公开

天目全数字实时仿真软件
SkyEye 用户手册

浙江迪捷软件科技有限公司

关键领域·国内领先的 MBSE 工业软件

注意

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目　　录

1 引言

1.1 编写目的

本文档是针对天目全数字实时仿真软件 SkyEye（以下简称 SkyEye）用户所编写的使用手册。通过阅读本文档，SkyEye 使用者可以快速了解并掌握 SkyEye Workstation 软件、SkyEye 自动化测试功能与 SkyEye 命令行功能的使用。

1.2 相关文档

《天目全数字实时仿真软件 SkyEye 安装手册》

《SkyEye 外设建模》

《SkyEye 自动化测试及接口（Python）使用说明手册》

《SkyEye 命令行手册》

《SkyEye 与原生 IDE CCS 远程调试方法》

《SkyEye demo 工程使用说明》

1.3 术语和定义

术语	描述
SkyEye	天目全数字实时仿真软件
License	SkyEye 软件使用许可证
SoC	System on Chip，片上系统
会话	SkyEye 控制运行的最小单元
加载	将工程加载到 SkyEye 中
运行	开始仿真
暂停	暂停仿真
卸载	将工程从 SkyEye 中卸载
工作空间	存储 SkyEye 项目、文件等资源的空间
接口（硬件模型搭建）	虚拟设备间通讯方式的软件抽象

2 软件概述

2.1 SkyEye 简介

SkyEye，中文全称天目全数字实时仿真软件，是基于可视化建模的硬件行为级仿真平台，支持用户通过拖拽的方式对硬件进行行为级别的仿真和建模。

SkyEye 以快速搭建数字化仿真系统为目标，对硬件行为以软件的方式模拟实现，可协助用户进行开发调试及测试硬件目标系统。用户成功搭建硬件目标系统后，无需修改可执行二进制文件即可使其直接运行在 SkyEye 仿真环境中。

SkyEye 产品由图形化系统建模软件 SkyEye Workstation、命令行工具、SkyEye 建模工具及其他辅助功能组成，以下将详细介绍 SkyEye 产品的各组成部分。

SkyEye 产品提供图形化系统建模软件 SkyEye Workstation。用户使用图形化软件 SkyEye Workstation 可以快速便捷地进行硬件目标系统的仿真模型搭建、SkyEye 核心启动脚本的配置、仿真系统的运行控制、仿真系统的运行信息查看、代码覆盖率统计、故障注入、GDB 调试等操作。

SkyEye 产品提供命令行工具，用于支持在命令行方式下启动 SkyEye 核心与运行仿真系统等功能，并提供仿真控制、仿真状态查看、仿真信息查看、仿真调试与故障注入等命令。

SkyEye 产品提供外设建模工具，支持用户进行二次开发，即可以由用户编写仿真系统中的外设仿真硬件。详细内容请查看《SkyEye 外设建模》。

其他辅助功能有：1、自动化测试脚本功能，为用户进行大量重复测试提供了便利。用户可以通过自动化脚本，控制 SkyEye 核心运行测试用例、控制仿真的重启、暂停和设置预期结果并比对，同时还可以在自动化测试时收集最高运行速度、设备内存泄漏等仿真模型信息。2、SkyEye 接口功能，为方便用户在测试用例运行过程中，进行数据采集、故障注入、监视地址、监视 PC 等操作，提供了 Python 语言接口，可以使用 Python 语言编写脚本，完成测试场景部署等功能。以上的两种功能可查看《SkyEye 自动化测试及接口（Python）使用说明手册》。

2.2 功能和特点

（1）支持 ARM、TI DSP、PowerPC、X86、SPARC、龙芯、飞腾、Tricore 等多种处理器体系架构。

（2）实现了包含 CPU、总线、外设等硬件模块的完整硬件目标系统的仿真。

（3）通过图形化方式快速搭建硬件目标系统。

（4）硬件目标系统搭建完成后，可在界面工具栏进行运行控制。

（5）可直接运行原二进制程序——引导程序、BIOS、操作系统、BSP、应用程序。

（6）提供现场保存和恢复、暂停、查看状态、逆向运行等功能。

（7）提供硬件级故障注入功能。

（8）提供丰富的系统级调试工具，包括运行状态控制、CPU 寄存器查看、设备寄存器查看、仿真内存查看等调试功能。

（9）提供 GDB Server 模块，并支持多种架构处理器的 GDB 调试功能。

（10）提供目标码及源码的覆盖率统计，可生成多种格式统计报告并导出。

（11）提供自动化测试功能，实现了仿真时间获取、注入故障、删除故障、查看故障、运行控制、读/写寄存器值、日志输出的自动化控制。

（12）支持仿真硬件设备模型读取激励数据文件，达到闭环测试需求。

（13）支持全局变量读写功能，可以对程序中的全局变量进行读/写操作。

（14）支持指令流记录功能，支持记录程序执行过程中的 PC 地址信息。

（15）支持地址监视功能，可以对指定地址进行读写监视功能，可以在读/写指定地址时调用相应回调函数。

（16）提供二次开发接口，支持采用 C/C++编程语言进行虚拟硬件设备建模满足二次开发需求。详细内容请参考《SkyEye 外设建模》。

2.3 性能

SkyEye 目前支持主流的嵌入式硬件平台，可以运行主流的操作系统，此外还能适配国内自主研发的操作系统天脉。通过利用基于 LLVM 的动态二进制翻译技术，使虚拟处理器在典型的桌面计算机上运行速度达到 2000MIPS 以上。

3 SkyEye 推荐配置

为了软件流畅运行，推荐最低配置：

（1）处理器：Intel I5-7600K 或 AMD R5-1500X （最少 4 个核心）

（2）内存：8G

（3）硬盘：500G

（4）运行系统：Windows7/10

4 SkyEye 的基本使用

4.1 工作台介绍

工作台旨在为工作空间内 SkyEye 工程资源的创建、管理和导航提供工具集成。工作台由菜单栏、工具栏、编辑器、视图组成。如图 4-1 所示。

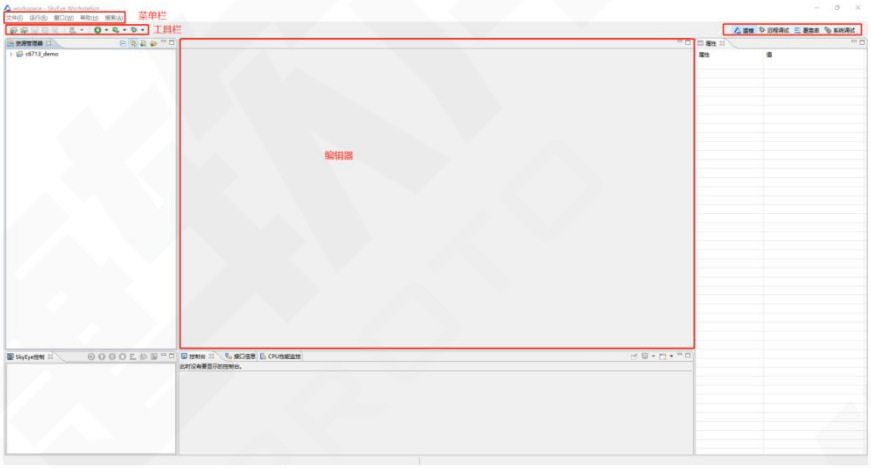


图 4-1 工作台

4.1.1 工作空间

SkyEye Workstation 在启动时需要选择工作空间。工作空间中主要用于保存当前所有的工程以及相关文件，其数据存储在工作空间中的 targets 目录下，表 4-1 为各工作空间下各目录存放内容。

表 4-1 工作空间文件/文件夹

文件/文件夹	存放的内容
bin/	其他各种工具的目录
include/	存放用户开发模块头文件的目录
modules/	包含用户开发模块的目录。所有外设模型的代码实现均在该文件夹下
targets/	SkyEye Workstation 工程的数据目录
win32/	存放用户开发模块编译完成后的动态加载库文件，这些文件会被 SkyEye 自动调用和加载
Makefile	编译所需的 Makefile 文件

4.1.2 菜单栏

菜单栏包含了 SkyEye 的大多数功能入口，并按照功能分组排列。菜单栏的功能会根据当前的透视图或活跃的视图、编辑器而启用或禁用。

4.1.3 工具栏

工作台中有三种工具栏。

主工具栏，又称为工作台工具栏，显示在工作台窗口顶部的菜单栏正下方。此工具栏的内容会根据当前的透视图而变化。工具栏中的按钮可能会根据当前活跃的视图或编辑器的状态启用或禁用。可以使用鼠标拖拽工具栏图标重新排列主工具栏的部分。

透视图切换工具栏，位于右上角，用于切换透视图。

视图工具栏，位于视图标题栏右方，主要用于视图相关的功能操作。

4.1.4 视图

视图提供操作功能并允许用户以图表形式更直观的查看项目的元数据。例如，资源管理器视图、SkyEye 控制视图、控制台视图以及属性视图等。

视图有自己的右键菜单。在视图中右键即可打开。右键菜单的选项会根据视图中所选项不同而变化。

视图有自己的工具栏，例如 SkyEye 控制视图中，可以控制 SkyEye 的运行、暂停、卸载、生成覆盖率等。

视图可能会单独出现，也可能以选项卡形式和其他视图堆叠在一起。用户可以通过打开和关闭视图以及拖动它们并停靠在工作台中的不同位置来更改它们的布局。

4.1.5 状态栏

状态栏位于工作台的底部，左侧显示了资源管理器中所选工程的信息，右侧显示了当前控制视图中的会话对应核心的仿真时间。

4.1.6 编辑器

编辑器用来进行文件浏览和编辑。不同的编辑器与不同类型的文件相关联。例如，在资源管理器视图中双击打开一个文件进行编辑时，相关的编辑器会在工作台中打开。例如双击.c 文件时，对应打开的是具有 c 语法着色、语句跳转等功能的 c 编辑器。

可以同时打开任意数量的编辑器，但一次只能激活一个。工作台窗口的主菜单栏和工具栏适用于活动编辑器的操作。

编辑器区域中的选项卡指示当前打开进行编辑的资源的名称。编辑器标题中带星号(*)表示编辑器有未保存的更改。

默认情况下，编辑器堆叠在编辑器区域中，用户可以选择平铺它们以同时查看源文件。

如以下是工作台中的 c 文本编辑器示例：

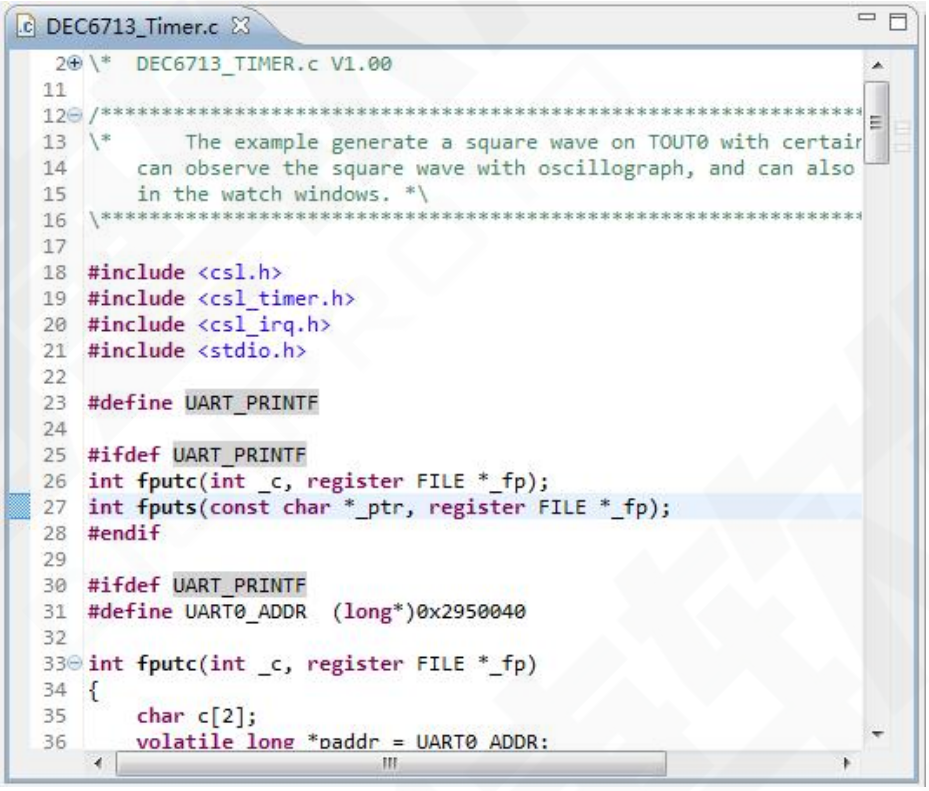


图 4-2 文本编辑器

4.1.7 透视图

工作台包含了多个透视图，每个透视图都对应了一个应用场景（各透视图功能详见 5.1 章节）。透视图定义了工作台中视图的初始设置和布局，用户可以通过菜单栏中【窗口】--【显示视图】将视图添加到当前透视图中，也可以通过【窗口】--【恢复默认布局】功能项恢复到透视图的初始布局。在工作台内，不同的透视图共享同一组编辑器并提供一组视图集合，旨在完成特定类型的任务或使用特定类型的资源。例如，远程调试透视图包含了调试程序时将使用的视图。在工作台中工作时，用户可以通过切换透视图完成不同的工作。

4.2 SkyEye 初始化配置

4.2.1 产品注册

安装完成后首次使用 SkyEye 时需要进行产品注册，如图 4-3 是用户打开 SkyEye Workstation 时的 License 验证窗口，可以选择本地验证或网络验证。

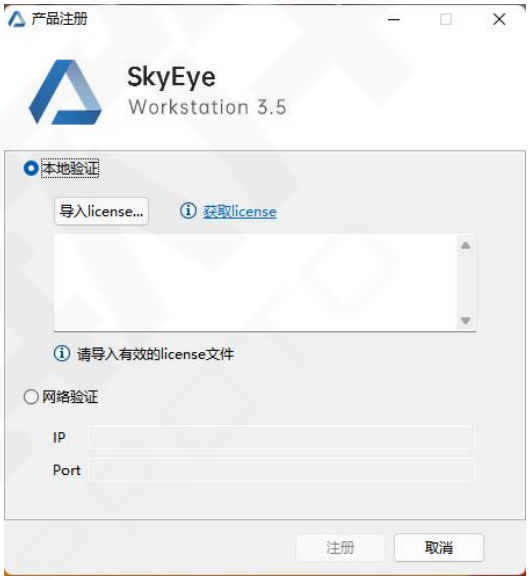


图 4-3 产品注册界面

4.2.2 本地验证

选择本地验证，步骤如下：

本地验证需要 License 文件。如没有，需点击“获取 license”，再选择需要访问权限的网卡，会生成如图 4-4 显示的密文或二维码，将显示的密文或二维码发给厂商，以获取有效的 License。

图 4-4 二维码界面

准备好 License 后，点击导入 License，选择 License 文件后点击打开，如图 4-5

所示。

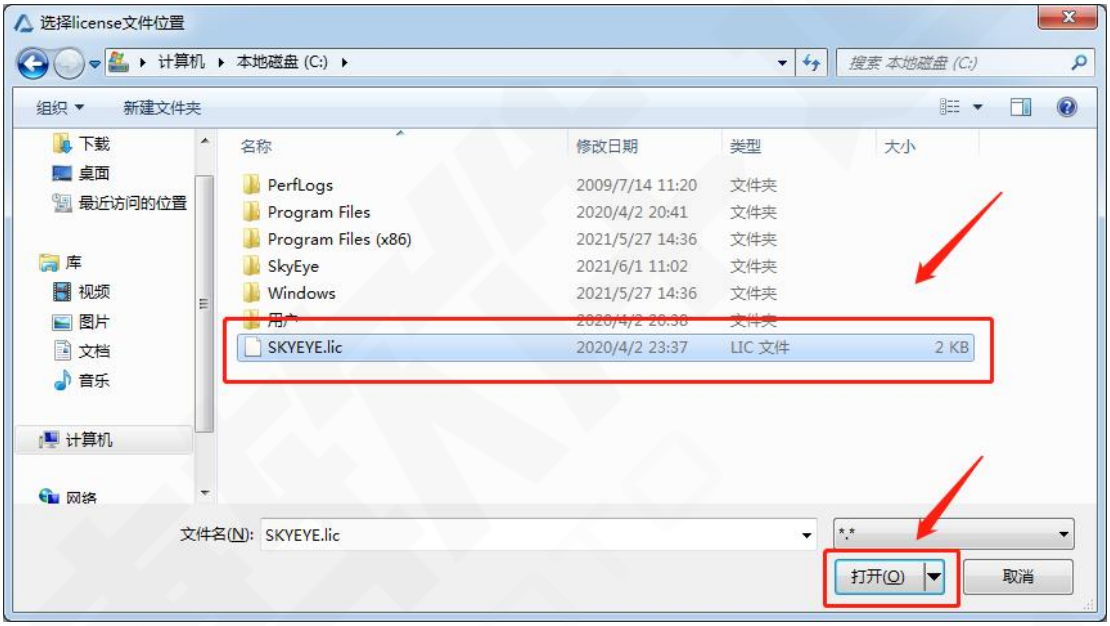


图 4-5 导入 Lisence 文件

界面将显示该 License 的有效期、授权核心等信息，如图 4-6 所示，如 License 正确，点击注册后即验证成功。

图 4-6 Lisence 相关信息

4.2.3 网络验证

选择网络 License 验证需要输入 License 服务器的 IP 地址及端口号，如图 4-7 所示，然后点击连接即可验证成功。

图 4-7 网络验证 Lisence

4.2.4 首次设置工作空间

License 验证完后首次打开 SkyEye，需要设置工作空间，如图 4-8 所示

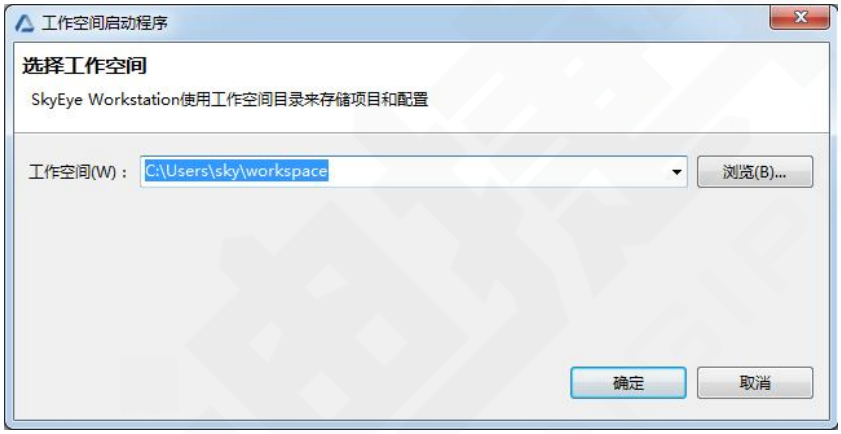


图 4-8 设置工作空间

工作空间的默认路径为 C:\Users\%UserName%\workspace 目录，用户可以点击“浏览”按钮更换工作空间。

4.2.5 欢迎页

首次进入 SkyEye Workstation 可以进入欢迎页，如图 4-9 所示，欢迎页可以帮助用户快速搭建工程。

图 4-9 欢迎页面

4.2.6 工作空间的切换

通过图 4-10 所示的操作可以在非工程加载或运行时进行工作空间的切换。

图 4-10 切换工作空间

单击“其他”后，会弹出如图 4-11 的对话框。

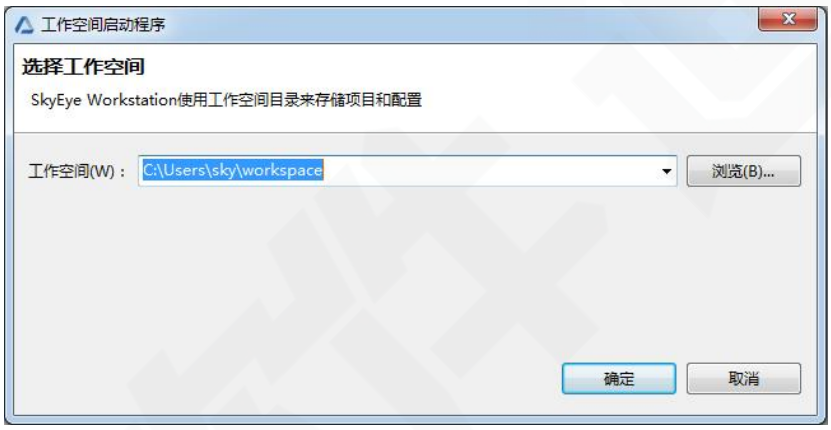


图 4-11 选择工作空间路径

这时就可以选择切换的工作空间路径，确定后切换到该工作空间。

4.3 快速开始

在安装目录下的 demo 目录中，提供了一个简单的 c6713_demo。选择欢迎界面中的导入工程（详见 4.8.2 节），如果用户导入了一个完整的 SkyEye 工程，可以跳过 4.4 节搭建 SkyEye 工程和 4.5 节配置启动脚本文件，进行加载和仿真。如图 4-12 所示可以导入 demo 工程。

