DT10 如何帮助用户有效达成灰盒测试目标

前言：

在 1999 年，美国洛克希德马丁公司发表了灰盒测试的论文，提出灰盒测试方法，是一种介于白盒测试和黑盒测试的一种新的测试方法。2000 年洛克希德马丁公司在之前灰盒测试基础上，完整论述在真实环境中，以实时方式对嵌入式设备进行灰盒测试的方法（*《Graybox Software Testing in the Real World in Real-Time》*），进一步完善了灰盒测试理论，不单从覆盖角度验证软件功能正确性和测试完整性，同时从时间维度测试嵌入式设备的性能指标是否满足实时系统性能需求。

黑盒、白盒、灰盒测试比较：

我们都知道黑盒测试一般根据系统的需求文档、设计文档，来设计测试用例，从系统的角度验证功能是否能否满足需求。

黑盒测试的优点：

由于其面向系统级别，不涉及代码，测试人员只需了解需求文档，根据系统功能描述设计测试用例，执行测试用例，简单易行，基本上只需要对测试有一定了解，对产品功能需求由较深的理解，即可实施黑盒测试。

黑盒测试的缺点：

由于只关注系统外部输入输出，而不会深入到系统内部，因此测试不够深入，容易造成测试遗漏，同时发现功能问题后，难以定位问题原因。

白盒测试也即单元测试，一般由开发人员自己针对所写代码进行测试，其测试对象一般函数单元或者文件单元。

白盒测试的优点：

针对每个函数进行测试，测试粒度足够细，测试足够充分，通过单元测试用例的回归，发现问

题时，很容易定位到具体的出错函数，同时测试用例一旦建立完成，可以通过命令行的方式自动化的实施回归测试。

白盒测试的缺点：

对测试人员要求较高，需要具备较强的阅读、编写代码能力；测试粒度细，因此需要花费大量的测试用例编写以及维护时间；测试对象是单元级别，主要验证函数的功能逻辑是否正确，没法覆盖系统级的验证，即使做了单元测试，仍需进行系统测试；同样由于其面向函数级别，单元测试也没办法实施性能测试。

上面我们梳理了白盒测试、黑盒测试的优缺点，那么灰盒测试是否能够兼顾白盒、黑盒测试的优点，尽量避免白盒和黑盒的缺点，取长补短呢？

灰盒测试：

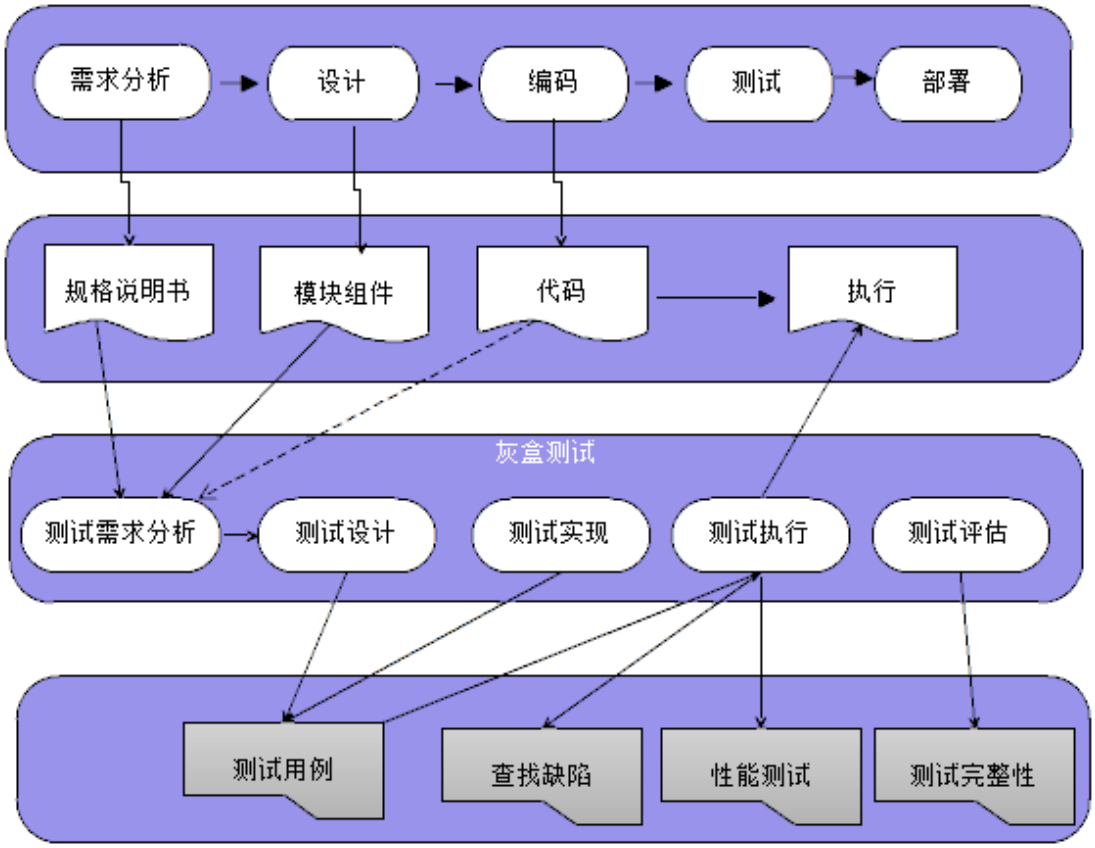
从验证系统功能正确性的角度，以常规黑盒测试方式，同时结合程序内部逻辑结构设计用例，执行程序并采集程序执行路径信息和外部输入输出结果。

灰盒测试的优点：

灰盒测试对程序内部逻辑的关注不像白盒测试那样细致，是兼顾测试效果和效率的测试方法；能够进行基于需求的覆盖测试和基于程序路径覆盖的测试；测试结果可以对应到程序内部路径，便于 bug 的定位、分析和解决；能够保证设计的黑盒测试用例的完整性，防止遗漏软件的一些不常用的功能或功能组合； 能够分析需求或设计不详细或不完整对测试造成的影响；由于面向系统级测试，能实施性能测试；当然灰盒测试也有其自身面临的挑战，文章后面部分会专门论述。

