid iteration and high-quality development of embedded software.

针对以上问题，知从提供完整的嵌入式自动化测试服务，可有效提高客户测试效率，降低测试成本，提高产品质量。

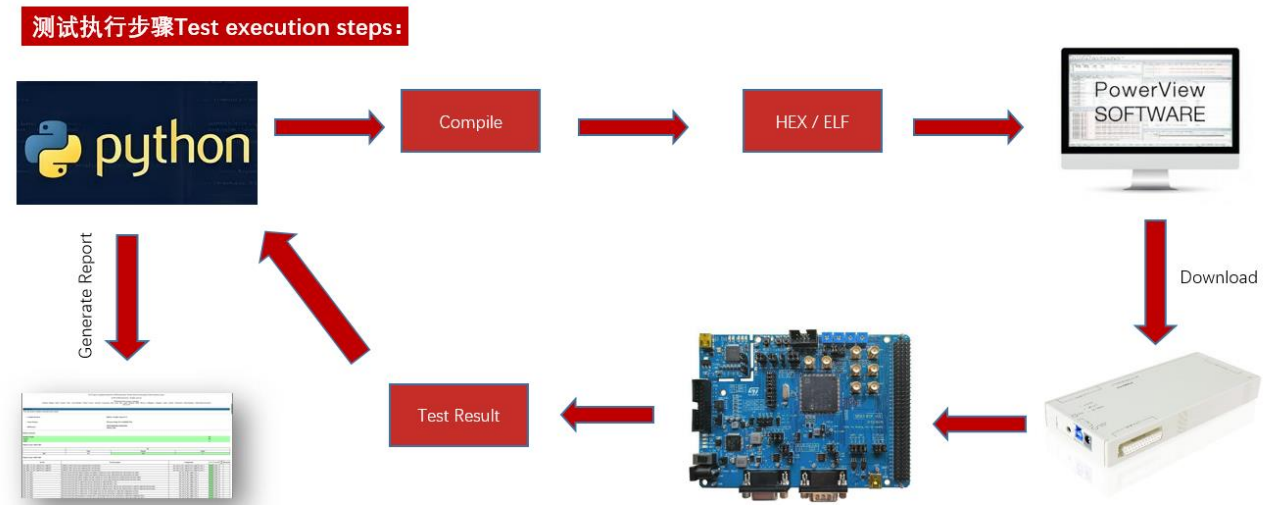
In response to the above issues, ZC provides complete embedded automation testing services, which can effectively improve customer testing efficiency, reduce testing costs, and improve product quality.

2 自动化测试方案 AUTOMATED TESTING PLAN



测试流程

Test Process



测试流程

Test Process

Config:不同的配置文件

Config:Different Configuration Files

Compiler:编译

Compiler: Compile

Debugger:烧录调试

Debugger: Burn debugging

TestCase:测试代码

TestCase: Test Code

TestResult:测试结果

TestResult: Test Result

TestReport:测试报告

TestReport: Test Report

3 测试流程 TEST PROCESS

1. 通过配置工具生成多组配置文件 Config(如木牛工具，EB 工具)

Generate multiple sets of configuration files Config through configuration tools (such as MuNiu tool, EB tool).

多组 Config

Multiple sets of Config

2. 增加测试代码到编译路径中

Add test code to the compilation path

方式 1：通过 uart 串口打印的形式进行测试

Method 1: Test through UART serial port printing

```
[ 8] cmocka_driver_gpio: [ RUN      ] drivertest_gpio
[input and output test]  outvalue is 0, invalue is 0
[input and output test]  outvalue is 0, invalue is 0
[input and output test]  outvalue is 0, invalue is 0
[input and output test]  outvalue is 1, invalue is 1
[input and output test]  outvalue is 0, invalue is 0
[input and output test]  outvalue is 0, invalue is 0
[input and output test]  outvalue is 0, invalue is 0
[input and output test]  outvalue is 0, invalue is 0
[input and output test]  outvalue is 1, invalue is 1
[input and output test]  outvalue is 1, invalue is 1
[ 8] cmocka_driver_gpio: [       OK ] drivertest_gpio
[ 8] cmocka_driver_gpio: [==========] tests: 1 test(s) run.
[ 8] cmocka_driver_gpio: [  PASSED  ] 1 test(s).
```

Uart 打印 log

Uart prints logs

方式 2：通过读取测试代码中的测试结果变量进行测试

Method 2: Test by reading the test result variables in the test code

```
EU_TEST_SUITE_BEGIN(MCU_TS_GetResetRawValue_001)
    EU_TEST_CASE_ADD(Mcu_TC_GetResetRawValue_0001, "Call Mcu_GetResetRawValue and check applied configuration"),
EU_TEST_SUITE_END(MCU_TS_GetResetRawValue_001)

EU_TEST_REGISTRY_BEGIN()
    EU_TEST_SUITE_ADD(MCU_TS_GetResetRawValue_001, "Mcu_GetResetRawValue validation")
EU_TEST_REGISTRY_END()
```