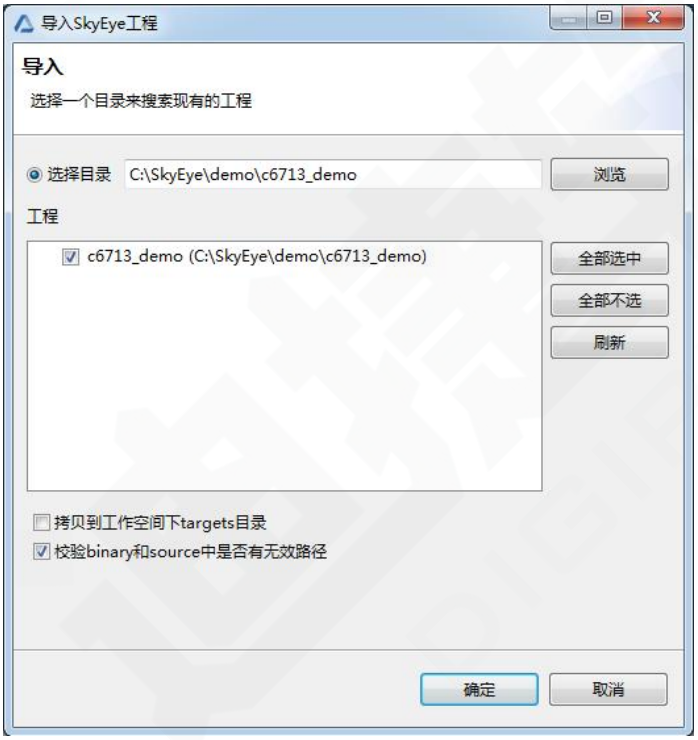


图 4-12 导入 demo 工程

4.4 搭建 SkyEye 工程

SkyEye 是以 json 格式来描述一个硬件目标系统。在 SkyEyeWorkstation 中，可通过拖拽组件库组件方式快速构建虚拟目标系统模型，自动生成 json 文件，无需手写代码。通过这种可视化图形的硬件建模方式，软件研发或测试人员能够更快速地搭建硬件模型。

4.4.1 创建新的工程

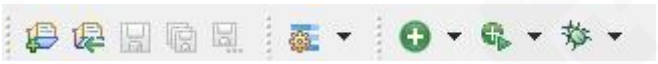
按以下操作步骤完成工程的新建：

第一步：调出新建工程向导。界面上有多种方式能调出新建工程向导：

1. 菜单栏【文件】--【新建工程】。

图 4-13 新建工程菜单

2. 工具栏默认布局时第一个按钮。



3. 资源管理器未选择资源时，右键--【新建工程】。如图 4-14 所示。

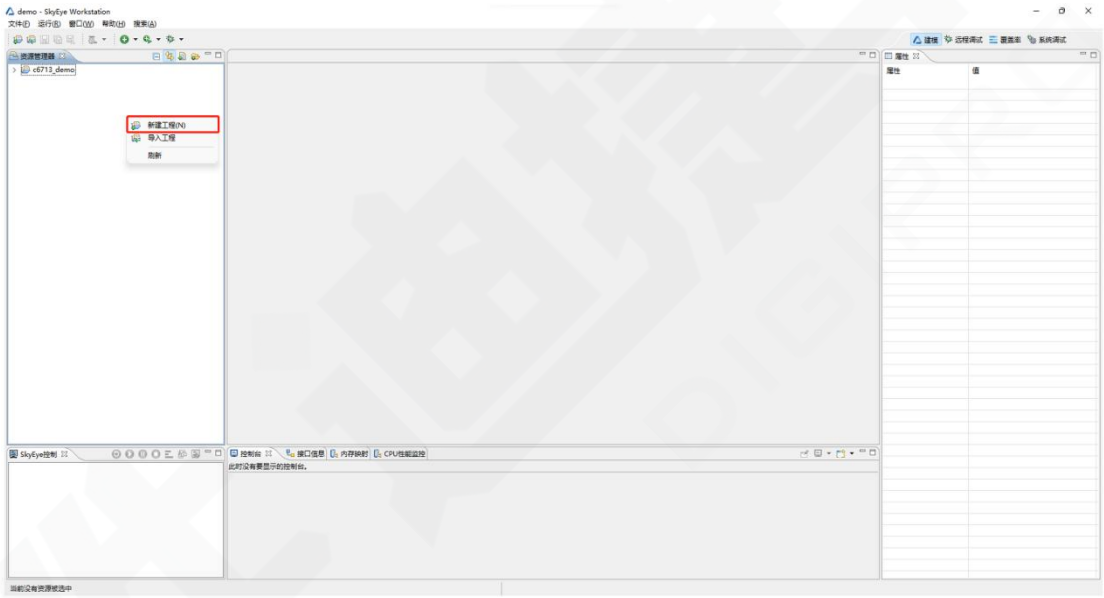


图 4-14 通过资源管理器新建

4. 资源管理器选择任意资源，右键--【新建】--【工程】。如图 4-15 所示。

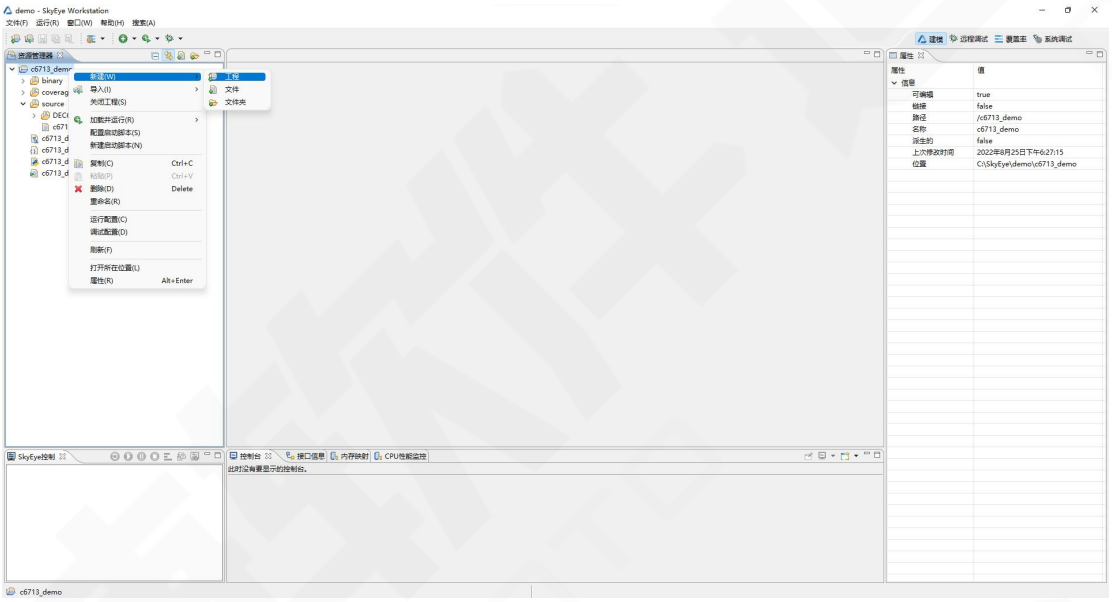


图 4-15 通过资源管理器新建

5. 快捷键 Ctrl+N，选择 SkyEye 工程。如图 4-16 所示。

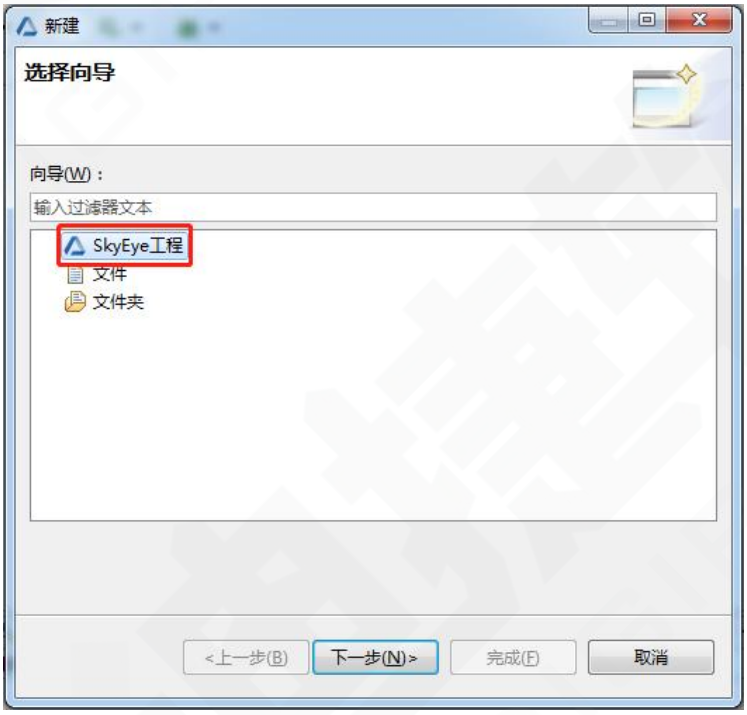


图 4-16 快捷键新建向导

第二步：输入合适的工程名并设置路径。使用默认地址该项目将保存在当前工作空间下，或取消使用默认地址自定义选择路径。注意：工程名和自定义地址都只能由字母数字和下划线组合。

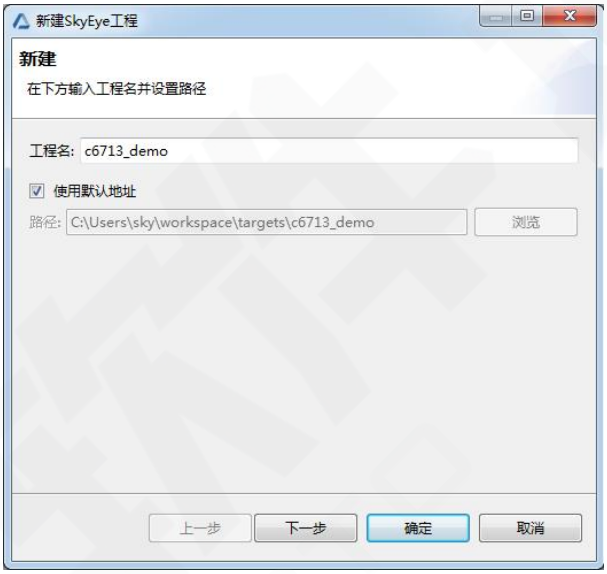


图 4-17 输入合适的工程名

第三步：导入可执行文件。选择要加载的可执行文件，根据实际情况可添加多个，如图 4-18 所示。

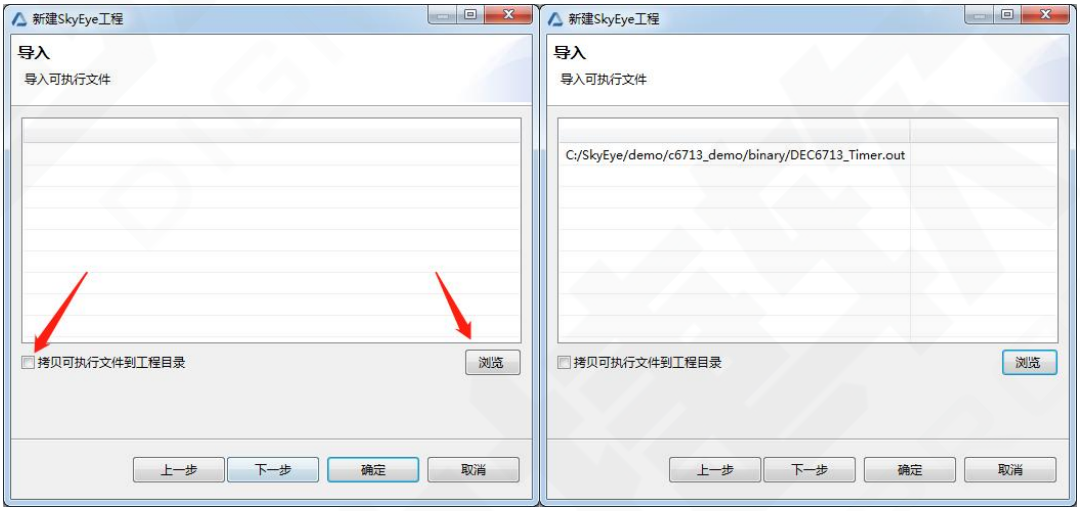


图 4-18 选择可执行文件

添加完成后，通过“拷贝可执行文件到工程目录”复选框决定是否需要拷贝可执行文件到工程目录下。拷贝到工程目录后配置启动脚本文件将使用相对路径，方便迁移到其他机器上，但重新编译二进制文件工程后需要重新拷贝到工程目录下；不拷贝到工程目录时，配置启动脚本文件将使用绝对路径，重新编译二进制文件工程后不需要重新拷贝，但迁移到其他机器时需要手动迁移并重新配置该二进制文件。

第四步：导入源码目录。添加生成二进制文件的源码目录到当前工程下，如图 4-19 所示，配置源码目录后可以进行源码覆盖率的统计并且远程调试时可以自动映射源码。

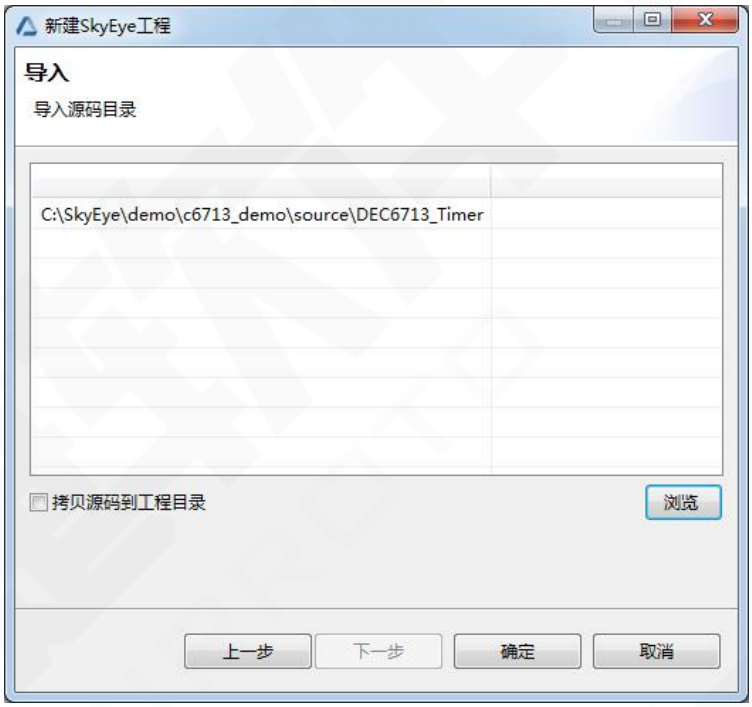


图 4-19 选择源码目录

在第三步和第四步时，用户也可以通过直接点击“确定”按钮暂时跳过该步骤。后续通过界面中的功能（可参考 4.8.2 章节）导入二进制文件或源码。以上操作完成后，单击“确定”即可看到刚才所创建的工程。以下是创建完成后的工程目录，如图 4-20 所示。每个目录及文件的作用说明如表 4-2。

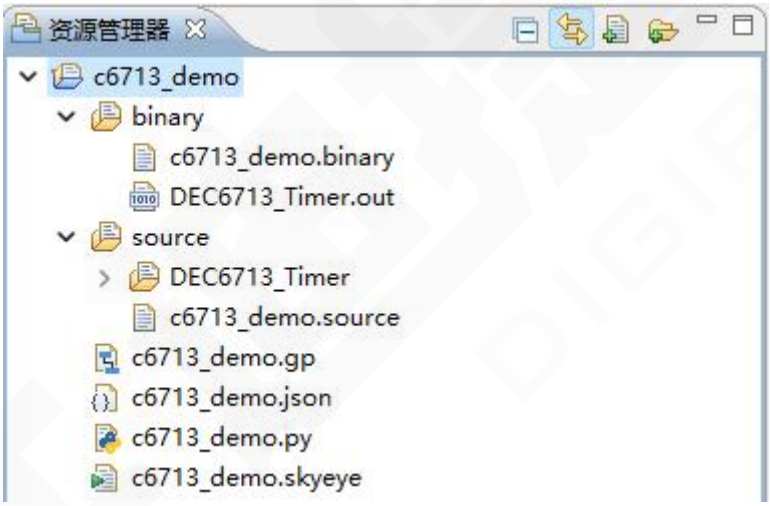


图 4-20 工程目录

表 4-2 工程目录说明

文件/文件夹	存放的内容
binary/	存放二进制文件及.binary 文件
.binary	存放二进制文件的绝对（相对）路径
source/	存放源码目录及.source 文件
.source	存放源码目录的绝对（相对）路径
.gp	图形建模文件
.json	硬件系统描述文件
.py	自动运行控制脚本文件，用以在工程运行时自动执行模拟器控制或目标系统监视相关功能
.skyeye	启动引导脚本文件，用以进行工程运行相关的配置

4.4.2 使用图形界面搭建硬件系统

双击工程中 gp 文件，将会打开图形化搭建界面，如图 4-21。可以通过点击所需组件后点击画布进行放置，再将不同组件组合以搭建虚拟目标环境。

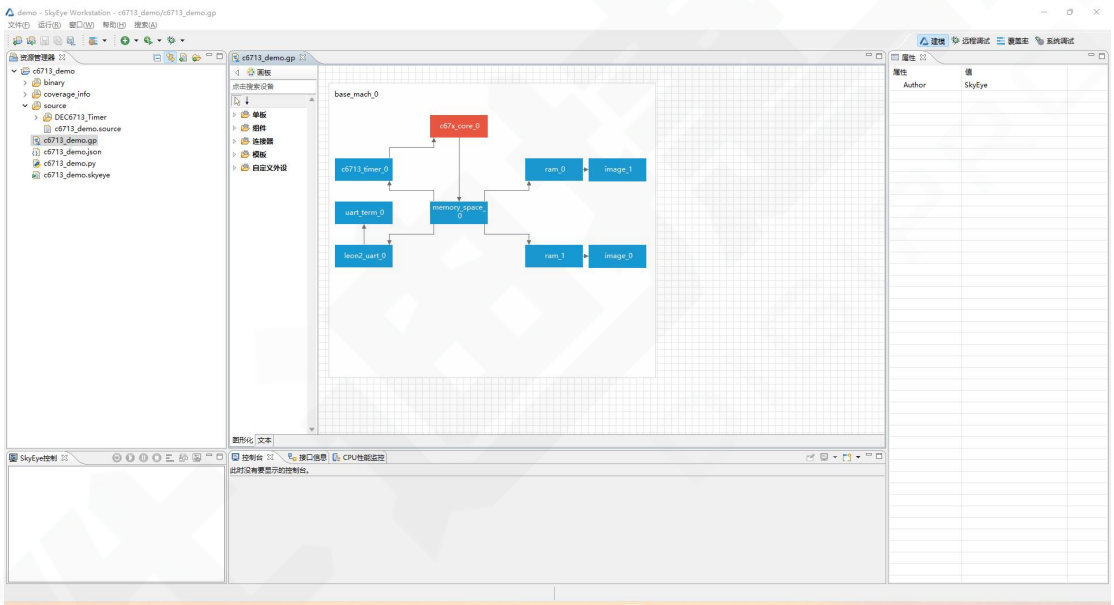


图 4-21 gp 文件主界面

本文档以构建核心为 C6713 的目标系统为例，需要添加以下组件完成目标系统的搭建：

1. 单板\base_mach
2. 组件\core\DSP\c67x_core
3. 组件\on_chips\uart\leon2_uart
4. 组件\off_chips\others\memory_space
5. 组件\on_chips\others\ram
6. 组件\on_chips\timer\c6713_timer
7. 组件\off_chips\uart\uart_term

各个组件的作用说明如下：

base_mach：对应物理世界的板卡。

c67x_core：DSP c67x 系列的核心。

leon2_uart：串口。

memory_space：SkyEye 对处理器寻址空间的抽象。

ram：内部存储器。

c6713_timer：c6713 定时器。

uart_term：功能性组件，通过该组件将串口和终端进行交互。

寻找组件时可选择手动在目录中查找或直接通过搜索框进行搜索。找到对应需要添加的组件后，先鼠标点击该组件，然后在建模区想要放置的地方单击鼠标左键，就将对应的组件放置在了指定位置，上述组件拖出后效果如图 4-22 所示。