灰盒测试流程概述：

灰盒测试主张测试人员早期介入测试，无需等待开发人员代码开发成型后，才介入测试。开发团队、测试团队相互协作，共同推进项目。下图描述了整个软件开发生命周期的各个阶段中灰盒测试的框架图：



这里重点对测试用例设计以及灰盒测试过程中的挑战加以探讨：

测试用例设计：

灰盒测试的测试用例设计，除了来自于需求规格说明书、设计说明书外，同时来自于对代码结构化的了解。

首先，开发团队根据客户需求，进行项目需求分析，并制定需求规格说明书，此时测试团队可同步参与需求分析工作，根据需求规格说明书，深入理解并审核需求，一方面对需求文档中不完善的地方加以改进，另外一方面进行测试需求分析，设计基于需求的测试用例。

然后，开发团队根据需求分析，从系统架构层面进行系统设计，将整个系统分模块设计，定义各模块接口；此时测试团队需了解和研究系统各模块架构，关系，各个接口定义输入、输出。设计基于模块的测试用例；

其后，开发团队进行编码，测试团队了解和浅阅读代码（*注： 浅阅读，即重点关注代码中重要常量，宏定义，核心函数等，无需对代码完整深入分析*），提取功能实现用到的主要常量、变量，

挖出边界值，对照这些边界的功能，设计测试用例，然后在集成环境中进行功能测试。这样通过代码浅阅读的方式，进一步完善黑盒测试用例。因为有些边界值，开发人员在编码过程中会通过宏定义的方式在头文件中界定，只有通过代码走读的方式，才能更好的确定边界值范围。

这样经过这一系列的工作，测试团队积累了整个系统所需的测试用例。待系统成型后，就进入到具体执行测试用例阶段。

灰盒测试挑战：

上述我们梳理了灰盒测试的流程，可以发现对于灰盒测试而言，测试用例的设计与传统黑盒测试相比较，多出的工作部分主要在于：需从设计文档入手了解模块设计，接口输入输出，同时结合代码浅阅读方式，设计更为完善的测试用例。从这个层面讲，灰盒测试似乎与黑盒测试区别不大，但灰盒与黑盒相较，更大的差异在于测试用例执行和评估等方面。根据洛克希德马丁公司对于灰盒测试的论述，就实践层面，灰盒测试面临如下挑战：

- 需要有判断测试覆盖的能力，评判测试的充分性

如何验证测试是否充分？如何评判测试用例是否存在遗漏？测试覆盖率是软件业界公认的评判标准。当测试人员在设计测试用例时，基于需求文档，设计文档，代码，可以通过需求与测试用例的矩阵图，来判断基于需求的测试覆盖，同时借助一些代码覆盖的工具，比如语句覆盖率，分支覆盖率等验证代码覆盖情况，通过代码覆盖率亦可评判代码是否有冗余？需求设计是否完整有效；

- 灰盒测试需要有记录程序执行的详细日志信息，便于定位分析程序问题

灰盒测试既关注系统外部表现，也关注软件内部执行情况，如何通过有效手段，使得软件执行时外部表现与内部代码相结合，深层次剖析代码执行情况，对于定位和分析问题非常有帮助

- 灰盒测试包含实时系统的性能测试，这对于实时嵌入式系统而言尤其重要，因为功能的输出正确仅仅是逻辑层面正确，还需要在时间维度上满足性能需求

2000 年洛克希德马丁公司在之前灰盒测试基础上，提出系统层级的灰盒测试加入时间维度的

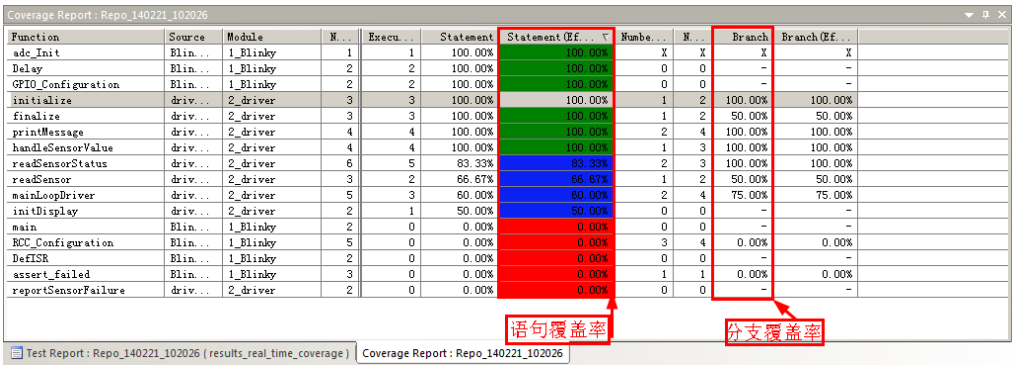
测量，也即对系统性能进行评估和测试，这对于灰盒测试的提出了新的要求。实践角度出发，对于嵌入式系统而言，性能评估和测试也是必不可少的。

DT10 如何帮助用户更有效的实施灰盒测试：

DT10 是动态测试和调试工具，可以长时间记录程序执行状态，其最重要的三大功能：错误定位，覆盖率统计，性能测试，另外还有诸如变量跟踪，软硬件同步示波器等功能，能够很好的帮助用户达成灰盒测试的各项要求：