嵌入式测试代码

Embedded testing code

3. 通过 Makefile 命令行形式自动化编译多组 Config,生成 elf 文件

可支持多种编译器，并且可使用命令行形式操作，自动化更加便捷

Automatically compile multiple sets of Config through the Makefile command line and generate elf files

Supports multiple compilers and can be operated in command-line format, making automation more convenient

Makefile

D: > ST_MakeFile > Workspace > Makefile

```
83
84  all: build_elf
85
86  build_elf: DEBUG_SHOW $(TARGET).elf
87
88  DEBUG_SHOW:
89      @echo "########################################"
90      @echo "Files to compile: $(C_SOURCES)"
91      @echo "########################################"
92
93  # Link all the object files to become one elf file
94  $(TARGET).elf : $(OUT_FILES) $(LINKER_DEF)
95      @echo "Linking $@"
96      @$(LD) $(LDFLAGS) $(OUT_FILES) -o $@
97      @echo ""
98      @echo "########################################"
99      @echo "All Done! $@"
100     @echo "########################################"
101
102 # Compile all source c files
103 vpath %.c $(addsuffix :, $(SRC_DIRS))
104 $(ODIR)/%.o: %.c $(DEPDIR)
105     @echo "Compiling $<"
106     @$(CC) $(CCOPT) $(INCLUDE_DIRS) -c $< -o $@
107
108 # Compile all source s files
109 vpath %.s $(addsuffix :, $(SRC_DIRS))
110 vpath %.S $(addsuffix :, $(SRC_DIRS))
111 $(ODIR)/%.o : %.s
112     @echo "Compiling $<"
113     @$(AS) $(ASFLAGS) $< -o $@
114 $(ODIR)/%.o : %.S
115     @echo "Compiling $<"
116     @$(AS) $(ASFLAGS) $< -o $@
117
118 $(DEPDIR):
119     @mkdir -p $@
120
121 ##########################################################################################################################
```

Makefile 编译环境

Makefile compilation environment

4. 通过 python 调用调试器进行烧录，运行，测试

Using Python to call the debugger for burning, running, and testing

```
remote PING received
remote PING received
printout sent to: file D:\User\Bart.xu\ST\T32\EBtresos_Cfg\icu\Icu_INT_TS_01_Unit_E1_cfgPC\generate\epc\log.txt (ASCIIE)
remote PING received
INFO: SW-DP DPv3 enabled; SwitchToSwd=TryAll->None; TARGETSEL=0xffffffff; DPIDR=0x4c013477
Dualport flash programming mode enabled
File 'D:\Git\ARSTM01\ZC135b1bd5_02_SR5E1E7\SR5E1_GPT\debug\SR5E1_GPT_Demo.elf' (ELF/DWARF4) loaded.
INFO: SW-DP DPv3 enabled; SwitchToSwd=TryAll->None; TARGETSEL=0xffffffff; DPIDR=0x4c013477
```

劳特巴赫 log

Lauterbach log

5. 通过 python 调用调试器得到结果

Call the debugger in Python to obtain the result

```
B::var.watch_TestRet=
  TestRet= ((pass = 0x0, resultBasePtr = 0x24020080, testRegistryPtr = 0x0800D1DC), (pass = 0x0, resultBasePtr = 0x0, testRegistryPtr = 0x0))
```

劳特巴赫输出的结果

Lauterbach's output result

6. 根据结果自动化生成测试报告

Automatically generate test reports based on the results

可以定制化生成测试报告

Can customize the generation of test reports

4 测试设备说明 TESTING EQUIPMENT DESCRIPTION

4.1 硬件需求 Hardware Requirements

测试设备 Test equipment			
序号 No.	设备种类 Device Type	设备名称 Device Name	设备要求 Device requirements
No.1	电源 power supply	程控电源 Programmable power supply	至少 2 路， 0 – 32V，分辨率，10mv/1mA，精度 0.2% At least 2 channels, 0-32V, resolution, 10mv/1mA, accuracy 0.2%
No.2	通信设备 communication devices	NA	NA
No.3	调试器 debugger	NA	NA

通信设备：在不同的测试环境中需要不同的通信设备，如测试 can 通信，则需要 can 相关通信设备，主要需要 python 能够调用设备的 API 进行操作

Communication devices: Different communication devices are required in different testing environments. For example, when testing CAN communication, CAN related communication devices are needed, mainly requiring Python to be able to call the device's API for operation

调试器：需要 python 能够调用设备的 API

Debugger: requires Python to be able to call the device's API