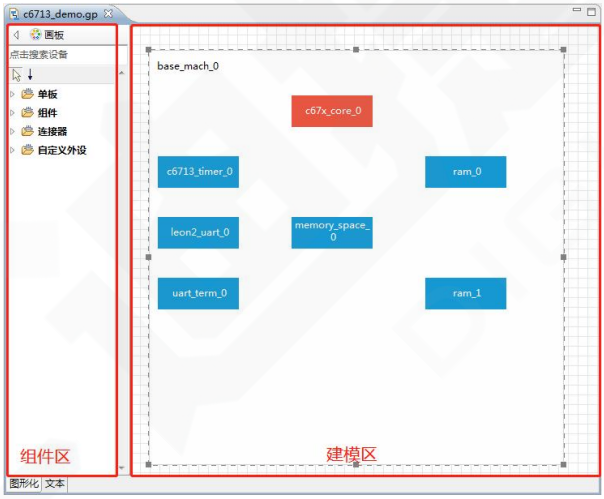


图 4-22 gp 文件单板图

组件准备完成后，需要通过连线将各个组件连接起来，SkyEye 中两个虚拟组件的连接方式是通过软件抽象出来的接口进行连接的。连接时，需要配置接口信息，如图 4-24 所示，memory_space 类是 SkyEye 抽象出来的地址空间，目的是让处理器寻址访问其他设备使用，memory_space 的实例化组件 memory_space_0 和 ram_1 通过 memory_space 接口连接，memory_space 接口用来配置处理器核心的寻址空间，因此就需要配置 ram_1 映射地址的起始地址和长度，接口信息设置完成后也可以通过双击组件之间的连线进行修改。同时，如果需要配置单个组件的属性，例如定时器 timer 的中断号属性 int_number，需要选中组件 c6713_timer_0 后在属性视图中设置，如图 4-24。

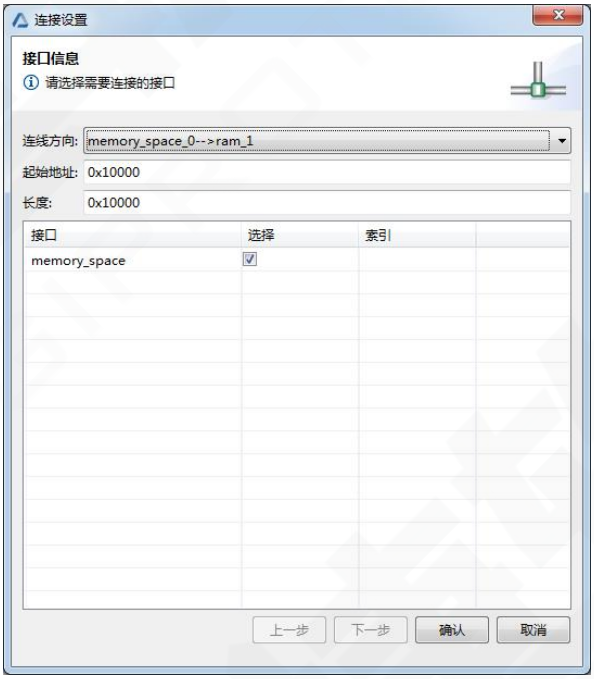


图 4-23 连线设置

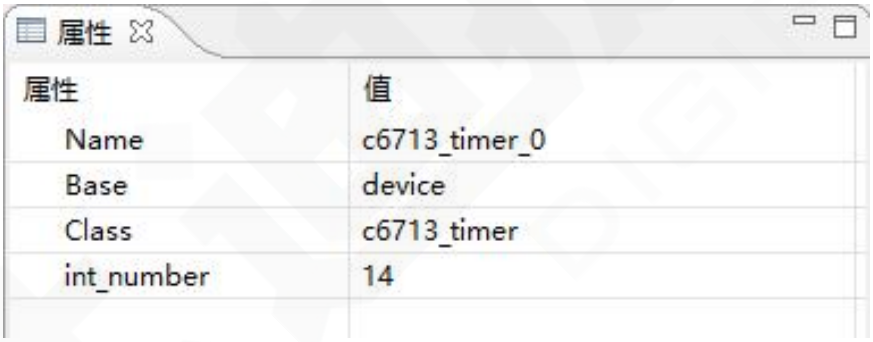


图 4-24 组件属性设置

带存储空间的设备在 SkyEye 中可以通过连接一个抽象的 image 模块来实现，当 memory_space 连接 ram 时 SkyEye 会自动在 ram 右边连接上一个 image，image 长度会根据 ram 长度自动设置。连线和属性设置完成后，如图 4-25 所示。

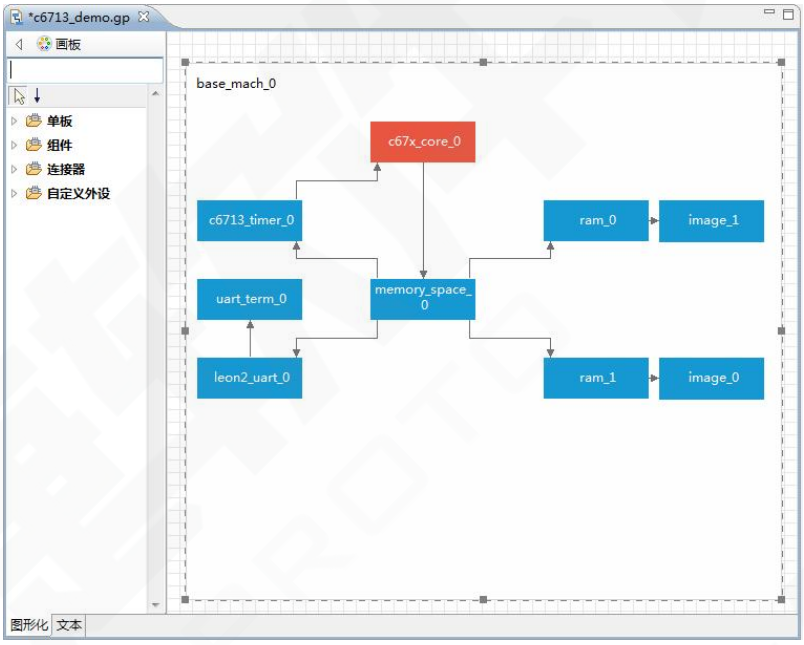


图 4-25 配置完成后的单板图

以下为本例中需要配置的属性（这些配置信息依据硬件手册）：

1.连线 memory_space_0 到 ram_0，并设置地址为 0x000，长度为 0x600。

2.连线 memory_space_0 到 ram_1，并设置地址为 0x10000，长度为 0x10000。

3.连线 memory_space_0 到 leon2_uart_0，并设置地址为 0x2950040，长度为 0x40。

4.连线 memory_space_0 到 c6713_timer_0，并设置地址为 0x1980000，长度为 0x40000，连线 c6713_timer_0 到 c67x_core_0，因其之间只有一个 core_signal_intf（中断触发接口）接口，将自动选择该接口，如果碰到多接口的会弹出连线配置框进行选择。

5.连线 uart_term_0 到 leon2_uart_0。

6.在右侧属性栏中设置 c6713_timer_0 的 int_number（中断号）为 14。

4.4.3 SoC 的使用

若上小节中搭建的模型复用率高，用户可选择右键保存为 SoC 设备，以便其他工程复用。SoC（System on Chip，片上系统），是一个集成电路，也被称为

“芯片”，如图 4-26。

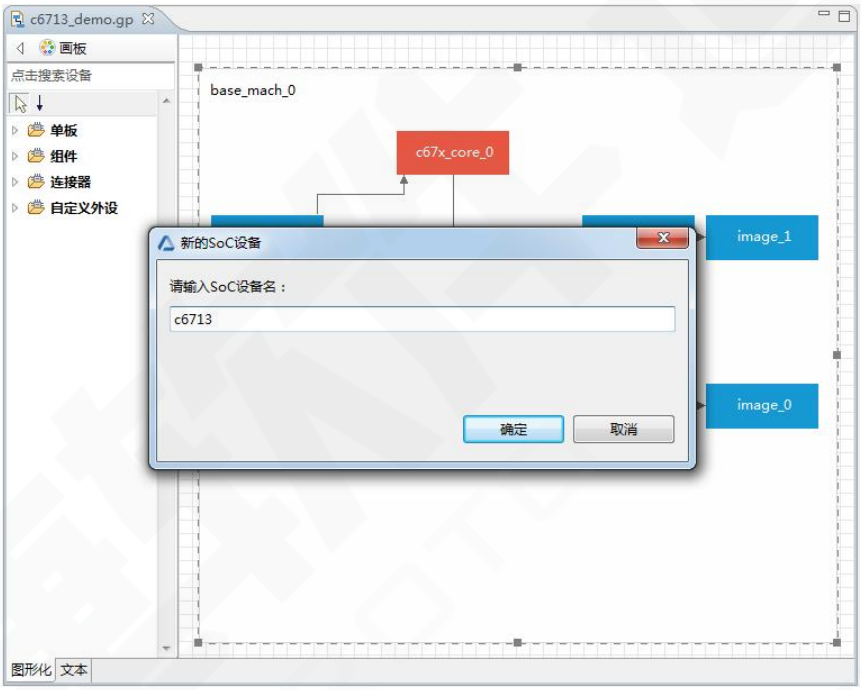


图 4-26 保存为 SoC

确定后在组件目录就会生成 SoC 文件夹，可以像其他组件一样使用。

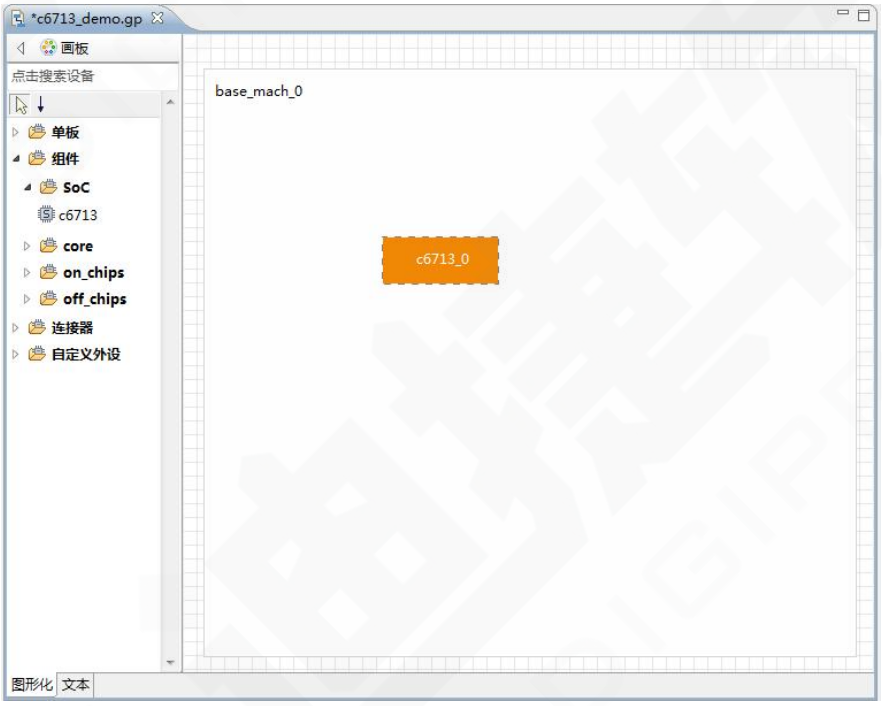


图 4-27 含 SoC 的板子

双击 SoC 或右键点击进入内视图能进入该 SoC 的具体组成部分，如图 4-28 所示，且能进行修改。按 Esc 或点击右键菜单的返回上一级便能回到主视图。

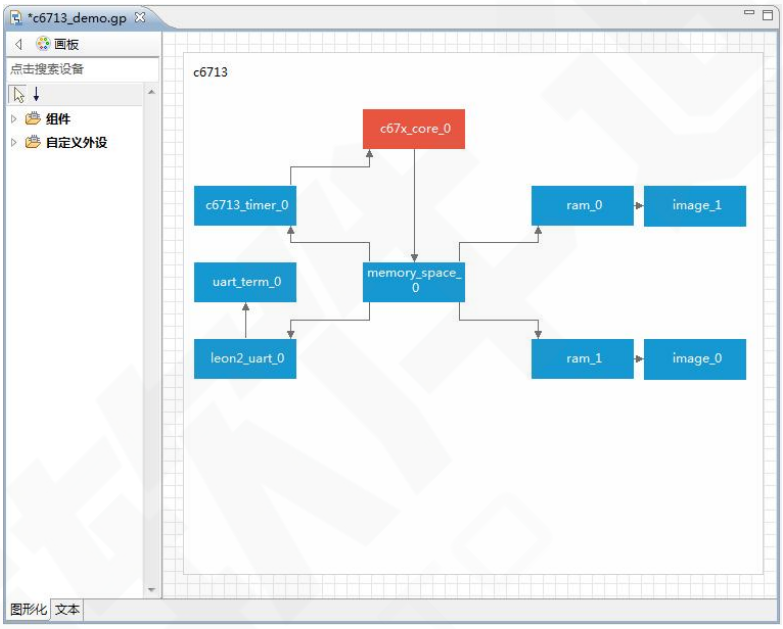


图 4-28 SoC 内视图

在工程中修改 SoC 后，为了保证其他工程的 SoC 一致性，建议在保存工程前将该 SoC 的修改内容更新到 SoC 中（在 SoC 内部时右键板子->更新 SoC 设备，在 SoC 外部时右键 SoC->更新 SoC 设备），在更新 SoC 时，会展示工作空间下所有受影响的工程列表，如图 4-29。

图 4-29 更新 SoC

更新 SoC 后打开含有该 SoC 的 gp 文件，将弹出如图 4-30 提示。如果选是将自动更新该 gp 文件中的 SoC，选否则需要将该 SoC 进行另存为。

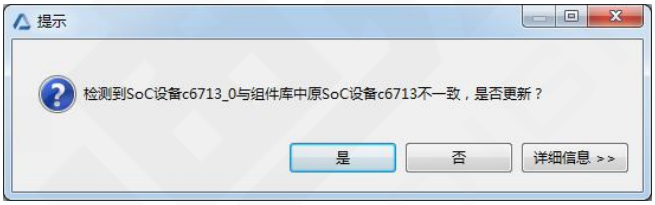


图 4-30SoC 校验

注意：SoC 数据存放在安装路径\eclipse\SoC 中，若要移动或复用 SoC，要遵循其目录相对路径规则，且不可随意更改文件或文件夹名，以防软件识别错误。

4.5 配置启动脚本文件

工程的加载与运行必备的 3 个文件分别是目标二进制文件、虚拟目标系统构建文件(.json)及启动脚本文件(.skyeye)。导入二进制文件可以参考第 4.8 节工程管理的相关内容。通过前一节的配置完成模型搭建后，打开虚拟目标系统构建文件可以看到已经自动生成对应的 json 格式文本。该节主要内容是启动脚本文件.skyeye 的配置。

SkyEye 项目支持一个工程配置多个启动脚本文件。默认启动脚本文件是新建工程时自动创建的与工程同名的 skyeye 文件，如图 4-31 所示。

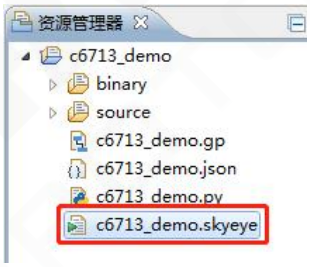


图 4-31 默认启动脚本文件

4.5.1 新建启动脚本文件

创建启动脚本文件有两种方法，一种是直接在工程右键菜单中选择新建启动脚本。

图 4-32 新建启动脚本

另一种是在配置启动脚本的弹窗中点击“配置另存为”，此时会将当前配置文件另存为一个内容一致的.skyeye 文件。

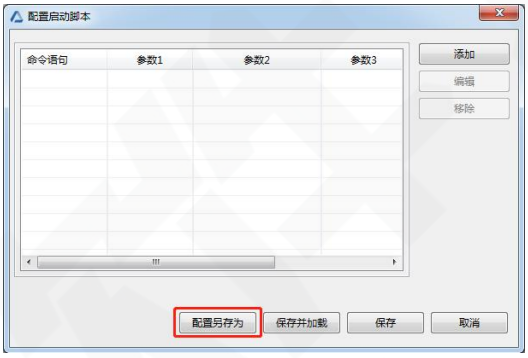


图 4-33 配置另存为

4.5.2 配置启动脚本文件

选中工程 -> 右键 -> 配置启动脚本，配置窗口如图 4-34 所示。

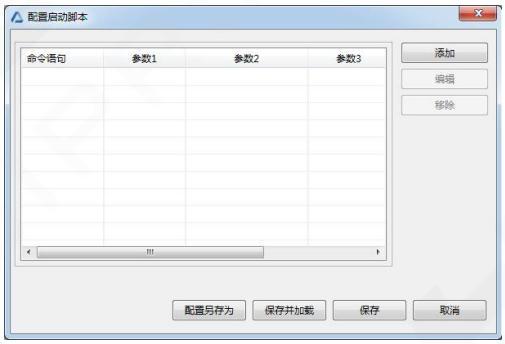


图 4-34 配置启动脚本

可以通过右侧添加、编辑和移除来配置脚本命令，命令的具体使用方法和用途可以参考第 7 章中内容。

本例中将添加一个 load-binary 命令，如图 4-35 所示。

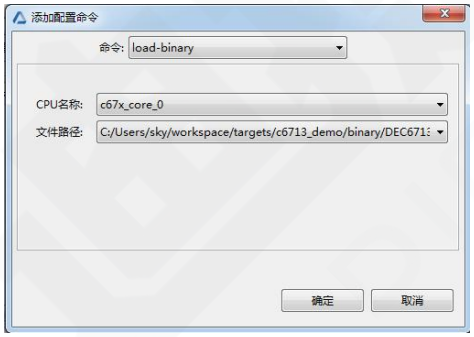


图 4-35 添加 load-binary 命令

注意：会话运行后，启动脚本内信息便已读取完毕，运行时修改配置脚本不会对该次运行起作用。

4.6 运行配置

运行配置，即配置加载时工程使用的启动引导脚本，配置将以会话的形式保存。选中工程并点击主工具栏的加载  按钮后，如果不存在该工程的配置，则会自动创建一个运行配置（优先使用工程同名的.skyeye）进行运行；如果存在历史记录，则直接运行最近一条历史会话记录。用户也可以按照需要增加运行配置。运行配置的入口有三个，一是菜单栏--【运行】--【运行配置】；二是工具栏中的加载或加载并执行按钮下拉框中的运行配置 