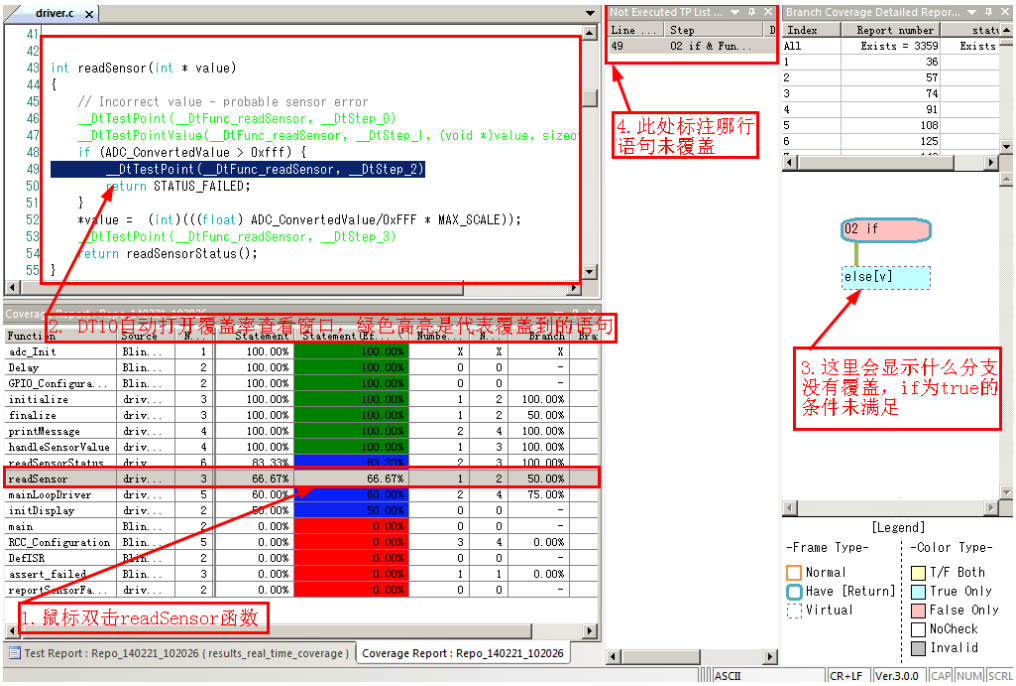
1. DT10 帮助用户获取程序运行时覆盖率，包括语句覆盖和分支覆盖。

通过 DT10 的覆盖率统计功能，在测试人员执行测试用例之后，可以统计相关功能测试之后，代码覆盖率情况。如下图：



Function	Source	Module	N...	Execu...	Statement	Statement (Ef...	Numbe...	N...	Branch	Branch (Ef...
adc_Init	Blin...	1_Blinky	1	1	100.00%	100.00%	X	X	X	X
Delay	Blin...	1_Blinky	2	2	100.00%	100.00%	0	0	-	-
GPIO_Configuration	Blin...	1_Blinky	2	2	100.00%	100.00%	0	0	-	-
initialize	driv...	2_driver	3	3	100.00%	100.00%	1	2	100.00%	100.00%
finalize	driv...	2_driver	3	3	100.00%	100.00%	1	2	50.00%	50.00%
printMessage	driv...	2_driver	4	4	100.00%	100.00%	2	4	100.00%	100.00%
handleSensorValue	driv...	2_driver	4	4	100.00%	100.00%	1	3	100.00%	100.00%
readSensorStatus	driv...	2_driver	6	5	83.33%	83.33%	2	3	100.00%	100.00%
readSensor	driv...	2_driver	3	2	66.67%	66.67%	1	2	50.00%	50.00%
mainLoopDriver	driv...	2_driver	5	3	60.00%	60.00%	2	4	75.00%	75.00%
initDisplay	driv...	2_driver	2	1	50.00%	50.00%	0	0	-	-
main	Blin...	1_Blinky	2	0	0.00%	0.00%	0	0	-	-
RCC_Configuration	Blin...	1_Blinky	5	0	0.00%	0.00%	3	4	0.00%	0.00%
DefISR	Blin...	1_Blinky	2	0	0.00%	0.00%	0	0	-	-
assert_failed	Blin...	1_Blinky	3	0	0.00%	0.00%	1	1	0.00%	0.00%
reportSensorFailure	driv...	2_driver	2	0	0.00%	0.00%	0	0	-	-

当用户希望详细了解某个函数覆盖情况时，可以双击某个函数，DT10 将自动打开函数代码，并在函数代码中详细标识出哪行语句覆盖，哪行语句未覆盖，以及什么分支覆盖了，什么分支未覆盖。如图：

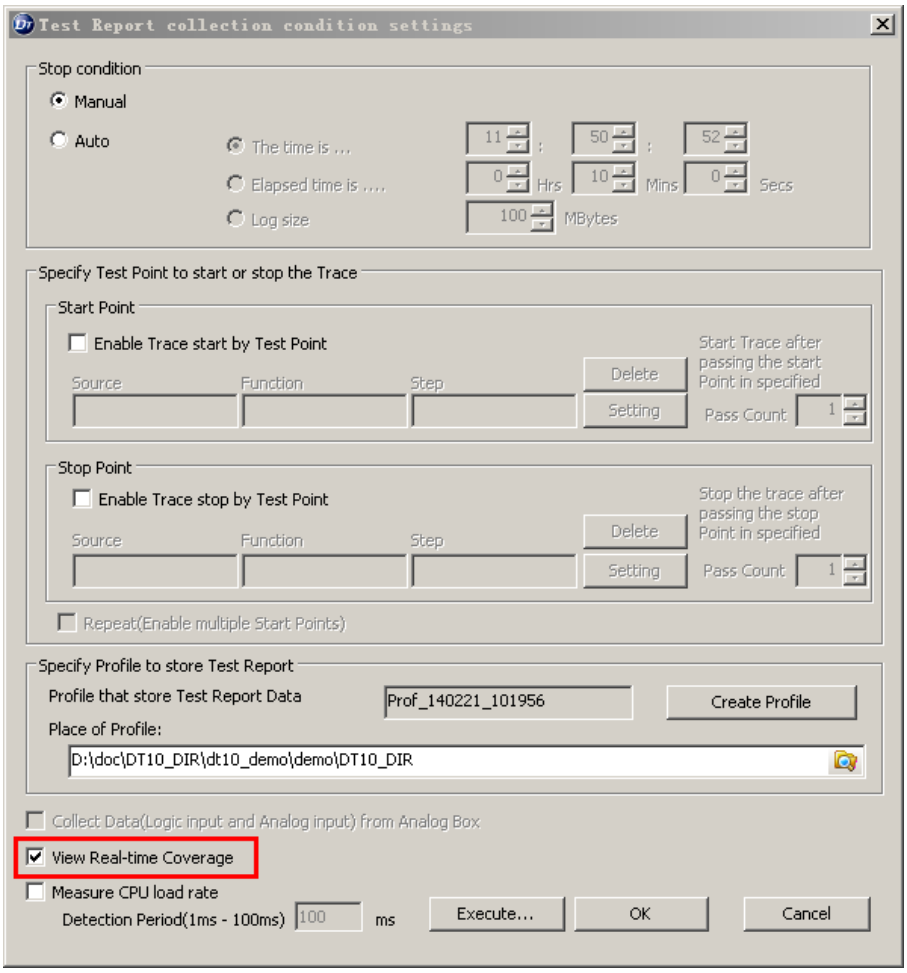


通过 DT10 的语句覆盖和分支覆盖 ,可以帮助用户评估灰盒测试过程中测试用例是否存在遗漏？从代码的角度，评估哪些代码覆盖，哪些代码未覆盖，从而判断代码是否存在冗余的情况。

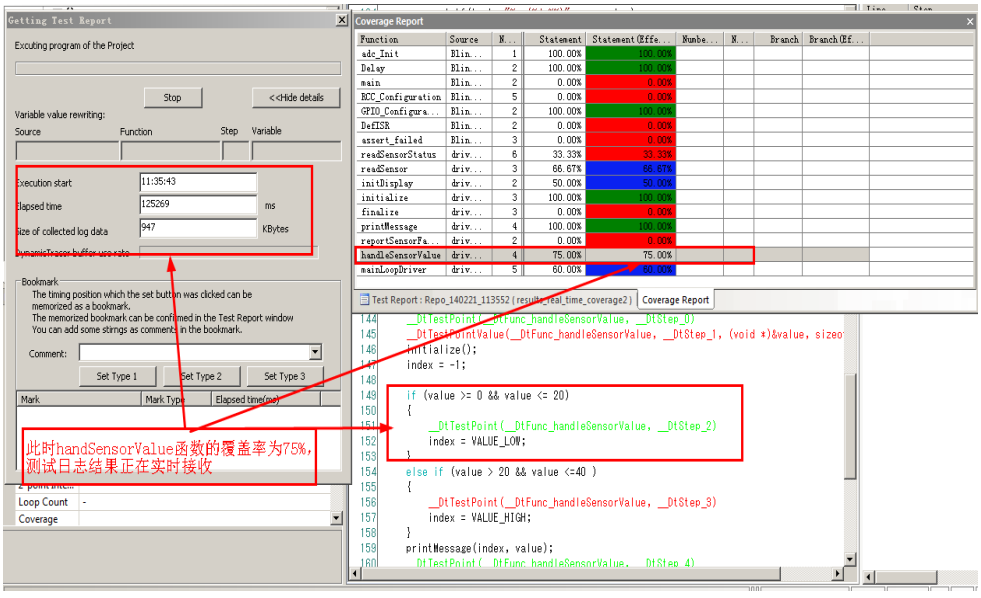
2. DT10 实时覆盖率（Real Time Coverage）

上面看到的覆盖率的获取，一般是 DT10 跟踪目标版执行，测试数据收集完成之后，然后通过 DT10 的分析功能分析覆盖率情况。另外，DT10 还提供 real time coverage，也即实时覆盖率。通过实时覆盖率，用户可以实时的看到覆盖率情况，比如你在目标版上操作某个按钮，从而触发某行代码，此时在目标板执行过程中，即可在 DT10 的窗口中实时的看到覆盖率数据。