；三是工程右键菜单--【运行配置】。

运行配置的界面如图 4-36 所示。

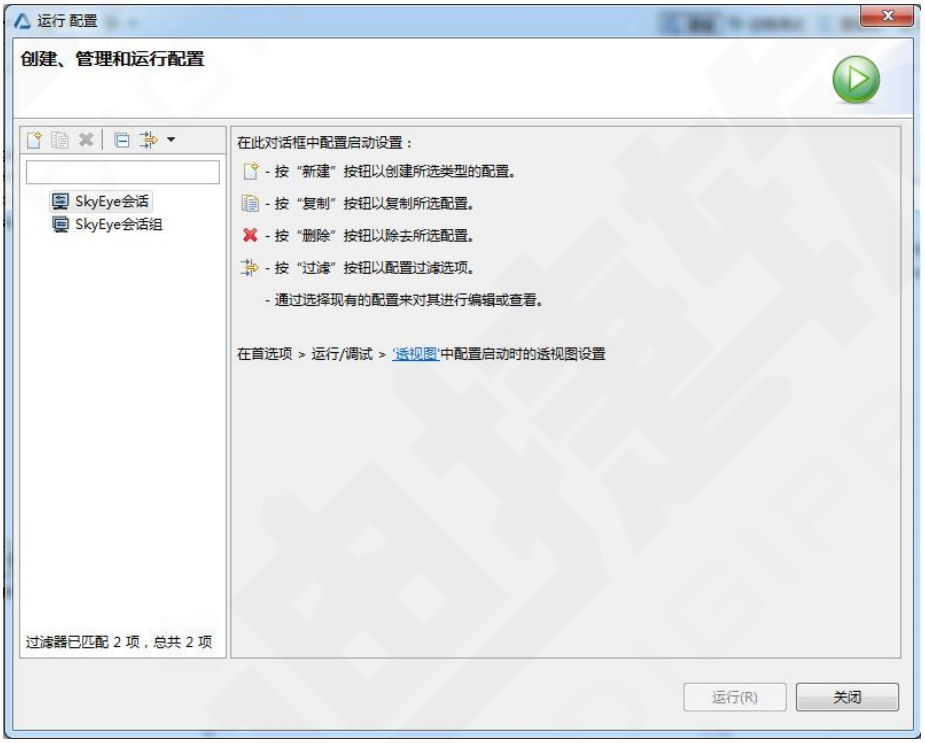


图 4-36 运行配置

双击 SkyEye 会话或点击上方  按钮进行新建会话，SkyEye 会话配置如下图。

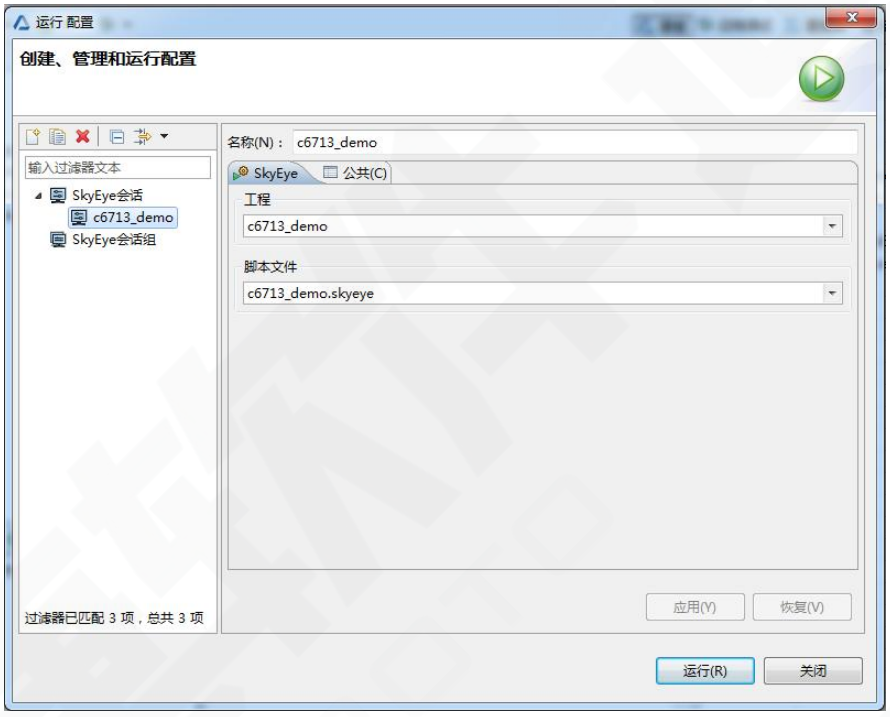


图 4-37 SkyEye 配置

可以进行会话名、工程名和脚本文件的选择。

4.7 SkyEye 会话的控制

SkyEye 会话加载后，该会话将出现在 SkyEye 控制视图中。如图 4-38 是成功加载的界面。

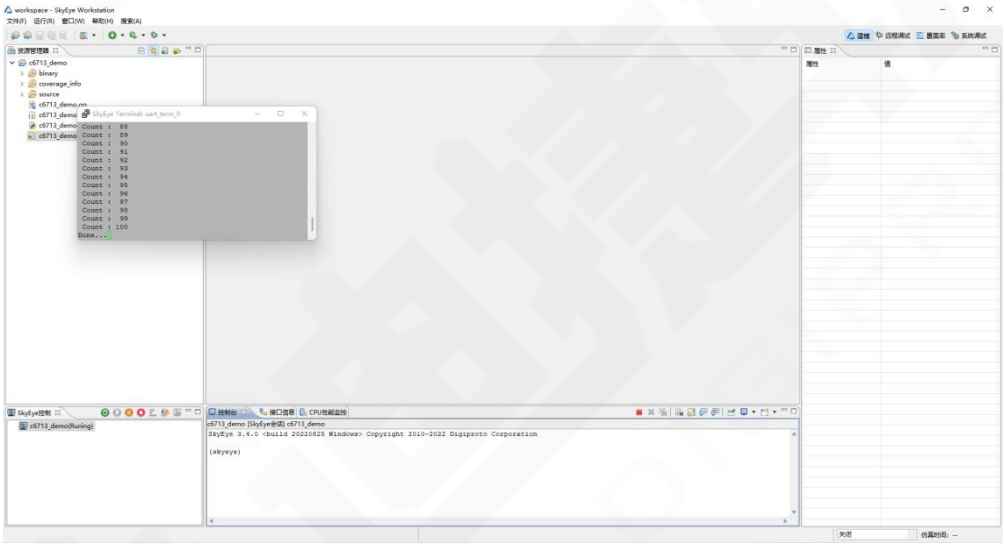


图 4-38 加载成功界面

点击 SkyEye 控制视图中的运行按钮 运行仿真，如下图是该会话的运行结果。

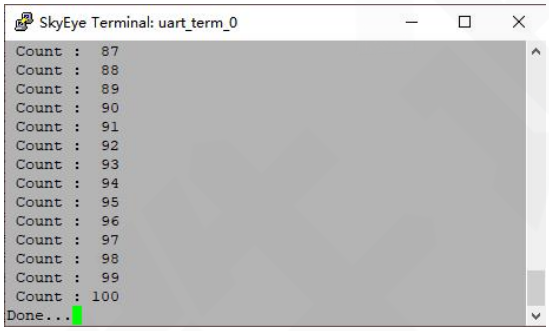


图 4-39 运行结果

运行过程中可以点击暂停，或者卸载。

4.8 工程管理

整个工程管理操作主要包含新建、导入、复制/粘贴、删除、重命名、调试等，所有以上操作可以在对应的工程通过右键完成。

4.8.1 新建工程

工程管理中的新建操作参考 4.4.1 节“创建新的工程”。

4.8.2 导入

导入操作主要包括导入工程、二进制、源码以及导入其他相关文件或文件夹。已有的工程可以通过导入操作进行管理；binary 导入的是需要加载的二进制文件；source 导入的是远程调试、覆盖率统计需要用到的源码目录。通过在对应工程上“右键”来实现以上操作，如图 4-40 所示。

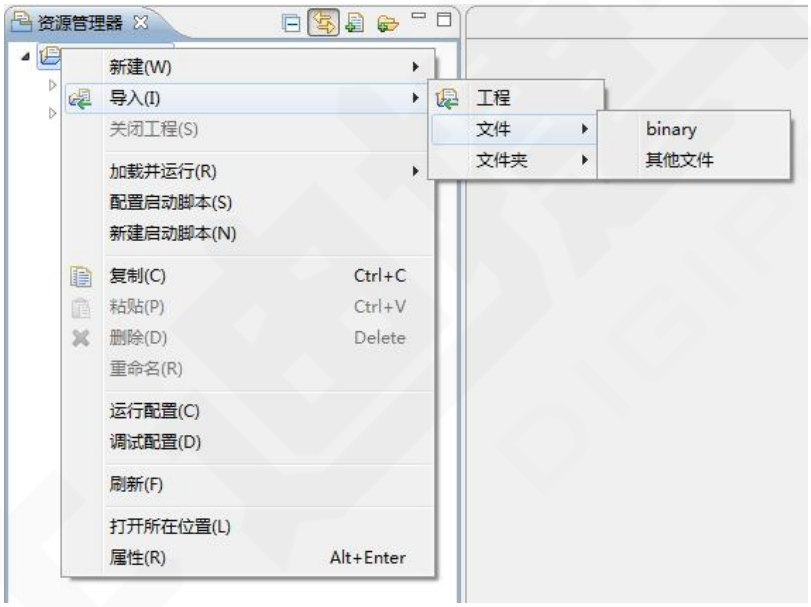


图 4-40 导入操作

4.8.3 复制/粘贴工程

对于已经创建的工程，如果需要复制/粘贴一份相同的工程副本，可以通过复制/粘贴操作来完成。

首先是选中对应的项目进行复制，然后在工程界面的空白处右键并选择粘贴，此时会弹出如下弹框，可根据自己的需要重命名，此处项目名称不可重复。

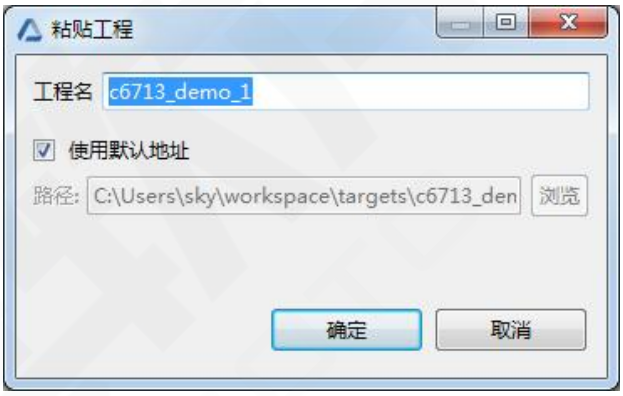


图 4-41 复制/粘贴工程

4.8.4 删除工程

可以通过“删除”对已经创建的工程进行删除。

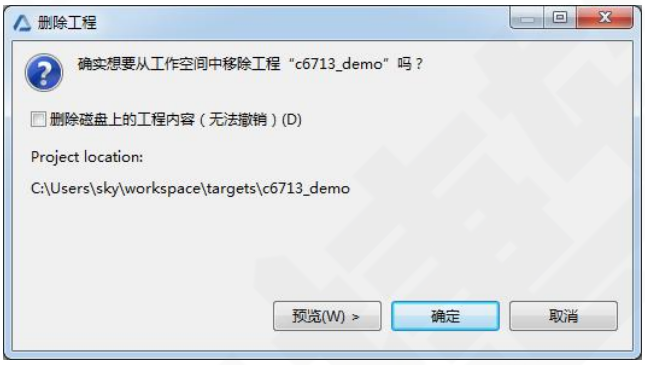


图 4-42 删除工程操作

直接删除只是在资源管理器中不显示该工程，要想完全删除磁盘中的工程，需要勾上“删除磁盘上的项目内容”复选框。注意：此操作不可撤销，用户无法找回删除的文件。

4.8.5 重命名工程

重命名操作用于对原有工程进行工程名的修改，重命名后原有项目中的相关工程名均会被替换，包括一些配置文件中的二进制文件名字、路径等信息。

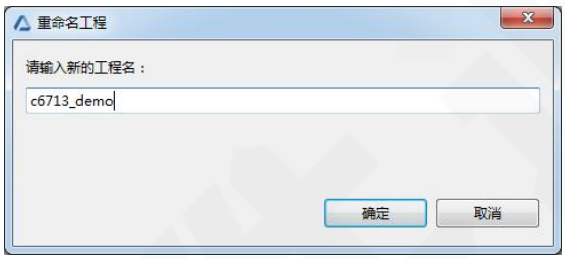


图 4-43 工程重命名

4.8.6 刷新

资源管理器中刷新分为三种刷新。

1.工程右键下的刷新，该刷新是刷新整个工程目录。

图 4-44 工程刷新

2.binary 文件夹的刷新，右键 binary 目录并点击刷新 binary 文件配置后会自动将 binary 文件夹下的二进制文件同步到.binary 文件中。比如将二进制文件复制粘贴到 binary 文件夹下后，该刷新将把 binary 文件夹下的文件路径自动写入.binary 文件中，省去导入的操作。

图 4-45 binary 文件刷新

3.source 文件夹的刷新，和 binary 文件刷新类似，右键 source 目录并点击刷新 source 文件配置，会自动读取 source 目录下的文件路径，并写入.source 文件中。

图 4-46 source 文件刷新

4.9 仿真时间

仿真时间即虚拟目标系统运行的时间，SkyEye 采用统一的时钟调度机制来管理整个虚拟的芯片目标系统的时间。因此，通过 SkyEye 搭建的虚拟目标系统采用统一的虚拟时间来保证系统的实时性，即不论宿主机运行性能如何，SkyEye 都能保证每秒执行的指令数是确定的，当 SkyEye 的仿真性能和真实处理器主频一致的时候，即能达到和真实目标系统的时间一致，而当 SkyEye 性能比真实的处理器快 5 倍时候，SkyEye 即可以实现超实时仿真，比真实系统运行速度快 5 倍；而当 SkyEye 性能比真实处理器慢的时候，例如 100M 的主频，SKyEye 真实性能是每秒只能运行 10M 的指令，那么虚拟时间的 1s，相当于真实时间的 10s，也就是比真实时间慢 10 倍。

但无论 SkyEye 比真实目标系统快、相同、慢这三种情况下，SkyEye 都可以保证严格的实时仿真，因为所有设备都是虚拟的，所以可以同步加快、保持不变或同步放慢，从而达到实时精度要求。

仿真时间位于状态栏右侧，运行 SkyEye 会话后，初始状态 CPU 选择框为关闭状态，通过下拉框选择对应的 CPU，即可看到对应 CPU 的仿真时间。图 4-47 为开启仿真时间的界面。

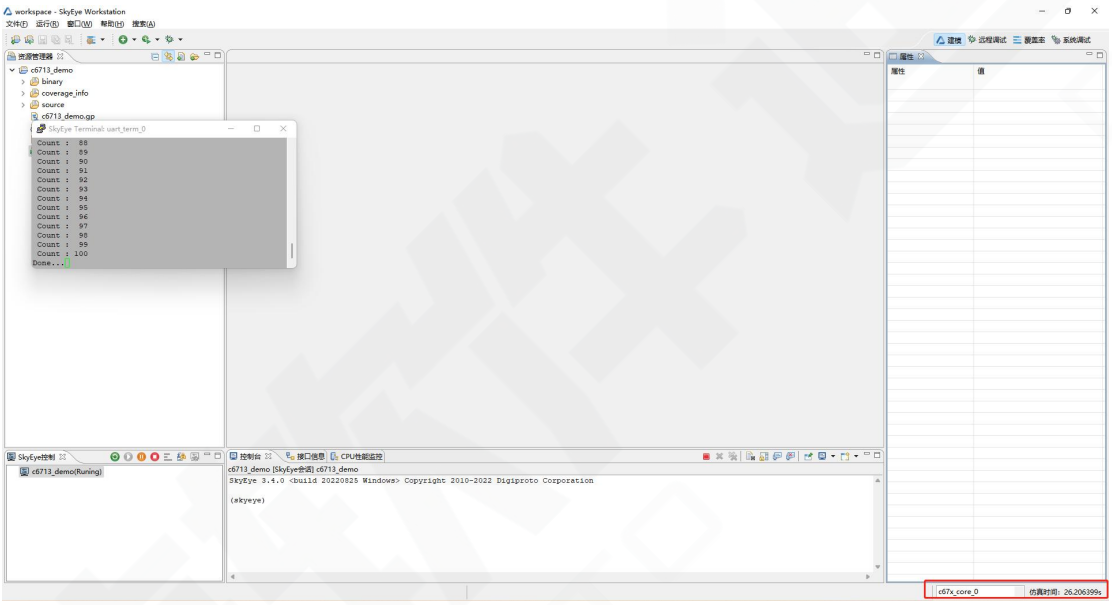


图 4-47 开启仿真时间界面

5 SkyEye 功能说明

本章节将在第 5.1 节至第 5.9 节详细介绍 SkyEye Workstation 中各功能，第 5.8 节为 SkyEye 自动化测试的介绍和使用，第 5.9 节为配合自动化测试使用的 SkyEye 接口介绍。

5.1 透视图

在 SkyEye Workstation 中透视图为若干视图的集合，各自拥有独有的功能。软件中定义了四个透视图：建模透视图、系统调试透视图、远程调试透视图、覆盖率透视图。

建模透视图主要负责工程的编辑配置，常用视图有资源管理器、SkyEye 控制、属性、接口信息、CPU 性能监控、控制台。

系统调试透视图用于查看和修改当前仿真工程的系统级信息。默认打开的视图有内存映射、处理器寄存器、反汇编查看、故障注入、内存查看、设备寄存器、资源管理器和 SkyEye 控制。部分调试操作在仿真暂停时才能操作。

远程调试透视图服务于 GDB 调试，进行 GDB 调试的时候会自动切换至该透视图中。

覆盖率透视图在点击生成覆盖率按钮后自动切换，并在下方的覆盖率视图

中显示出具体覆盖率详情。

透视图可通过界面右上角或菜单栏【窗口】--【打开透视图】进行切换，如图 5-1 所示。

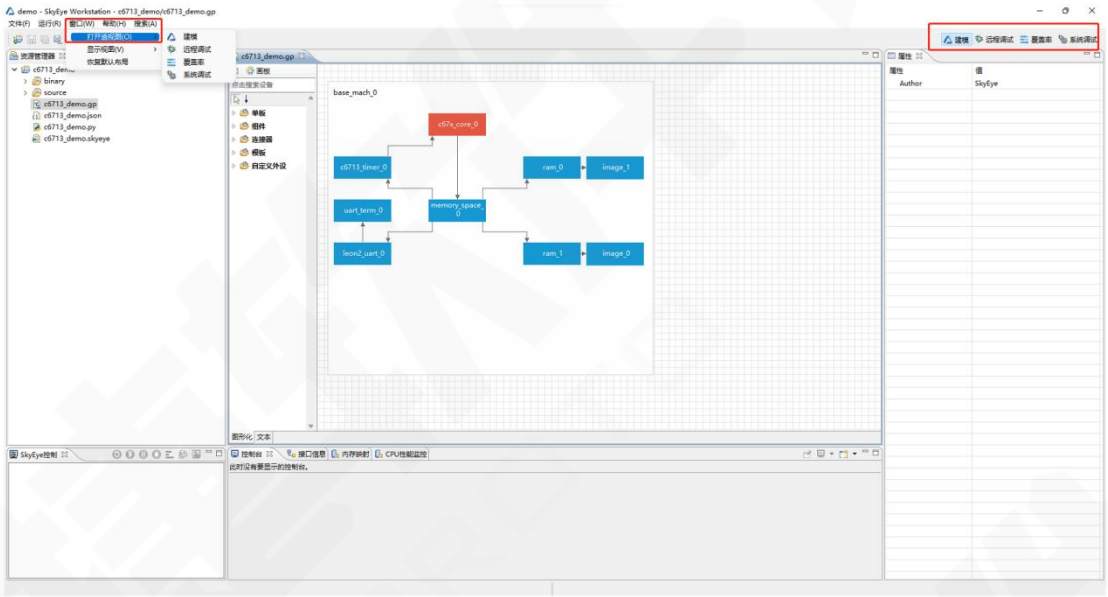


图 5-1 切换透视图

5.2 视图

当关闭资源控制台、管理器或属性等视图后，可以在【菜单栏】--【显示视图】中重新打开，如图 5-2 所示。默认只展示一些常用视图，用户可通过点击【其他】调出更多视图。

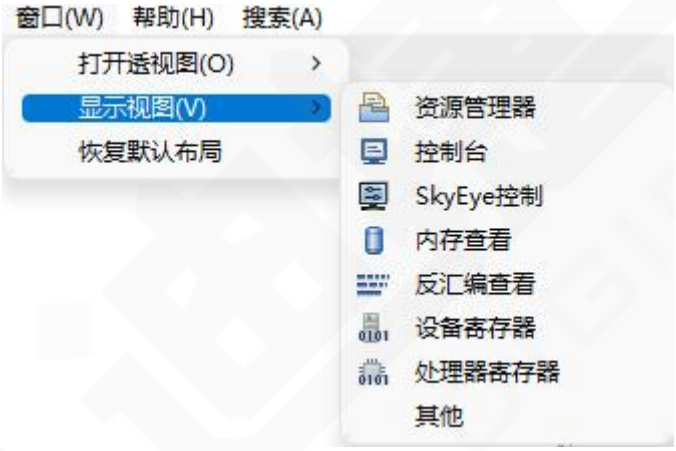


图 5-2 显示视图

下面介绍部分视图的作用。注意：只有当仿真暂停时，某些视图（处理器寄存器视图、反汇编查看视图、故障注入视图、内存查看视图、内存映射视图）的

数据才会更新和允许修改。

5.2.1 资源管理器

资源管理器用以进行工程相关的目录及文件的管理，资源管理器有自己的右键菜单及菜单栏，可以进行文件的新建、导入、删除、重命名等。

图 5-3 资源管理器视图

5.2.2 控制台

会话运行或调试启动后，在控制台视图中会生成 SkyEye 或调试的 gdb traces 终端控制台，进行数据的输出，同时控制台集成了命令行功能，SkyEye 也支持通过命令行进行控制，详细可参考《SkyEye 命令行手册》（注：为了保证实时输出，c 代码中需要在输出后进行缓存清理，例如使用 printf fflush(stdout) ）。可在如图 5-4 所示红框处切换控制台。

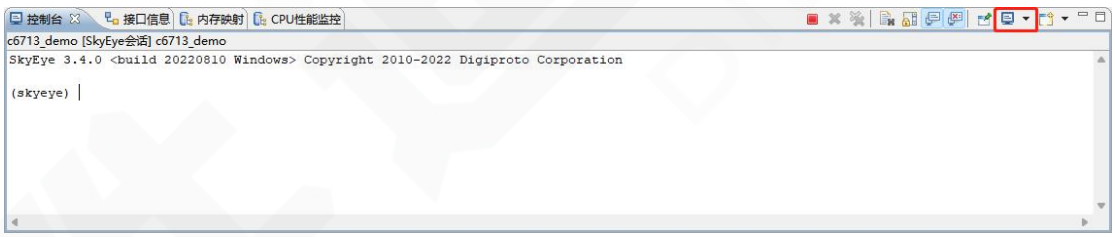


图 5-4 控制台视图

5.2.3 SkyEye 控制

SkyEye 控制视图用来控制 SkyEye 会话的运行、暂停、重启和卸载、生成覆盖率等功能。如图 5-5 所示。视图中按钮从左到右排列功能依次为：重启会话、运行/继续运行会话、暂停会话、卸载会话、生成覆盖率、异常/中断触发、连接调试。