首先在测试报告收集窗口中，启用“View Real-time Coverage”选项：

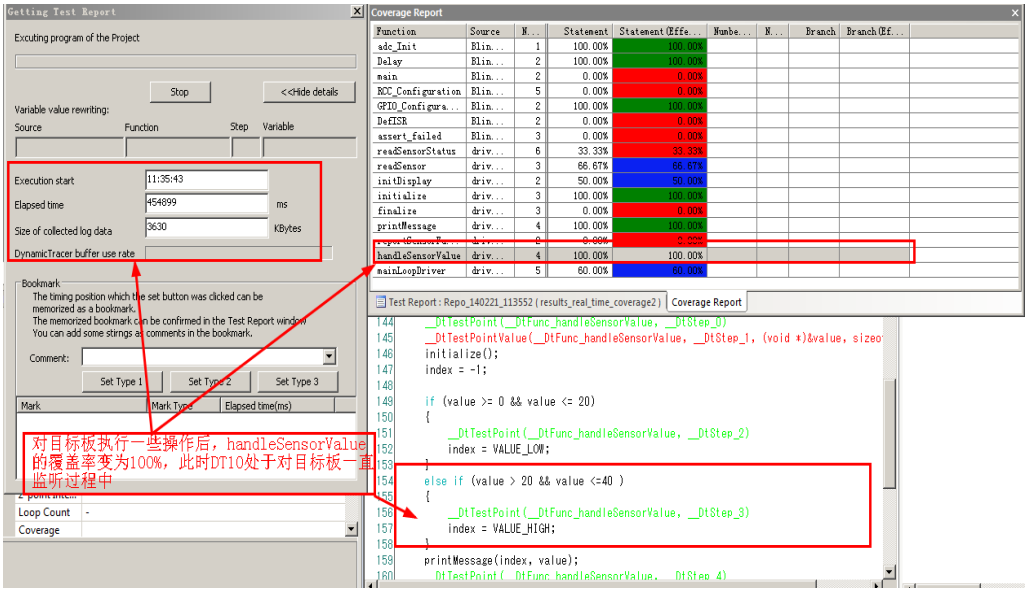


然后启动插入测试点后的目标板程序，并在 DT10 中实时监听测试结果数据，此时 DT10 可以实时的获取覆盖率数据，如下图：



然后我操作目标板上的按钮，使程序运行到另外一个分支，注意此时 DT10 一直在实时接收目

标板执行的测试数据，得到结果如下图：



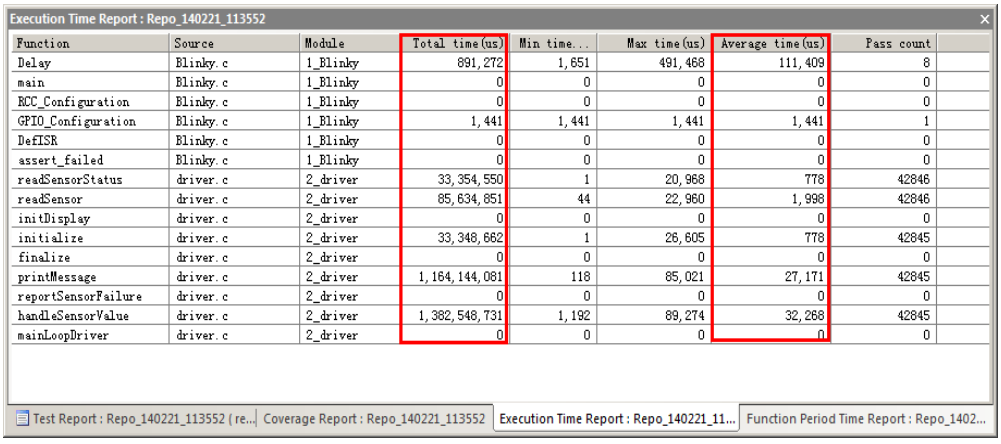
实时覆盖率，使得用户在硬件上操作后，在软件的角度实时看到代码执行和覆盖情况，这也有助于用户掌握目标系统实时执行过程中软件执行情况的了解。

3. DT10 帮助用户进行性能评估和测试

DT10 可监测每个函数的执行时间和周期时间，也可监测系统中任意两行代码之间的执行时间以及周期时间。对于多任务的系统而言，DT10 还可监测 CPU 压力。

函数执行时间和周期时间：

通过 DT10 对目标系统长时间跟踪测试，可以得到每个函数的执行时间，如下图：

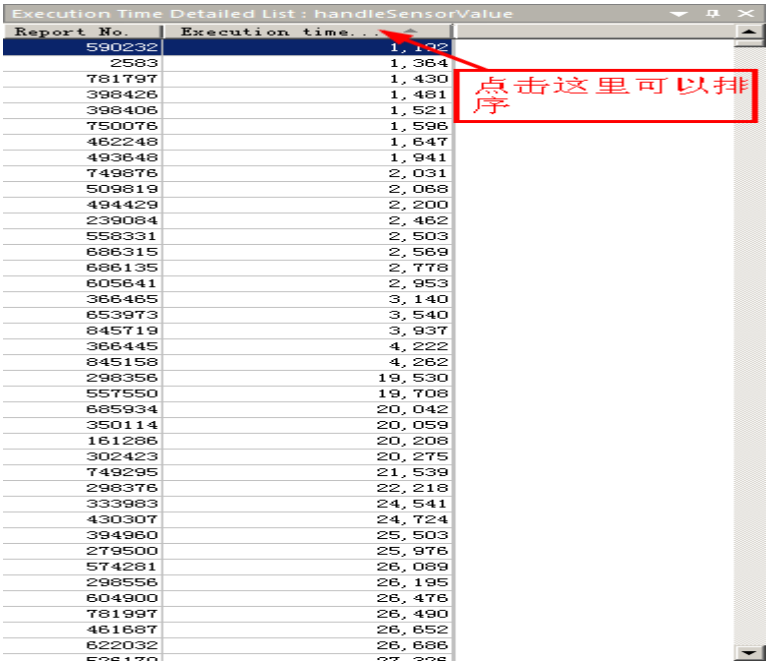


Execution Time Report : Repo_140221_113552

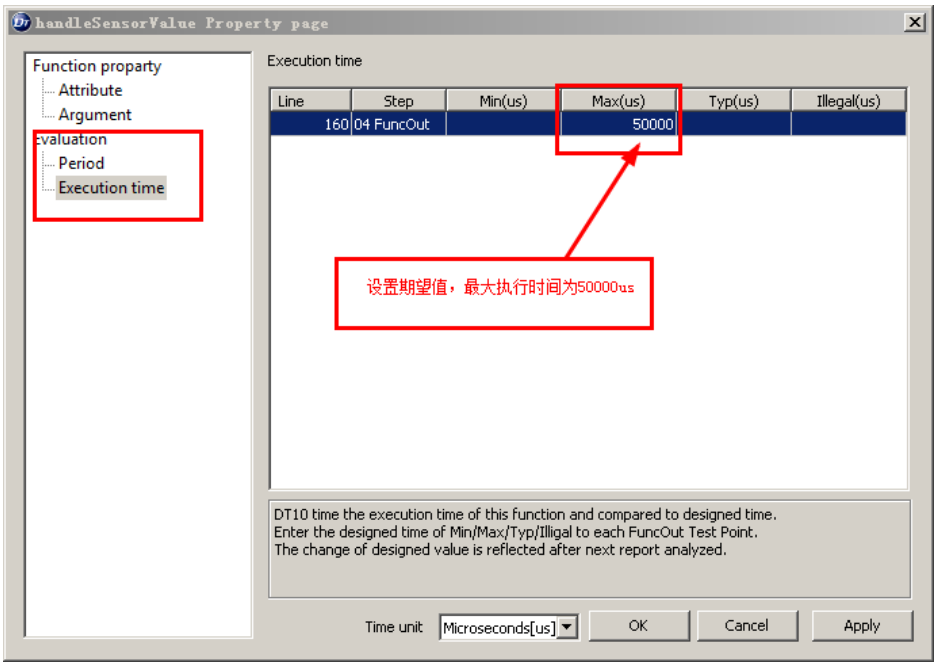
Function	Source	Module	Total time(us)	Min time...	Max time(us)	Average time(us)	Pass count
Delay	Blinky.c	1_Blinky	891,272	1,651	491,468	111,409	8
main	Blinky.c	1_Blinky	0	0	0	0	0
RCC_Configuration	Blinky.c	1_Blinky	0	0	0	0	0
GPIO_Configuration	Blinky.c	1_Blinky	1,441	1,441	1,441	1,441	1
DefISR	Blinky.c	1_Blinky	0	0	0	0	0
assert_failed	Blinky.c	1_Blinky	0	0	0	0	0
readSensorStatus	driver.c	2_driver	33,354,550	1	20,968	778	42846
readSensor	driver.c	2_driver	85,634,851	44	22,960	1,998	42846
initDisplay	driver.c	2_driver	0	0	0	0	0
initialize	driver.c	2_driver	33,348,662	1	26,605	778	42845
finalize	driver.c	2_driver	0	0	0	0	0
printMessage	driver.c	2_driver	1,164,144,081	118	85,021	27,171	42845
reportSensorFailure	driver.c	2_driver	0	0	0	0	0
handleSensorValue	driver.c	2_driver	1,382,548,731	1,192	89,274	32,268	42845
mainLoopDriver	driver.c	2_driver	0	0	0	0	0

如果想查看某个函数具体的函数执行时间情况，比如函数 handleSensorValue 函数，被执行

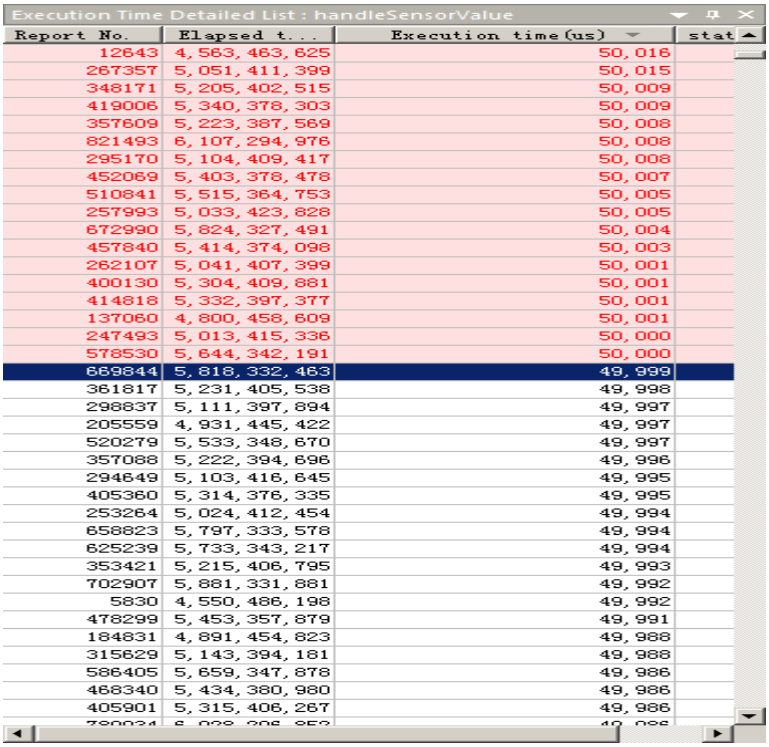
42845 次，通过 DT10 还可以看到该函数每次执行时间，只要双击 handleSensorValue 函数即可，弹出如下窗口：



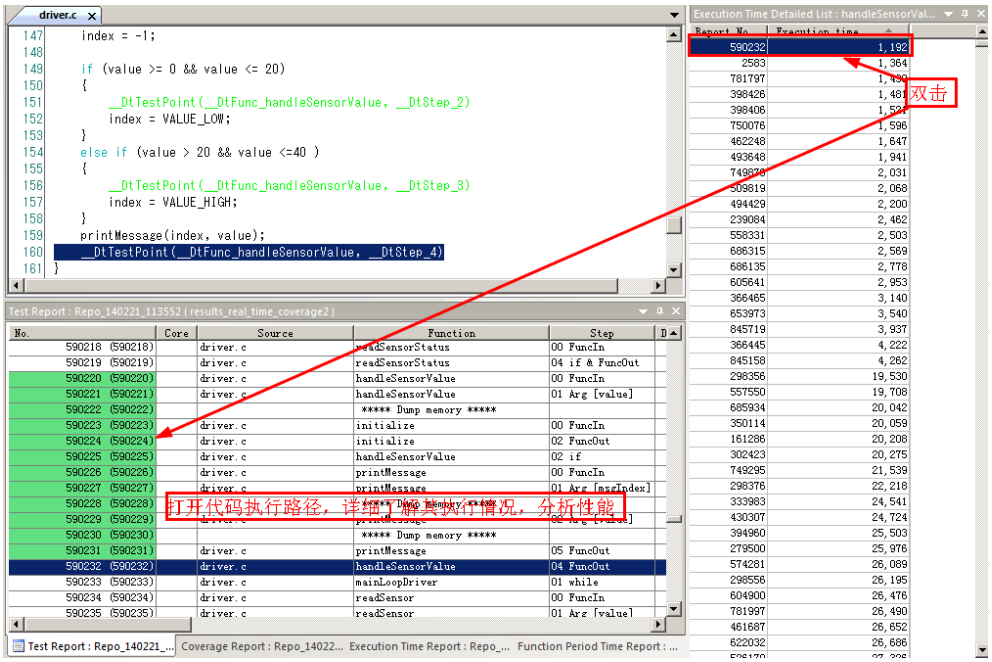
上图的列表中将 handleSensorValue 函数执行时间全部罗列出来，在 DT10 中可以设置某个函数的执行时间的标准值，比如该函数错逻辑上执行时间不能超过 50000us，在 DT10 中我们可以设置：