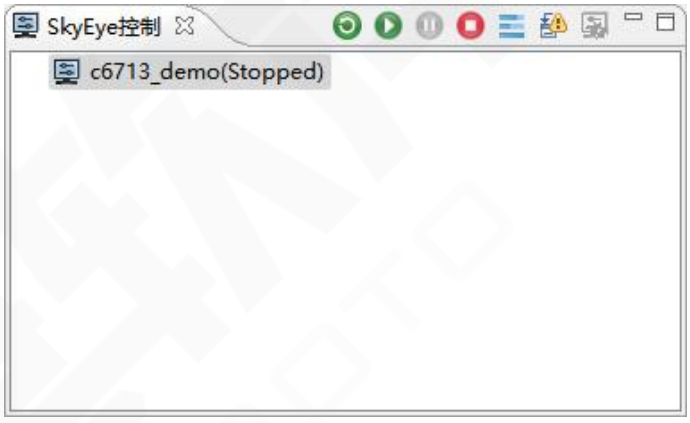


图 5-5 SkyEye 控制

5.2.4 处理器寄存器

处理器寄存器视图用于处理器寄存器信息和处理器指令速度的显示以及处理器寄存器值的修改（在会话暂停的情况下才能修改）。其中，寄存器值改变后将高亮显示，通常配合反汇编视图的单步功能使用。

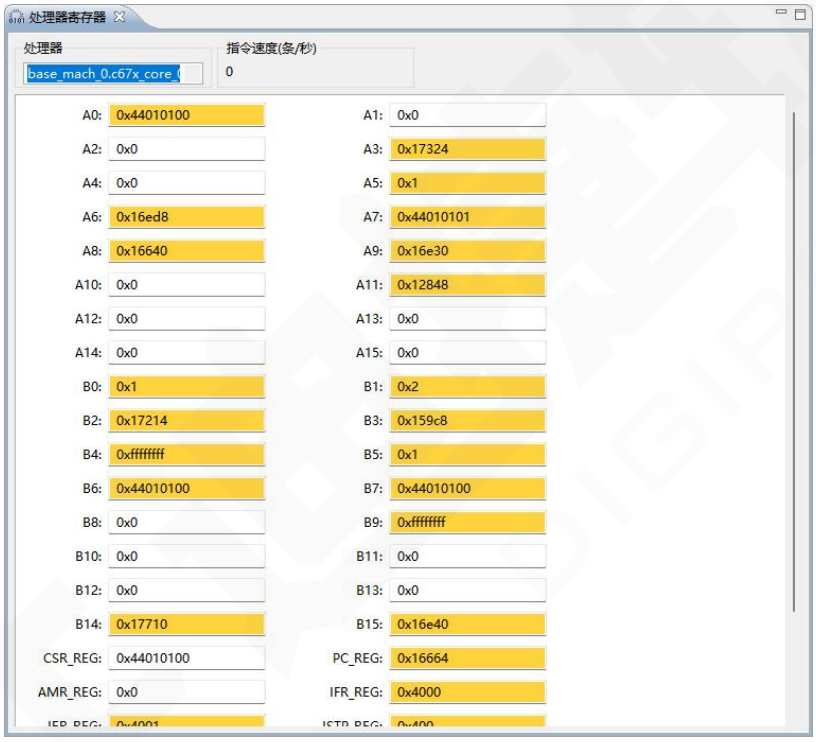


图 5-6 处理器寄存器视图

5.2.5 反汇编查看

反汇编查看视图用于函数名、地址、汇编指令的显示（仅 elf 格式的二进制文件展示函数名），并提供了插入断点并执行和单步执行的功能。

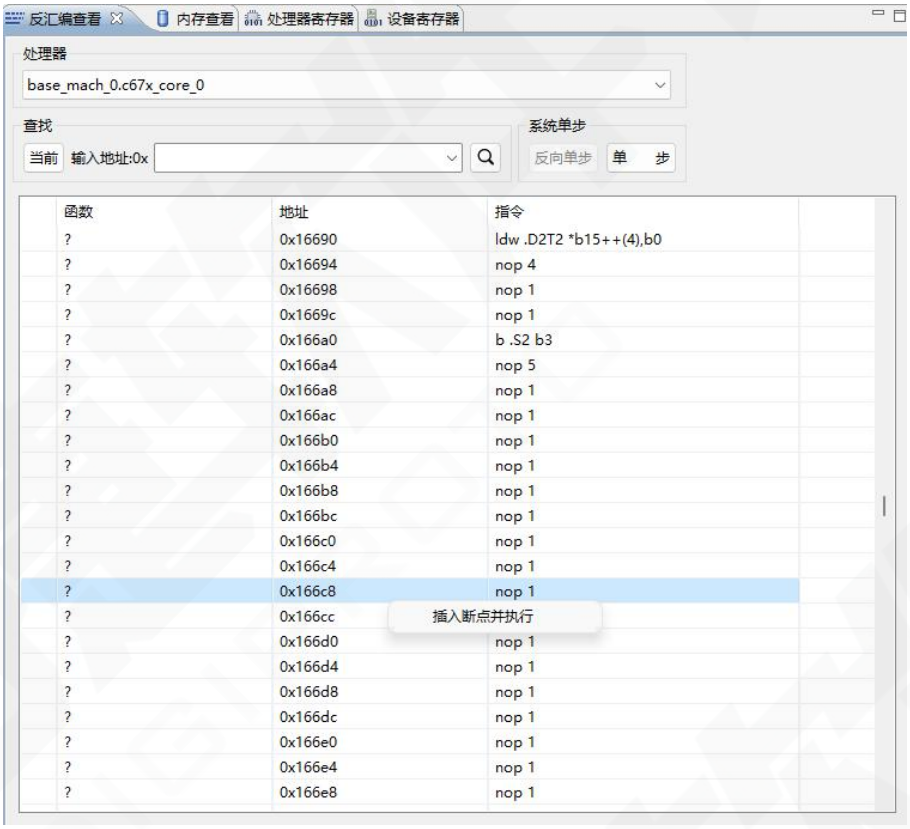


图 5-7 反汇编查看视图

5.2.6 设备寄存器

设备寄存器视图用于设备寄存器值的展示和修改，当值改变时将高亮显示。

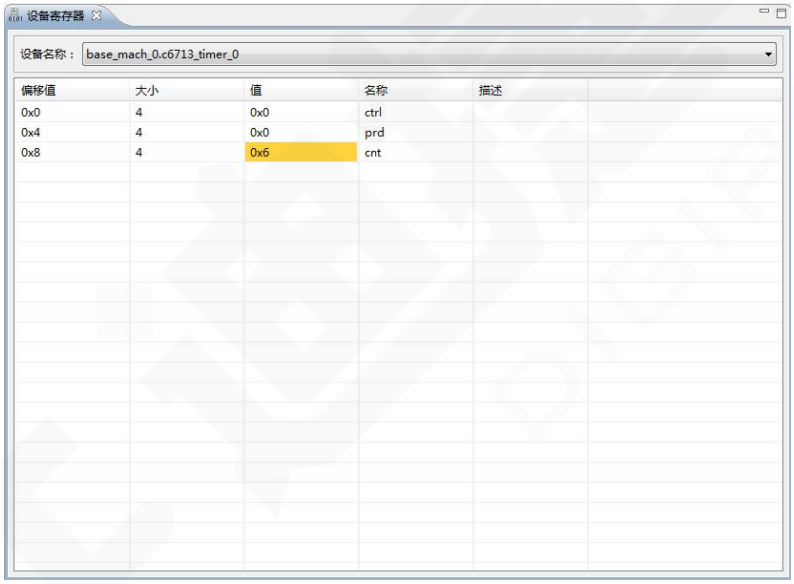


图 5-8 设备寄存器视图

5.2.7 内存查看

内存查看视图用于查看处理器物理内存地址中的值，以十六进制或 ASCII 码显示。设备名称中可选择要查看的处理器，同时可以对地址进行搜索。当会话暂停时，当前地址会高亮显示。

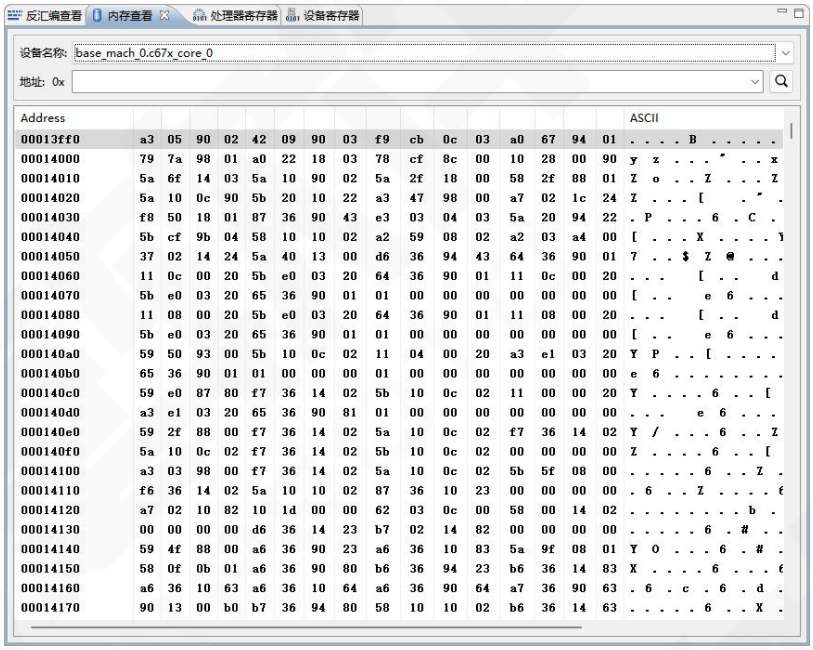


图 5-9 内存查看视图

5.2.8 故障注入

成功运行会话后，可对会话进行故障注入操作（需在暂停状态下才能注入故障，运行状态下，故障注入功能不可用）。其主界面如图 5-10 所示。

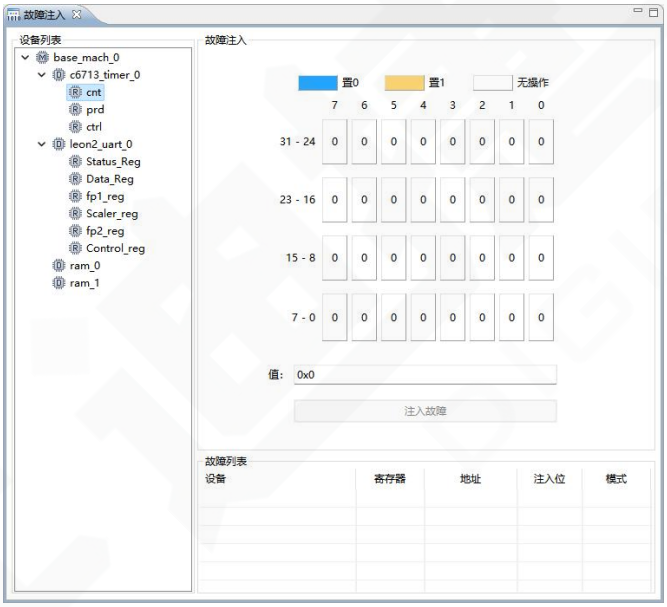


图 5-10 故障注入主界面

故障注入的对象有两种，一种是基于设备地址上的值（点击设备后还需要输入地址），一种是设备内寄存器的值。选择好对象后，右侧会出现该寄存器或设备首地址的值。注入方法也有两种，一种是对注入位进行操作，一种是直接修改值，两种方法之间会有联动，即修改完值后对应有改变的注入位颜色也会改变。最后点击注入故障，故障将记录在故障列表中。

如图 5-11 为向 fp1_Reg 设备寄存器中第 1 位注入置 1 故障。

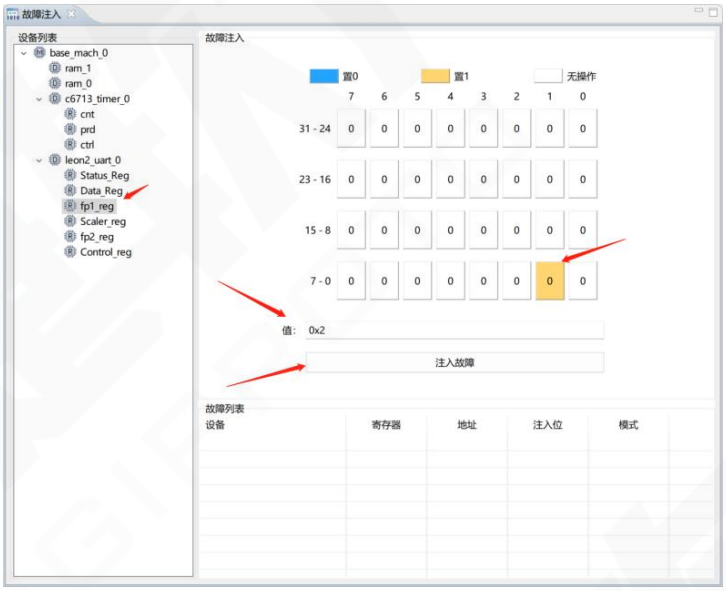


图 5-11 寄存器故障注入

如图 5-12 为往 ram_0 的地址为 0x100 的第 1 位注入置 1 故障。

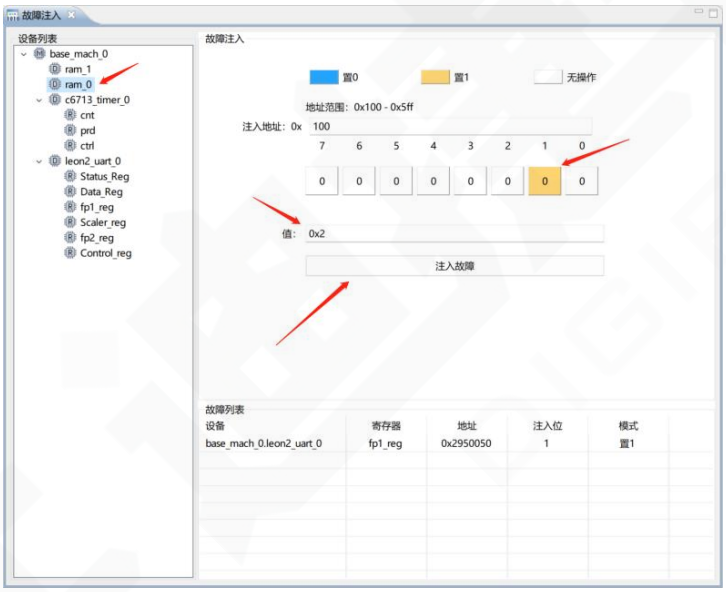


图 5-12 对设备地址故障注入

点击注入故障后，会在故障列表中显示当前已经注入的故障，如图 5-13。

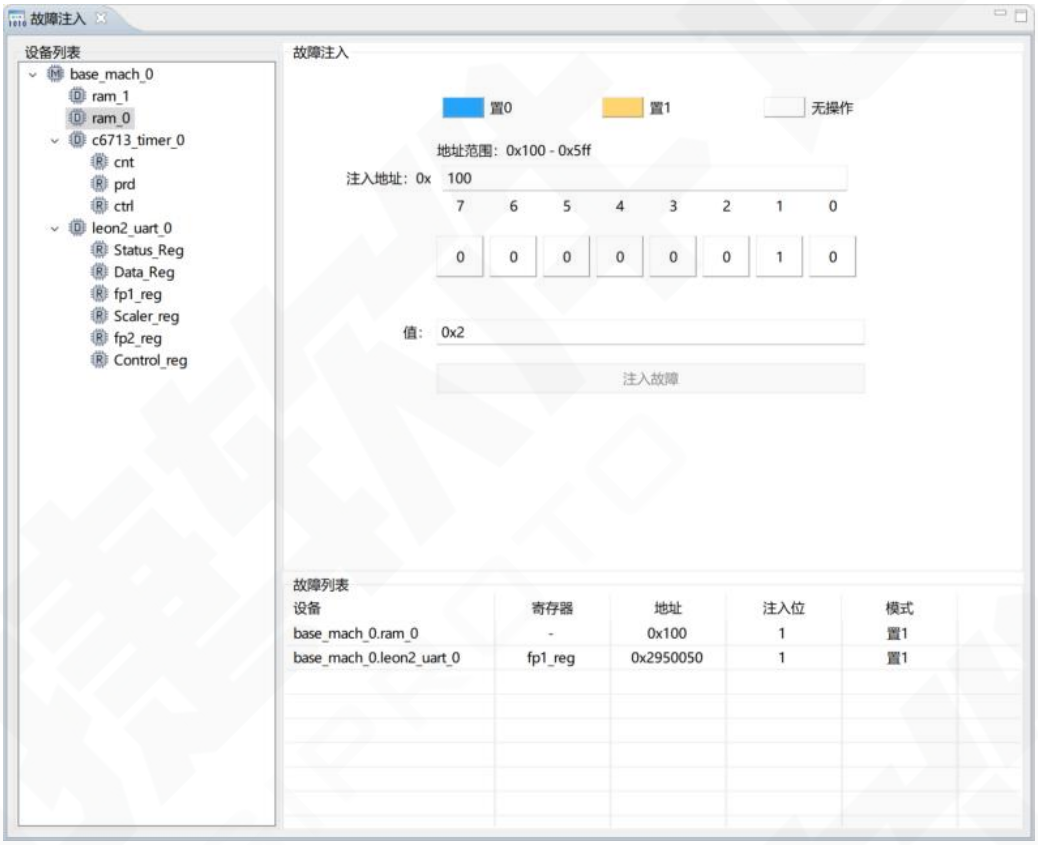


图 5-13 故障注入结果

5.2.9 CPU 性能监控

CPU 性能监控功能调用方法为：1. 菜单栏--【窗口】--【其他】--【建模】--【CPU 性能监控】。初始界面如图 5-14 所示：

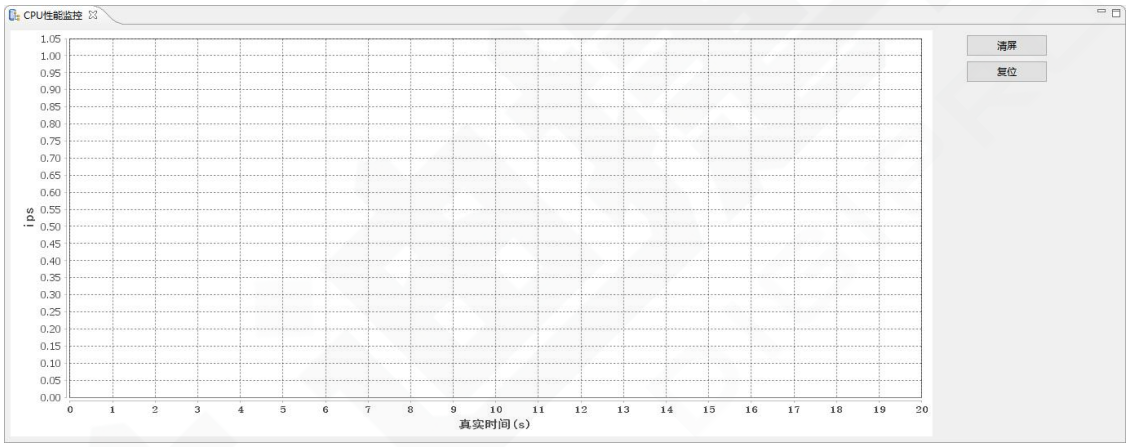


图 5-14 CPU 性能监控

当运行会话配置时，CPU 性能监控会实时监显示当前会话运行时间及其对应指令速度信息。注意，同一会话配置中，不同 cpu 的会有不同的性能监控走向图。

单核会话配置：

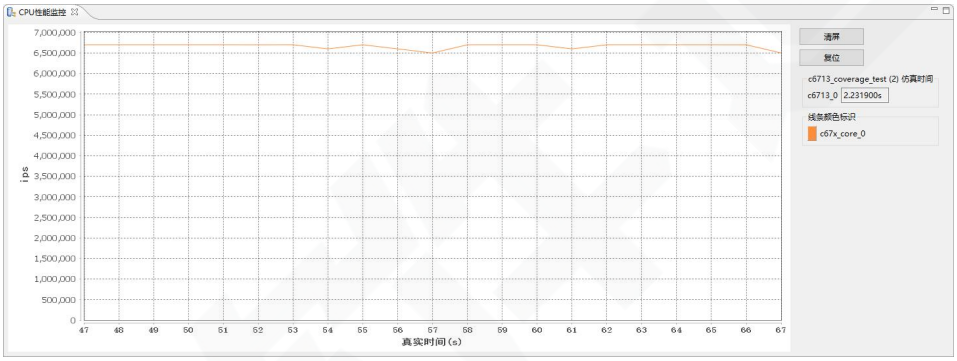


图 5-15 单核会话配置

多核会话配置：

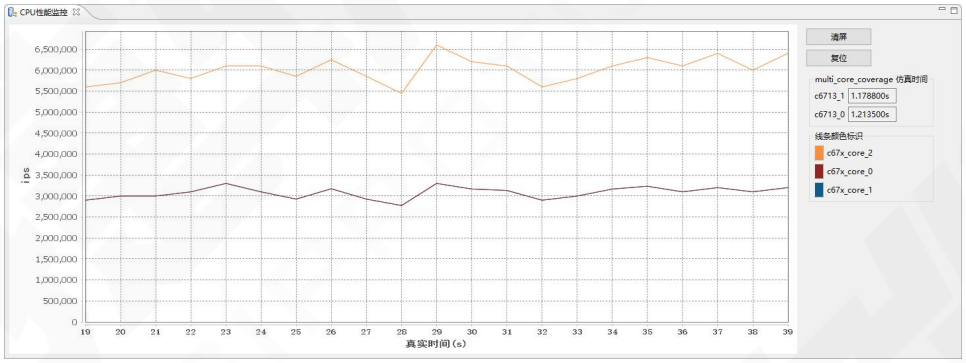


图 5-16 多核会话配置

其他功能：支持在曲线图区域按住鼠标左键，进行左右拖拽查看历史数据。也支持点击清屏（以当前时间为起点显示）与复位功能（回到当前时间）。

5.3 异常触发

成功加载会话后，可对会话进行异常触发操作。点击 SkyEye 控制视图工具栏的异常触发图标，进入异常触发操作界面，如图 5-17 所示。

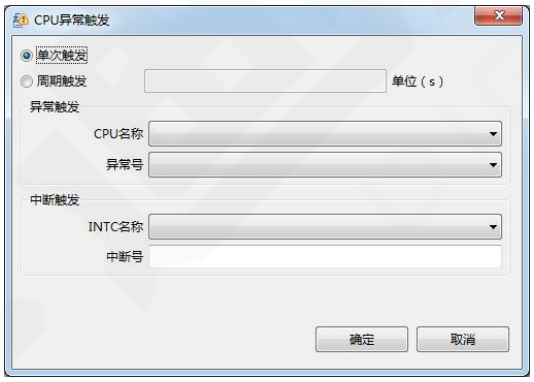


图 5-17 异常触发界面

异常触发的方式有两种，可以单次触发或以秒为单位定时周期触发。选择异常触发的 CPU 名称、异常号，或中断触发的 INTC 名称、中断号后，按确定按钮开始异常触发。若设备处于运行状态，可立即在终端看到结果：

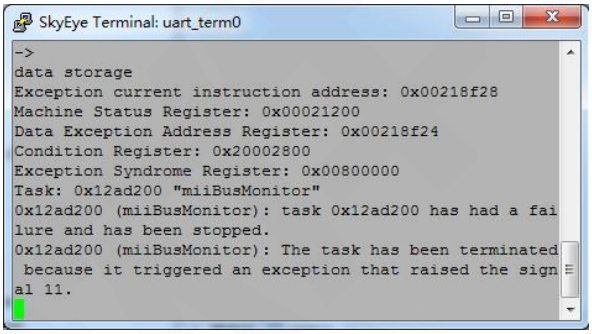


图 5-18 异常触发结果

5.4 远程调试

SkyEye Workstation 支持 SkyEye 应用程序和 C/C++应用程序应用程序的调试，下面主要介绍 SkyEye 应用程序的使用。

SkyEye 支持 GDB 调试，实现了设置断点、单步执行等功能。此外，在使用 GDB 时也可同时查看反汇编、寄存器、内存等辅助调试。

远程调试的配置和运行配置中的 SkyEye 会话相同，可以直接在调试配置 SkyEye 会话增加调试相关属性或直接新建一个 SkyEye 会话。调试配置的调出入口和运行配置类似，分别在主菜单、工具栏和工程右键菜单栏。图 5-19 调试配置界面。后续配置过程参考图 5-20 至图 5-21。

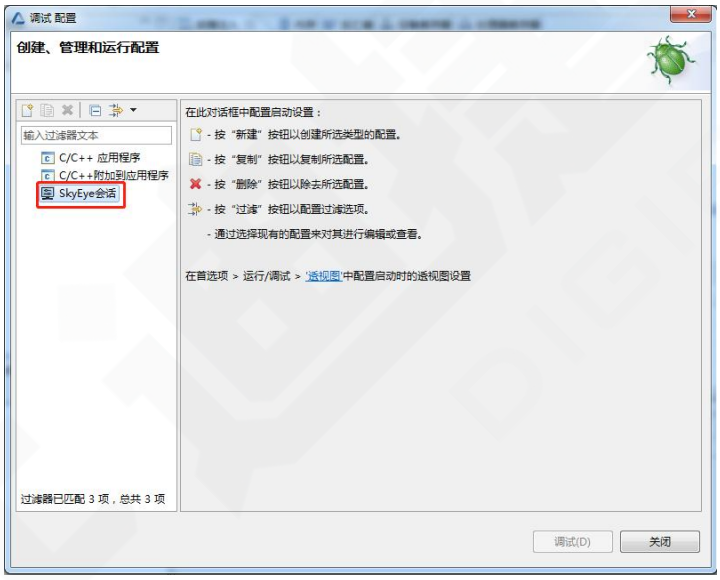


图 5-19 调试配置窗口

如图 5-20 为 SkyEye 会话的调试配置界面，其中包含了工程信息配置界面“SkyEye”、GDB 调试器配置界面“调试器”、“源”、“公共”。

初始时会自动填入所选工程、脚本文件、CPU、C/C++程序信息，也可在下拉框中重新选择脚本文件、CPU 名和 C/C++程序。

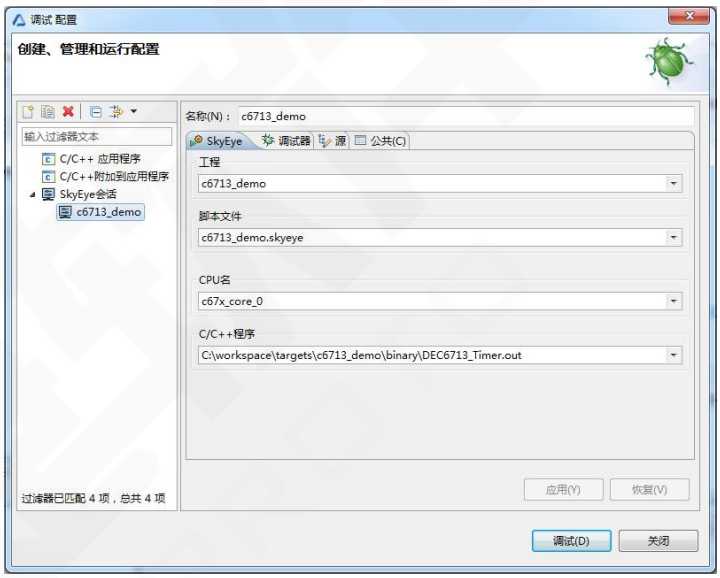


图 5-20GDB 配置界面

SkyEye 工程信息配置完成后，需要继续配置 GDB 调试器。应根据可执行程序选择对应的 GDB 调试工具，本示例工程使用的是加载在 TI C6713 开发板上 coff 格式的二进制程序，故选择 tic6x-coff-gdb（下拉框中自动填充预装的 gdb，如想使用自己的 gdb 工具可以通过浏览进行导入），参考图 5-21。

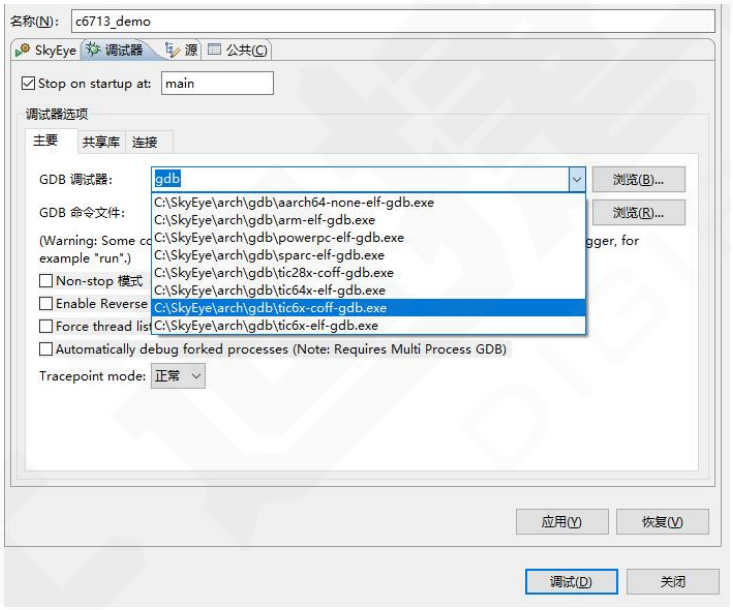


图 5-21 调试器配置页

至此工程的 GDB 调试配置已完成，点击“应用”保存配置信息，点击调试以启动调试功能，打开正常的 GDB 调试界面如图 5-27。

当出现“无法找到源文件（如图 5-22）源码错误”时，如果已经导入源码，按照图 5-23 配置步骤直接选择路径映射即可。而如果未导入源码到工程中，需要按图 5-24 至图 5-26 的步骤选择“缺省值”，手动配置路径映射。

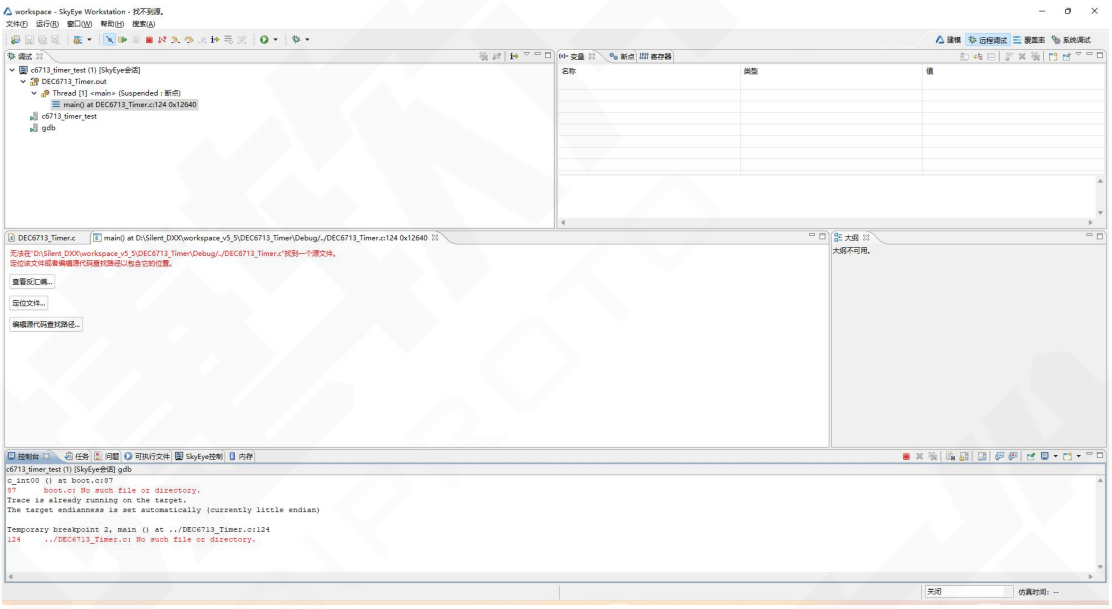


图 5-22 调试找不到源文件

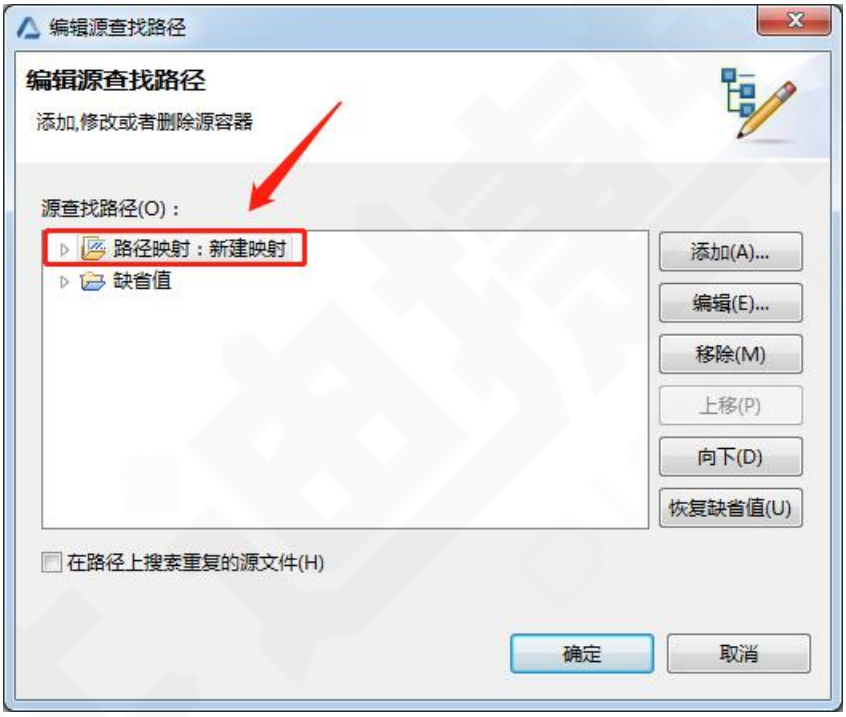


图 5-23 选择路径映射

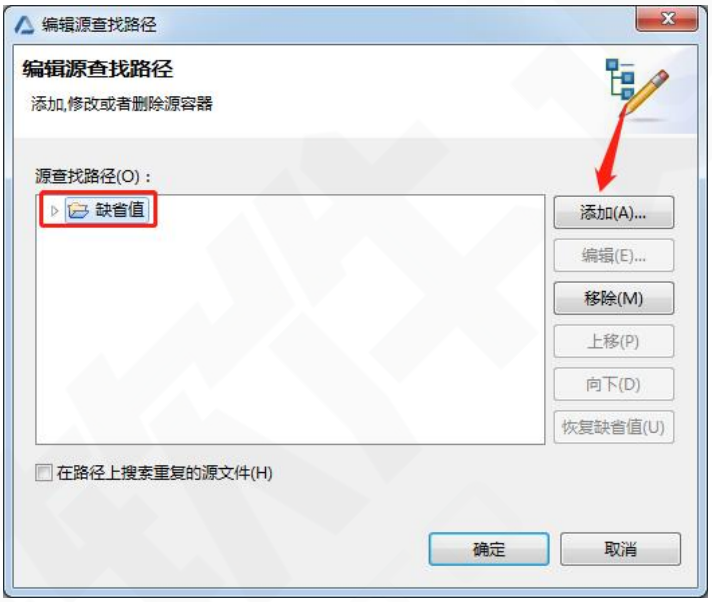


图 5-24 未导入源码手动配置缺省值

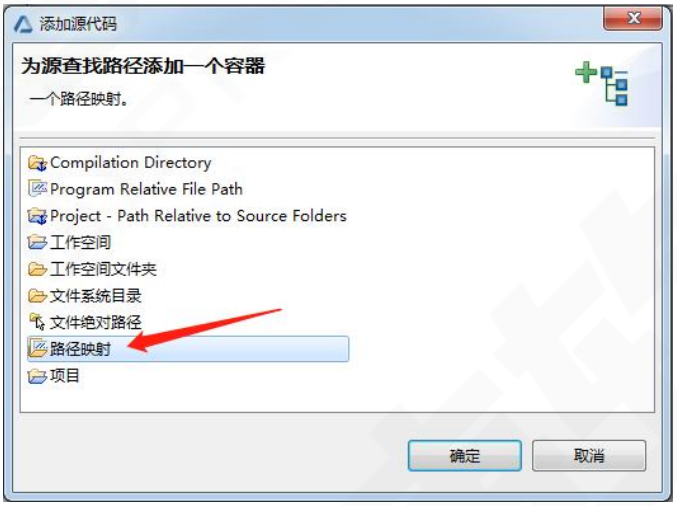


图 5-25 添加一个路径映射

图 5-26 配置路径

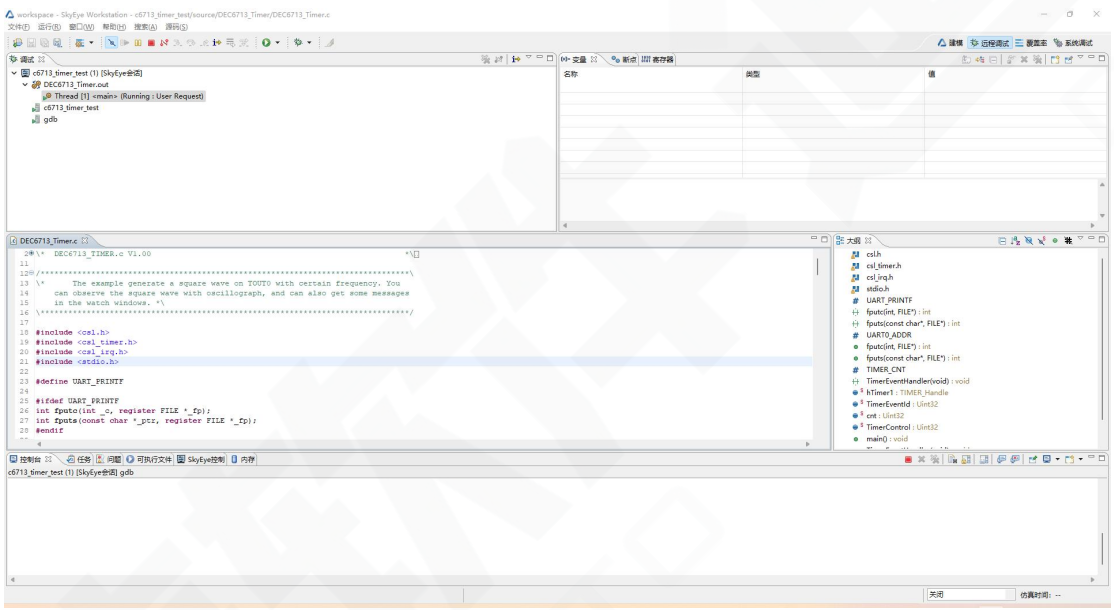


图 5-27 正常调出源文件

在远程调试视图中默认打开的视图有变量、断点、寄存器、表达式、反汇编、大纲、SkyEye 控制、控制台，如图 5-28。如需查看其他信息，可在菜单栏--【窗口】--【显示视图】--【其他】中打开。

图 5-28 调试的其他视图

如图 5-29 是最常用的 debug 按钮，1 表示当前继续执行代码，直到遇到下一个断点，快捷键 F8；2 表示暂停执行；3 表示终止调试；4 表示断开当前远程连接；5 表示进入当前方法内部，一步一步执行，快捷键 F5（step into）；6 表示运行下一行代码，执行当前行，但不进入执行细节，快捷键 F6（step over）；7 表示退出当前方法，返回到调用层，快捷键为 F7（step return）；8 表示进行指令调试。

图 5-29debug 按钮

用户还可以通过 SkyEye 控制视图工具栏中的连接调试  按钮进行调试连接。

5.5 覆盖率统计

代码覆盖率是软件测试中的一种度量标准，在做单元测试时，代码覆盖率常常被拿来作为衡量测试是否全面的指标，某些严格的行业标准规定代码覆盖率必须达到 100%。

SkyEye Workstation 支持目标码和源码的覆盖率统计，并可以导出 html 和 excel 格式的报告。

5.5.1 开启覆盖率开关

目前对于需要进行覆盖率测试的工程，默认未打开覆盖率开关，进行覆盖率测试之前需要配置启动脚本（详情见 4.5.2 章）中的使能覆盖率，具体如图 5-30 所示。覆盖率融合开启后，每次运行都在原覆盖率基础上进行统计。需要注意的是，工程目录下是否有源码文件会影响生成的覆盖率结果；当有源码文件时，只统计源码覆盖率。

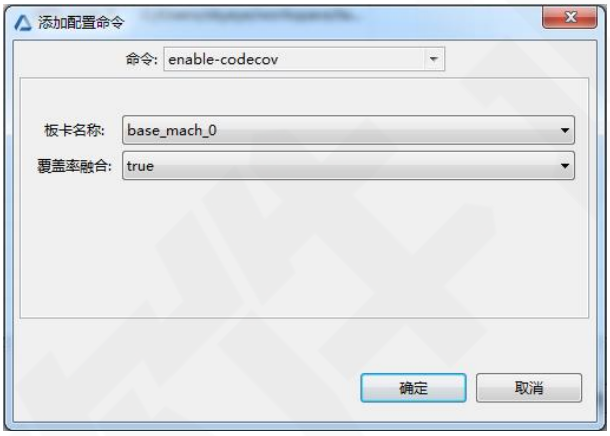


图 5-30 开启覆盖率开关

5.5.2 覆盖率测试

开启覆盖率开关后，运行该工程对应的 SkyEye 会话配置，并运行启动的会话。覆盖率测试需要在暂停情况下进行测试，因此需要在覆盖率测试之前暂停该会话，再点击 。如果存在多 cpu，还需要进行 cpu 的选择。