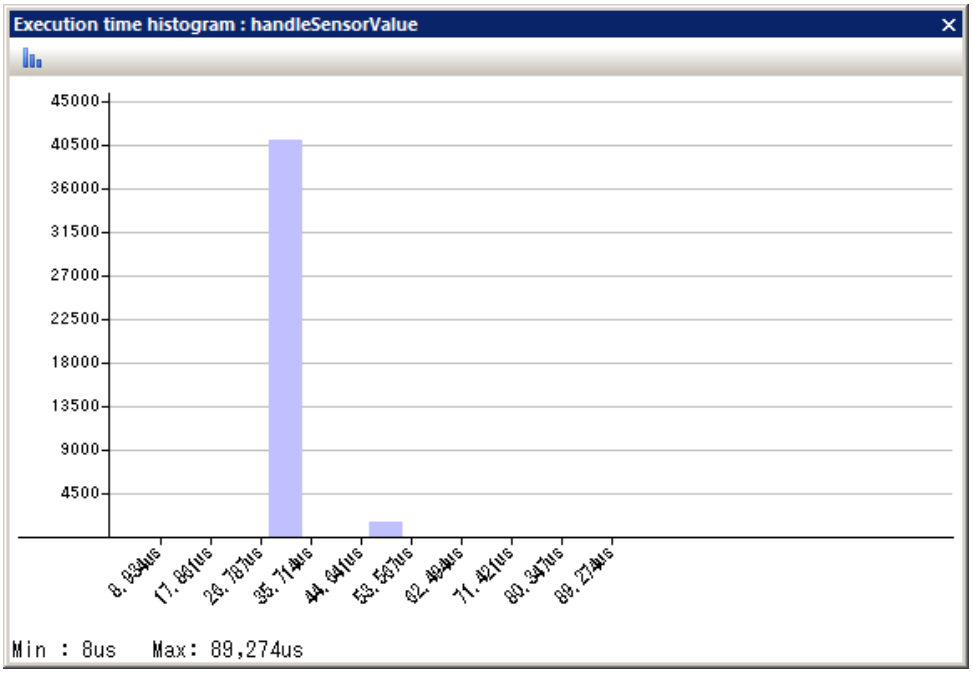
分析结构后，DT10 会将执行时间不符合预期的值全部用红色高亮显示：



当双击某次执行时间，DT10 会同步显示其执行代码路径情况，如下图：



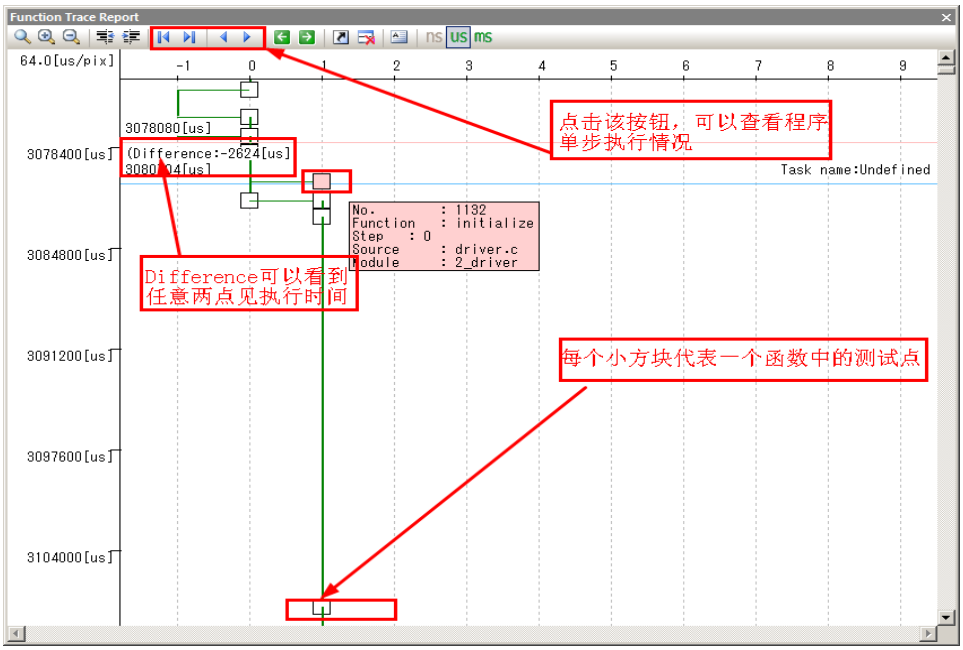
除此之外，DT10还可以看到函数执行时间统计图，如下图：横轴代表执行时间，纵轴代表执行次数，从下面的柱状图可以知道，执行时间为 26787us~35714us 区间的次数到达 40500 次，而在 44641us~53567us 区间的执行次数很少。



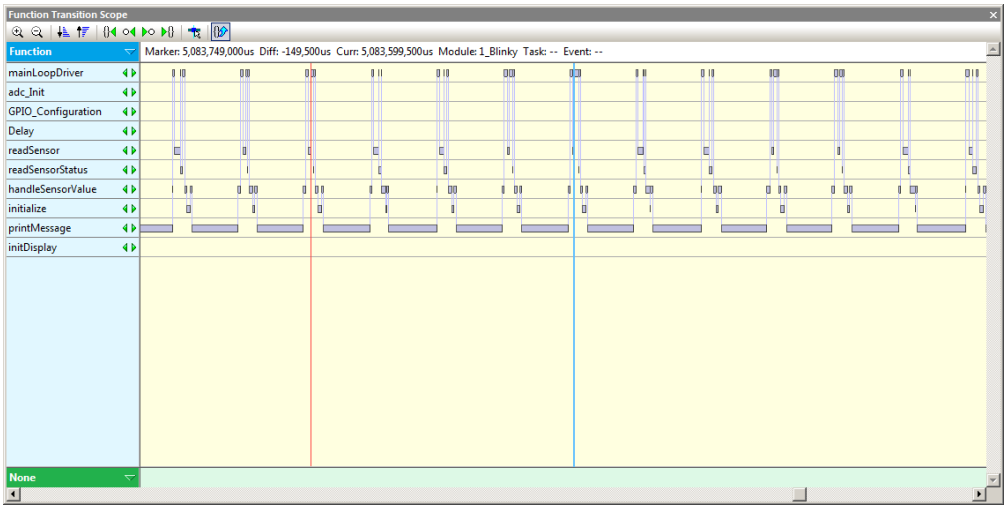
该统计图的意义，一方面我们可以了解某个函数执行时间主要区间，同时如果发现某低频率的时间出现，比如出现一次执行时间在 8934us，那么作为性能分析的人，我们需要重点分析，因为执行时间非常短，并且在系统长时间执行的情况下，该执行之间只出现一次，那么极有可能这一次的执行逻辑存在问题，因为它与多数执行时间差异太大，那么此时可以通过之前的双击本次执行情况，弹出其代码执行逻辑情况进行详细分析。

4. DT10 的 Function Trace Report 和 Function Transition Scope 通过可视化的方式帮助用户理解代码内部执行逻辑

下图是通过 DT10 的 Function Trace Report 获取程序执行逻辑的可视化报告，从图中的标注我们可以看到用户可以通过蓝色箭头重现目标板上代码执行逻辑



而下图是 Function Transition Scpoe 报告:

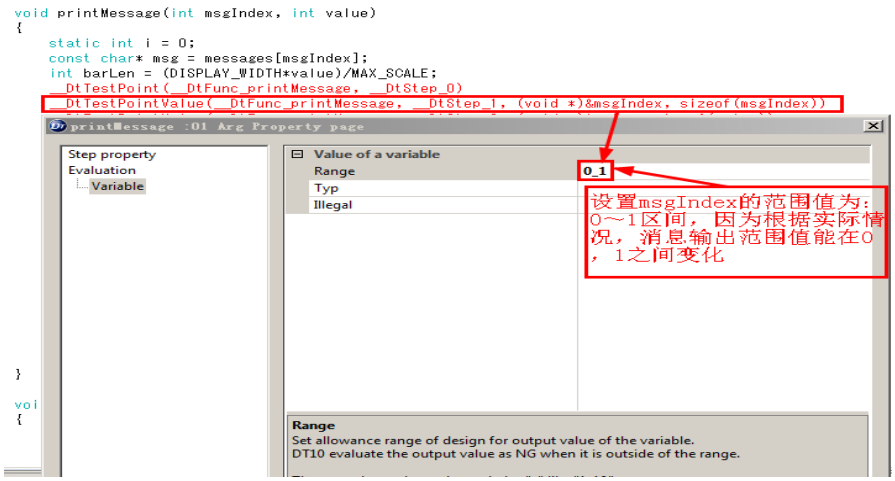


通过该报告，用户可以详细了解在系统执行过程中，各个函数任务跳转情况。

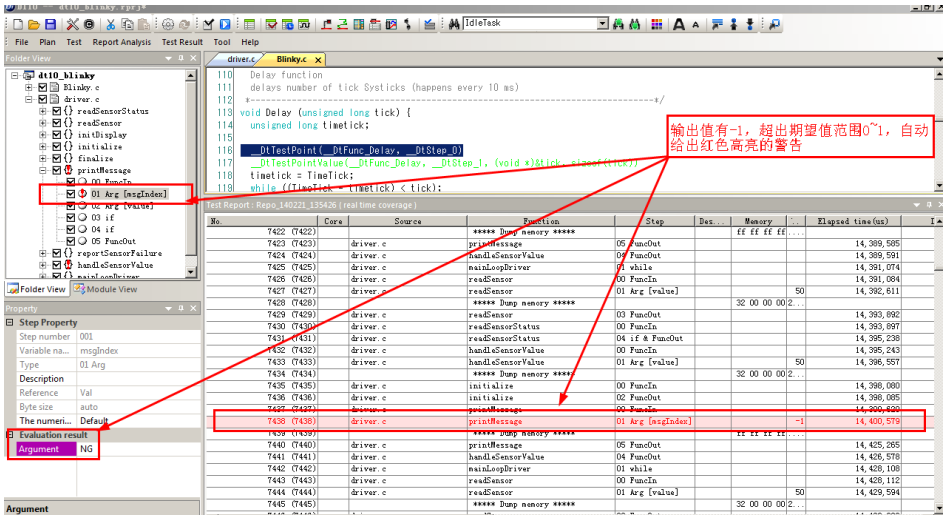
5. 通过 DT10 的 DTplaner 设置目标系统指定接口上的变量、参数的预期值

在灰盒测试的测试用例设计过程中，我们会从模块和代码的角度设计众多的测试用例，在这些测试用例中，涉及输入，输出值，当我们给予系统指定输入值，其响应输出值是否符合我们的预期？传统的无论黑盒还是灰盒测试，需要我们人为的查证输出是否符合预期或者写一些断言代码判断输出是否符合预期，在 DT10 中，我们可以通过 DTPlaner 设置变量，参数，包括函数执行时间，周期时间的期望值属性，当目标系统执行过程中，其实际值与期望值不符时，DT10 将通过红色的高

亮显示或者给出一个红色感叹号，警告此处与期望值不符。这对于自动验证边界值非常有帮助，同时对于后续版本的回归测试也非常有帮助。如下图：



通过 DT10 跟踪目标系统，收集测试数据后，进行分析，得出如下结果：



对于性能要求的执行时间，也可以采用同样的方式，设置执行时间的期望值。

结语：

灰盒测试结合了黑盒测试和白盒测试的优点

- 能够进行基于需求的覆盖测试和基于程序路径覆盖的测试；
- 测试结果可以对应到程序内部路径，便于 bug 的定位、分析和解决；
- 能够保证设计的黑盒测试用例的完整性，防止遗漏软件的一些不常用的功能或功能组合，反之，亦可发现一些 dead code；

- 能够评估需求或设计不详细或不完整对测试造成的影响；
- 方便进行性能评估和测试。由于灰盒测试以 Real Time 的方式对目标系统进行在线测试，并采集程序执行路径信息，关注代码执行情况，因此方便进行性能测试和评估。而采用白盒测试的单元测试方式，只是针对函数级别进行测试，并非系统层面的实时运行，因此无法获取实时性能测试数据；

结合 DT10 提供的覆盖率测试，性能测试，测试执行路径记录等功能，可以有效达成灰盒测试各项要求，提高测试效果和测试效率。