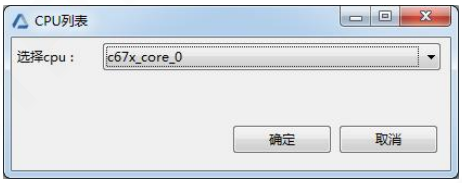


图 5-31 选择覆盖率 cpu

运行结果如图 5-32，将打开反编译文件并在下面显示单个函数覆盖率和总覆盖率。如果导入过源码，还会打开对应源码文件。

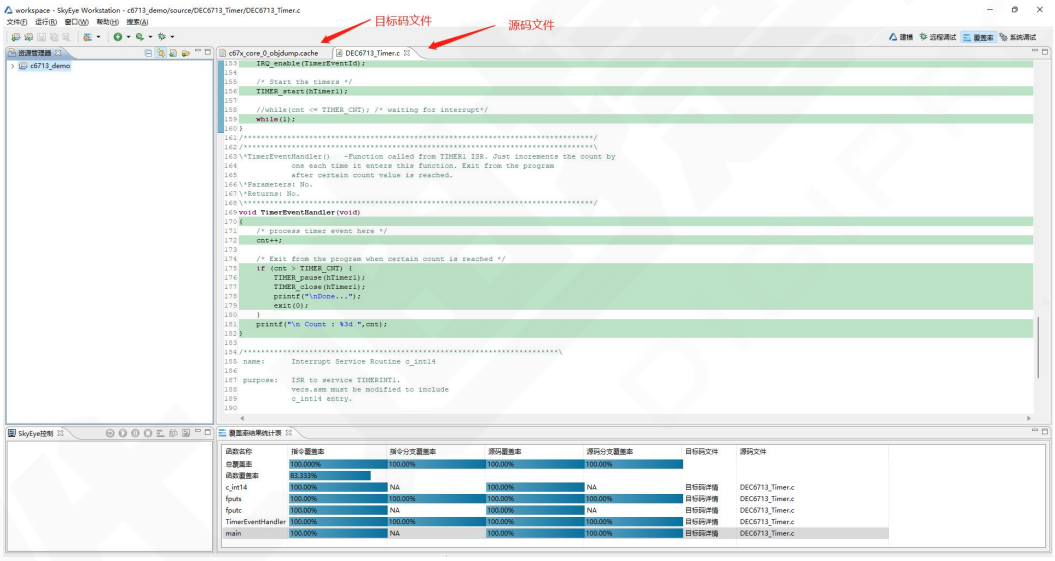


图 5-32 覆盖率运行结果

其中，绿色代表已执行，红色代表未执行，执行结果 true 时为黄色，false 时为紫色。

源码文件和统计表之间还存在联动关系，当鼠标位于某一函数中，覆盖结果统计表中也将自动选中该函数；鼠标点击“覆盖率函数结果统计”中函数名称对应的“目标码文件”或者“源码文件”项，编辑器中会自动显示所点击的函数。如图 5-33 所示。

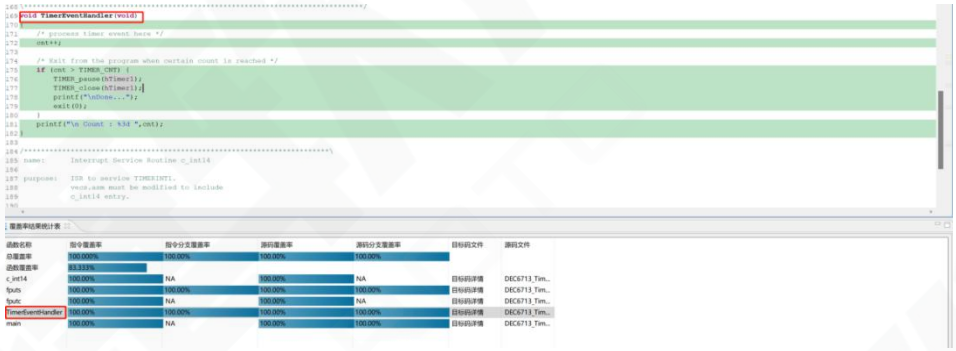


图 5-33 覆盖率源代码跳转显示

5.5.3 覆盖率报告管理

覆盖率计算后可以通过工具栏  中打开覆盖率报告和导出 html 或 excel 格式报告操作。打开覆盖率报告将会把覆盖率结果以网页的形式在浏览器中打开，如图 5-34 所示。注：使用 Windows 自带的 IE 浏览器可能会出现界面排版混乱或功能缺失的情况（解决方案参考 6.6）。

图 5-34 覆盖率报告

5.6 SkyEye 会话组

SkyEye 支持 SkyEye 会话组的使用，即同时启动多个工程，并一组会话同时进行控制。

SkyEye 会话组只支持运行，不支持调试，所以只能在运行配置中进行配置。双击 SkyEye 会话组将自动创建一个 SkyEye 会话组配置，通过右侧工程及对应配置脚本的选择，点击运行即会按会话组配置内容同时加载所选工程。

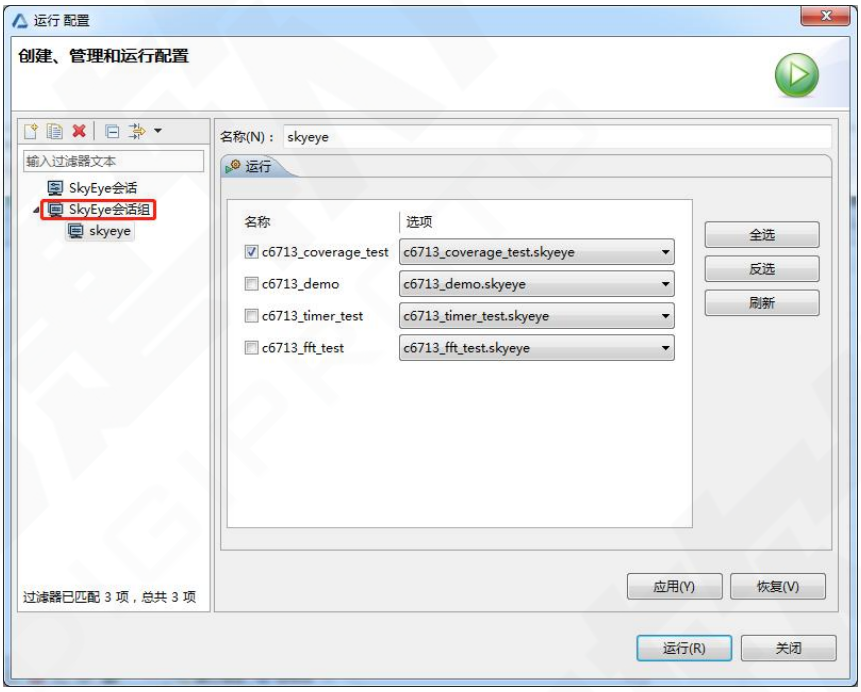


图 5-35SkyEye 会话组配置

5.7 生成反汇编

对于二进制文件，右键菜单中提供生成反汇编功能。如下图选择正确的反汇编工具后，即可生成对应的.s 文件。

图 5-36 生成反汇编

5.8 SkyEye 自动化测试

本章节将简要介绍 SkyEye 自动化测试功能，详细内容请查看文档《SkyEye 自动化测试及接口（Python）使用说明手册》。

SkyEye 自动化测试脚本功能，为用户进行大量重复测试提供便利。可以方便的控制 SkyEye 运行哪些测试用例，执行运行、暂停、重启，结果比对等功能，在测试中还可收集如最高运行速度、设备内存泄漏等信息。

SkyEye 自动化测试功能不以命令方式或界面方式手动启动测试工程，而是通过/c/SkyEye/opt/skyeye/bin/skyeye -a 的方式启动单个自动化测试脚本，或通过 python autotest_skyeye.py 批量进行自动化测试（详见《SkyEye 自动化测试及接口（Python）使用说明手册》）。其主要作用是快速比对期望与实际运行结果。

下面以执行单个自动化测试工程为例进行简要介绍：

执行自动化测试时，SkyEye 会自动查找当前目录下名称为 skyeye_test.py 的自动化测试脚本文件，并根据文件中内容进行运行控制。

自动化测试脚本示例框架代码如下：

```
#coding:utf-8
import time
import os
import sys
from skyeye_autotest_command import *

class test():
def __init__(self):
        testname="c6k.skyeye"
        test_branch='new_framework'
        enable=True
        run_system="all"
        self.ac=autotest_command(testname,enable,run_system,test_branch)

def dev_init(self):
```

```
        self.ac.add_file_out_dev('core_0_uart_0','log.txt')


def test_flow(self):
        self.ac.run_script()
        self.dev_init()
        self.ac.run(30)
        self.ac.stop()
        self.ac.compare("log.txt",self.expect()[0],100)
        self.ac.reset()
        self.ac.output_result()
def test_main(self):
        self.test_flow()
def expect(self):
        expected_output=['''
hello


 Count :    1
 Count :    2
''']
        return expected_output
```

init 函数：testname 是测试用例中引导文件名称。test_branch 为运行的源码分支，不需要修改。enable 是测试用例测试开关。run_system 是测试用例测试的系统，all：Linux 和 Windows 系统都测试，linux 表示测试用例只测试 Linux 系统，windows 表示测试用例只测试 Windows 系统。

dev_init 函数：add_file_out_dev('core_0_uart_0','log.txt')，添加 uart_file 设备，将 uart 中显示的数据输出到文件中，便于后面结果输出对比，core_0_uart_0 表示要将 uart_file 设备连接到的 uart 设备，log.txt 表示输出的文件名称。

test_flow 函数：run_script()加载引导脚本。run(30)运行测试用例，30 表示运

行 30 秒钟。stop()暂停测试用例。compare("log.txt",self.expect()[0],100)对比输出结果，log.txt 表示实际 uart 输出文件名称，self.expect()[0]表示期望输出值，100 表示实际输出和期望输出的对比百分比，只有到达设置的百分比才算对比一致。reset()重置 SkyEye。output_result()输出自动化测试结果。

expect 函数：expected_output 是存储期望输出的变量。

SkyEye 自动化测试功能一般是通过 SkyEye 命令行工具来运行的。在运行 SkyEye 命令行模式时加入-a 选项就可以自动加载当前目录下测试用例中的自动化测试脚本，完成自动化测试。

启动 SkyEye 命令行工具后，切换路径至测试用例目录，运行命令 /c/SkyEye/opt/skyeye/bin/skyeye -a 或者 skyeye -a。

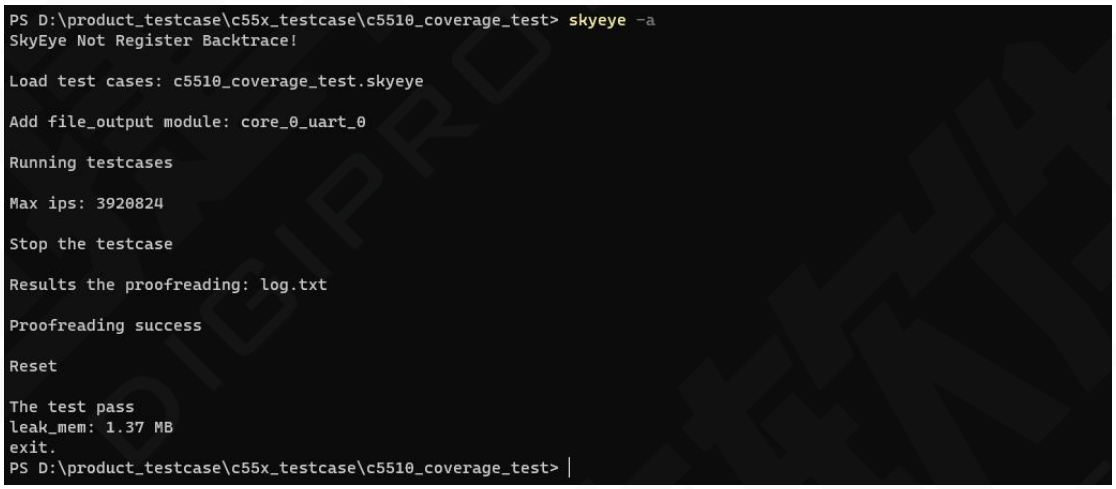


图 5-37 自动化测试

5.9 SkyEye 接口说明

为方便用户在测试用例运行过程中进行数据采集、故障注入、监视地址、监视 PC 等操作，SkyEye 提供了 Python 语言接口，可以使用 Python 语言编写脚本，完成测试场景部署。详细内容可见《SkyEye 自动化测试及接口（Python）使用说明手册》。

SkyEye 接口功能需要通过在 SkyEye 引导文件.skyeye 中添加引导命令 run-pyfile xxx.py 后，使用命令行或界面方式启动 SkyEye，从而实现用户编写的接口脚本注入。此功能可以获取更多的调试与运行信息。

运行 SkyEye 测试用例时，在引导文件(.skyeye)中加入运行脚本命令

run-pyfile xxxx.py，就可以在运行测试用例的同时加载 Python 脚本。

如在引导文件中配置了加载 Python 文件命令，在运行测试用例时会自动加载相应 Python 脚本文件。须在 Python 脚本文件中导入 se、se_system 两个库。如下示例中，创建了一个线程，线程会运行 func 函数，func 函数中对内存值进行了读取和写入。

```
import se
import se_system as ss
import threading
import time
from ctypes import *
class thread(threading.Thread):
        def __init__(self,func):
            threading.Thread.__init__(self)
            self.func=func
            self.setDaemon(True)
            self.ThreadStop=False
        def run(self):
            while self.ThreadStop == False:
                self.func()
                self.ThreadStop = True

        def stop(self):
            self.ThreadStop=True
def func():
        ####memory
        se.SE_print("read
byte: %x"%se.SE_read_byte("c67x_core_0",0x14aac))
        se.SE_write_byte("c67x_core_0",0x14aac,0x88)
        se.SE_print("read
```

```
byte: %x"%se.SE_read_byte("c67x_core_0",0x14aac))
                se.SE_print("read
dword: %x"%se.SE_read_dword("c67x_core_0",0x14aac))
                se.SE_write_dword("c67x_core_0",0x14aac,0x180)
                se.SE_print("read
dword: %x"%se.SE_read_dword("c67x_core_0",0x14aac))


    th=thread(func)
    th.start()
```

在引导文件中 init-ok 后加入 run-pyfile 命令。使用 SkyEye 命令工具打开 SkyEye 命令行模式，加载测试用例。

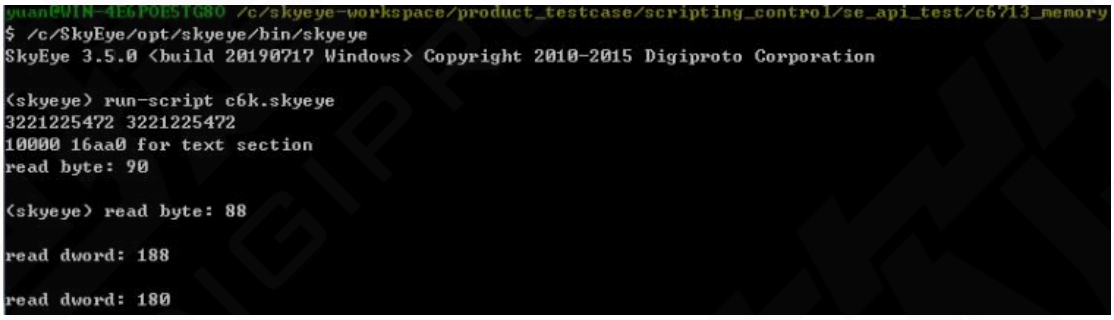


图 5-38 加载测试用例

常用的接口如下表。

Python 自动化测试接口列表	
模拟器控制	
SE_define_conf	加载 JSON 文件
SE_load_binary	加载二进制文件
SE_init_ok	初始化硬件环境
SE_run_script	加载引导脚本
SE_run	运行模拟器
SE_stop	停止模拟器
SE_stop_autotest	停止自动化测试
SE_reset	重置模拟器
SE_restart	重启模拟器
SE_running_status	获取模拟器运行状态
SE_run_to_time	运行到指定时间
SE_create_breakpoint	创建断点
故障注入	
SE_set_fault_inject	设置故障

SE_get_fault_inject	获取故障列表
SE_clear_fault_injece	删除故障
获取模拟系统信息	
SE_get_cpu_freq	获取 CPU 频率
SE_get_current_pc	获取当前 PC 地址
SE_get_simulation_time	获取当前模拟时间
SE_set_register_value	写入寄存器值
SE_get_register_value	读取寄存器值
内存操作	
SE_read_byte	从内存中读取字节
SE_read_dword	从内存中读取双字
SE_write_byte	向内存中写字节
SE_write_dword	向内存中写双字
全局变量操作	
SE_enable_parse_symbol	打开符号解析
SE_get_global_variable_addr	获取全局变量地址
SE_get_global_variable_value	获取全局变量值
SE_set_global_variable_value	修改全局变量值
SE_get_func_addr	获取函数地址
SE_get_func_length	获取函数长度
回调函数	
SE_sync_callback	设置同步回调函数
SE_create_timer	设置定时器回调函数
SE_set_watch_on_pc	设置执行监视回调函数
SE_set_watch_on_mem	设置内存监视回调函数
SE_delete_timer	删除定时器
SE_del_watch_on_pc	删除执行监视
SE_del_watch_on_mem	删除内存监视回调函数
其他	
SE_sleep	真实时间延时
SE_print	打印
SE_compare	比较输出结果
SE_out_error_log	输出错误到运行控制层
SE_log_output	输出到日志文件

区别：SkyEye 接口功能和 SkyEye 自动化测试功能是两个不同层级的关系，自动化测试功能可以理解为 SkyEye 外层脚本，可以控制整个 SkyEye 的加载运行停止。接口功能可以理解为 SkyEye 内层脚本，可以获取 SkyEye 运行过程中的信息。需要注意的是，这是两种不同的脚本文件，前一种功能使用的是自动化测试脚本，后一种功能使用的是接口脚本。

6 错误与恢复

6.1 工作空间被占用

打开软件，发现以下弹窗，如果当前计算机任务栏中没有 SkyEye Workstation 栏，可能是上次软件异常，在后台有残留。

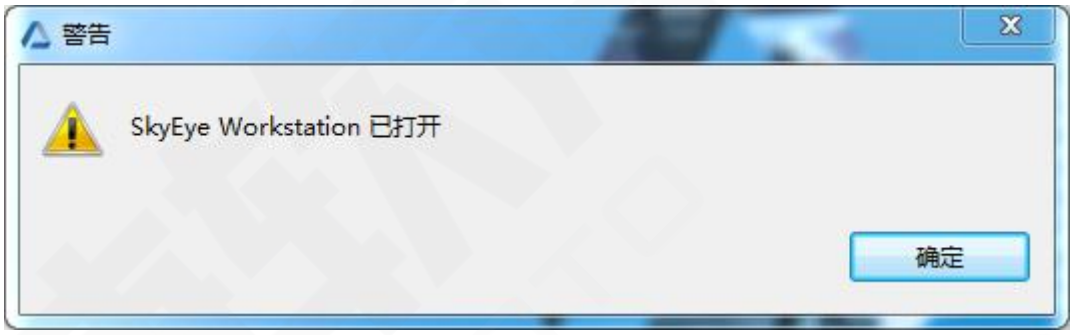


图 6-1 工作空间被占用

解决方案：调出任务管理器强行结束 SkyEye Workstation 应用。

6.2 未获取到环境变量

打开软件，如弹出“未获取到 SKYEYEBIN 环境变量，请检查后重试。”如果是安装后首次启动报错，那可能环境变量设置有延后效果，可以尝试重新启动软件或重新启动电脑后；如果之前启动正常，突然启动失败。可能是因为电脑的环境变量被误删了。

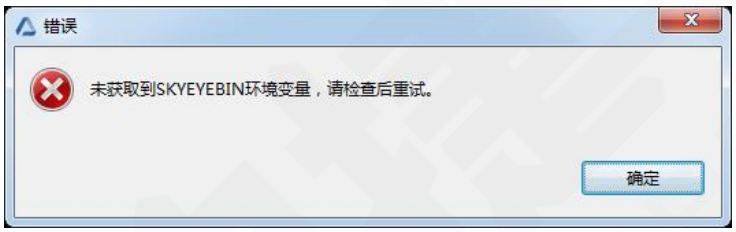


图 6-2 未获取到环境变量

解决方案：

方案一：手动添加该环境变量，点击我的电脑右键->属性打开系统页面，win7 点击更改设置，win10 点击高级系统设置，如下图。

图 6-3Win7 更改设置

图 6-4Win10 更改设置

点击高级页中的环境变量。

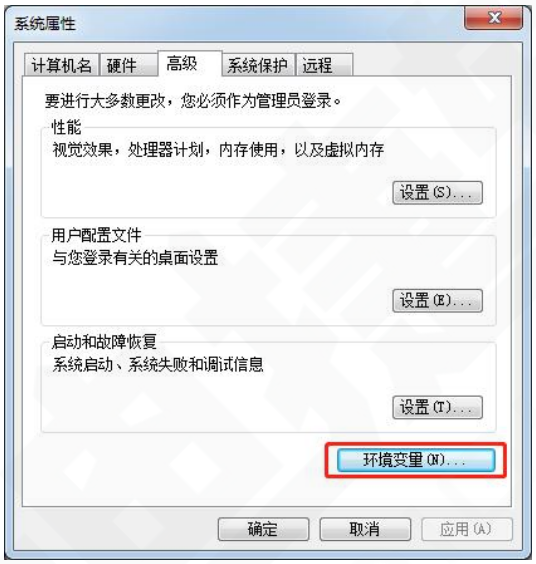


图 6-5 系统属性

新建一个变量名为 SKYEYEBIN（用户变量或系统变量都可以），变量值为安装目录下/opt/skyeye/bin 例如 C:\SkyEye\opt\skyeye\bin，点击确定，最好重启电脑后再打开软件。

图 6-6 新建系统变量

方案二：用户可以直接通过安装包进行修复，双击安装包后点击修复，修复完成后最好重启电脑后再打开软件。

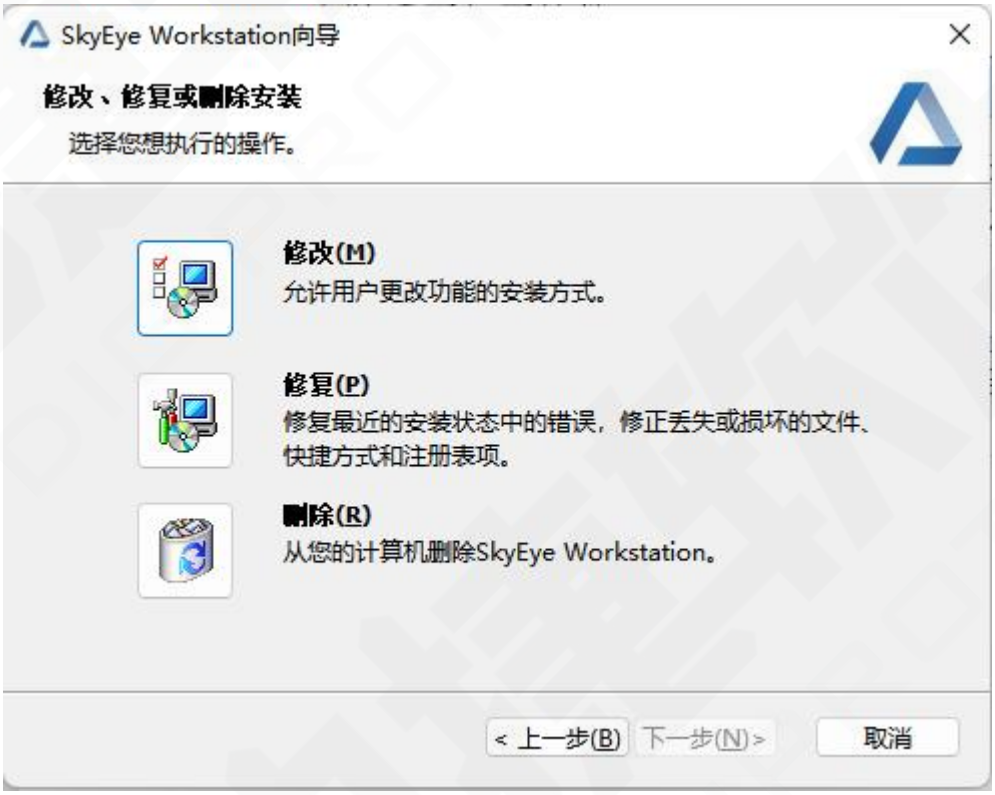


图 6-7 通过安装包修复

6.3 软件启动后卡在初始化核心组件

SkyEye Workstation 软件经过初始化核心组件后闪退，如图 6-8 所示。该情况很有可能是丢失运行库，导致启动失败。

图 6-8 初始化核心组件

解决方案：可以打开天目.exe，确定缺少的 dll，如图 6-9 所示。当确定是由于缺少 dll 导致 SkyEye Workstation 界面无法启动时，可以选择重装 SkyEye 产品，也可以联系软件提供商修复运行环境。

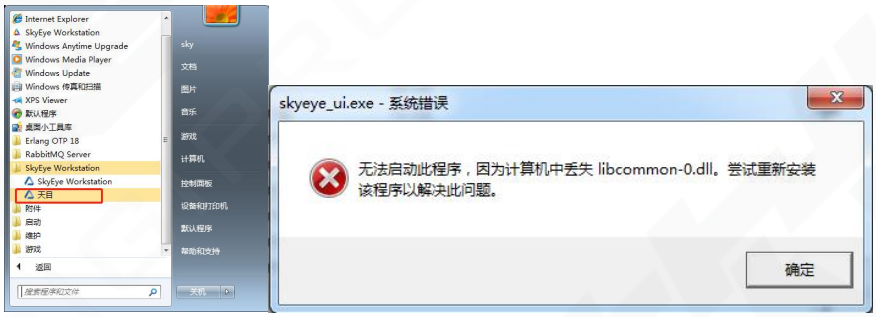


图 6-9 确定缺少 dll

6.4 run_script 错误

下面是一些工程加载时的常见报错。

6.4.1 实例对应的类不存在

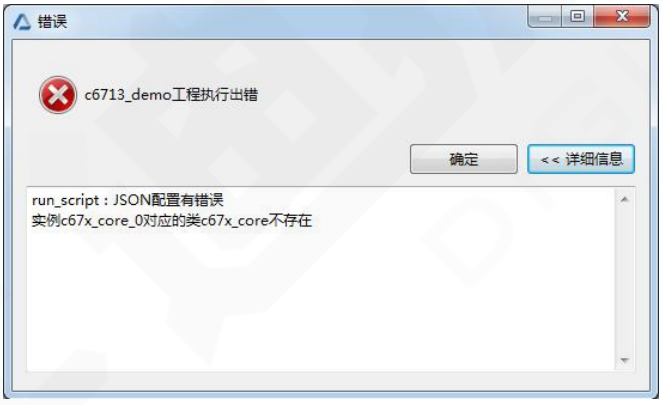


图 6-10 实例对应的类不存在

原因是没有找到 c67x_core 的库文件。

解决方案：放置库文件到正确位置，安装目录为\opt\skyeye\lib\skyeye，也可以联系软件提供商修复运行环境。

6.4.2 设备创建失败

图 6-11 设备创建失败

解决方案：首先检查一下授权信息，菜单栏帮助 -> License 验证 -> 许可核心，如图 6-12 所示。如果需要增加许可核心，请联系迪捷软件工程师。

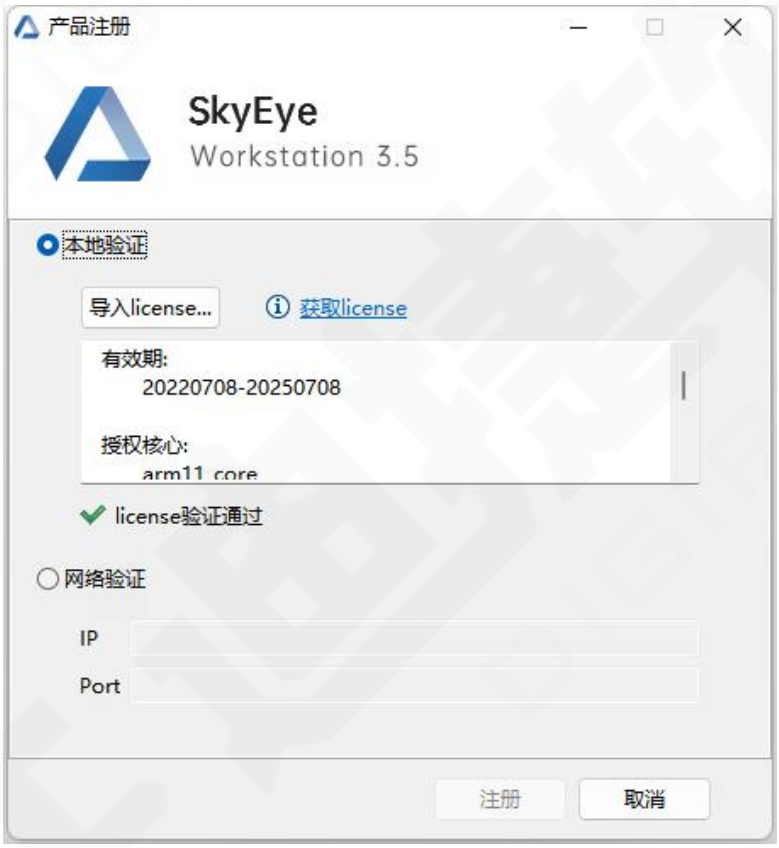


图 6-12 检查授权信息

6.4.3 没有找到与 xx 设备的接口 xx 连接的设备

如图 6-13 所示，提示找不到对应的设备。该问题很可能是手动误修改了 json 文件所致。

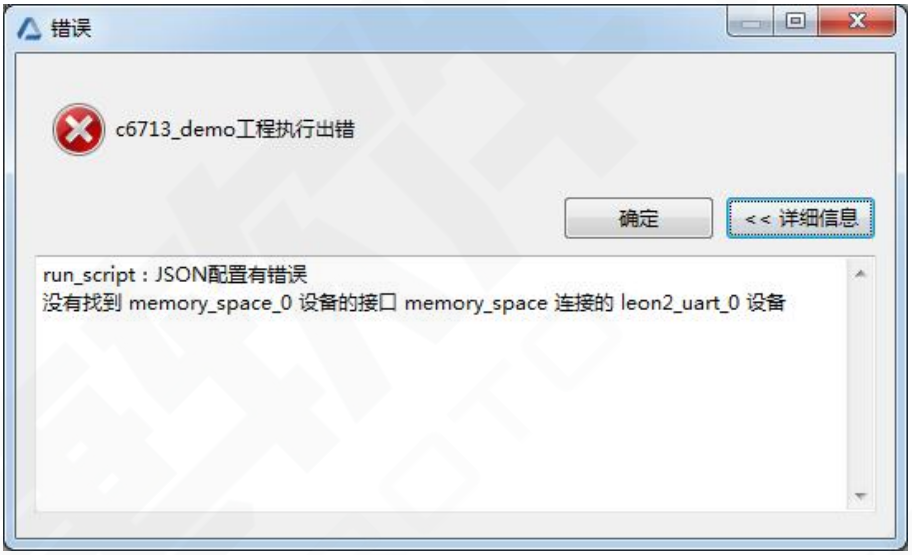


图 6-13 没有找到与 xx 设备的接口 xx 连接的设备

解决方案：如果该 json 有对应 gp 文件，检查 gp 文件无误后保存（随意移动一下设备再保存，才能确保触发了保存），此时将自动更新 json 文件。

6.4.4 加载二进制文件失败

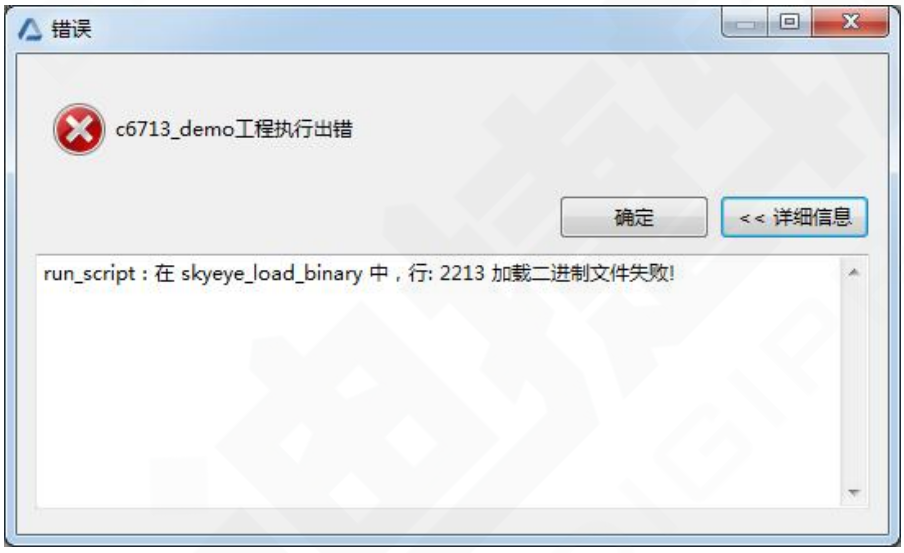


图 6-14 加载二进制文件失败

解决方案：检查是否为该二进制程序正确分配内存空间。

6.5 覆盖率报错

运行时间太短导致执行覆盖率时出现如图 6-15 的错误。

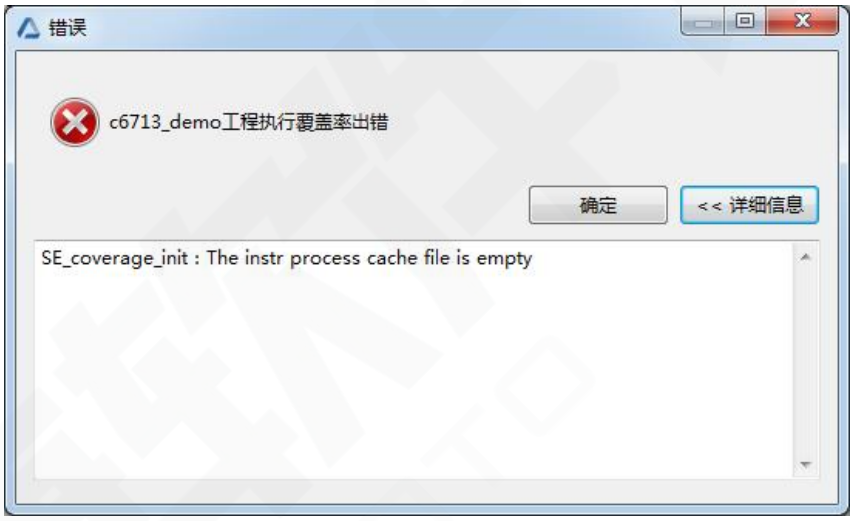


图 6-15 覆盖率报错

解决方案：增加工程的运行时间后再作尝试。

6.6 覆盖率 html 文件无法跳转

用 win7 自带 IE 等其他低版本浏览器时，默认会限制 js 文件的调用，导致功能缺失。下面方案中推荐方案三，方案一二只能解决功能缺失问题，排版可能还会存在混乱的情况。

解决方案：

方案一：右键提示->允许阻止的内容。该方案只能生效一次，下次打开还是需要重新允许阻止的内容。

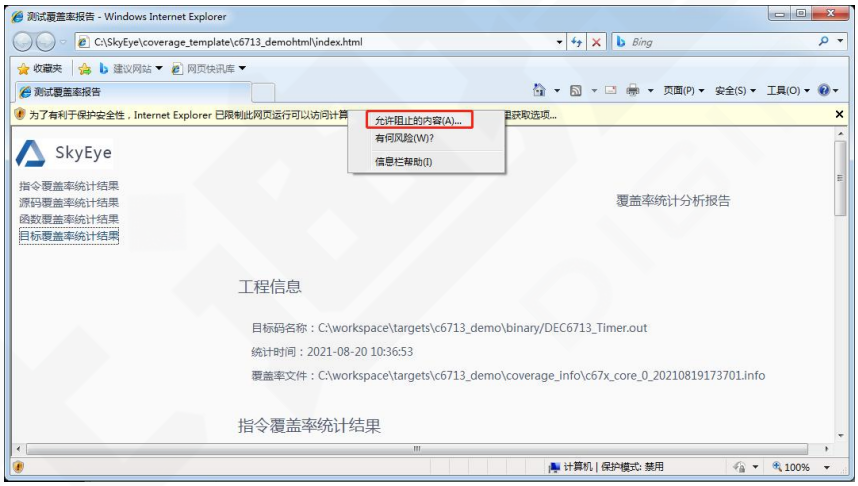


图 6-16 允许该次 js 调用

方案二：打开 IE->工具->Internet 选项->高级标签->安全->勾选“允许活动内容在我的计算机上的文件中运行”，最后重新打开 html 文件将不再限制 js 文件的调用。

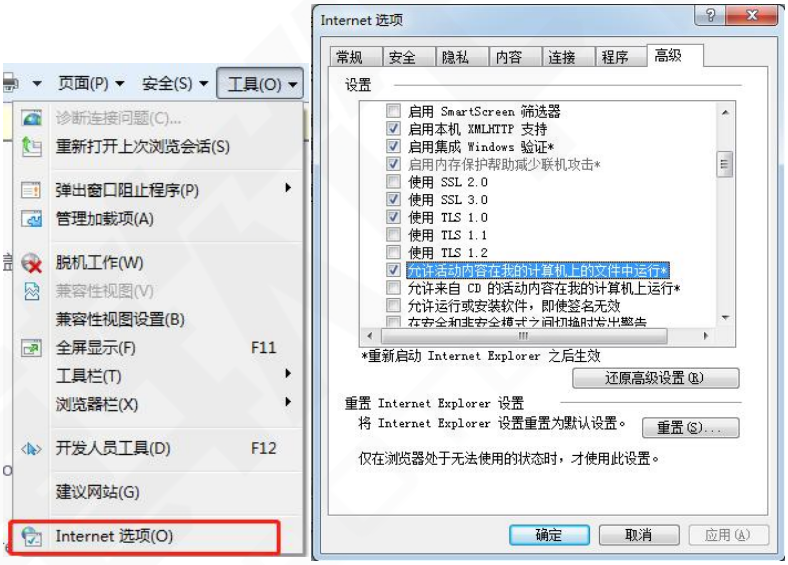
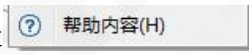


图 6-17 设置 IE 浏览器允许 js 调用

方案三：

更换谷歌等高版本浏览器。

6.7 帮助界面

在使用过程中遇到问题，可以从帮助菜单栏下的帮助内容 

中寻找解决方法，如图 6-18 所示。

图 6-18 帮助界面

6.8 其他

遇到其他错误无法解决时，请保存好相关环境，其中包括 SkyEye Workstation 界面日志（软件安装目录/eclipse/configuration、软件安装目录/eclipse/log 和工作空间目录/.metadata 中的 log 文件）、SkyEye 核心运行日志（软件安装目录/opt/skyeye/bin/的 log 文件和 rabbitmq_log.csv 文件）等，并反馈给软件供应商。

7 SkyEye 脚本命令

本章节将简要介绍 SkyEye 启动脚本文件中各命令。

7.1 定义目标系统

define_conf

命令：

define_conf <filename>

参数:

名称	说明
filename	硬件配置文件（*.json）文件名称

描述：

加载一个 SkyEye 硬件配置文件（*.json），该文件中定义 SoC，cpu，device 等。

7.2 加载二进制文件

load-binary

命令：

load-binary <cpuname> <filepath>

参数：

名称	说明
cpuname	加载二进制文件的 CPU 的名字
filepath	二进制文件的路径，支持 elf/coff/hex 格式二进制文件

描述：

对于包含符号信息、调试信息等内容的二进制文件直接使用 load-binary 加载即可。

load-file

命令：

load-file <cpuname> <filepath> <address>

参数：

名称	说明
cpuname	加载二进制文件的 CPU 的名字
filepath	二进制文件的路径
address	加载地址

描述：

对于一些可执行文件需要从指定地址开始加载，使用 load-file 指定其加载地址。

load-bin-binary

命令：

load-bin-binary <cpuname> <filepath> <address> <length> <start_pc>

参数：

名称	说明
cpuname	加载二进制文件的 CPU 的名字
filepath	二进制文件的路径
address	加载地址
length	二进制文件大小
start_pc	pc 起始地址

描述：

对于一些 bin 文件需要指定其加载的加载地址、文件大小和 PC 起始地址，使用 load-bin-binary 进行加载。

load-aout-binary

命令：

load-aout-binary <cpuname> <filepath> <address> <start_pc>

参数：

名称	说明
cpuname	加载二进制文件的 CPU 的名字
filepath	二进制文件的路径
address	加载地址
start_pc	pc 起始地址

描述：

对于一些 aout 文件需要指定地址的内存中首条 PC 地址，使用 load-aout-binary 进行加载。

load_binary_x86

命令：

load_binary_x86 <cpuname> <filename>

参数：

名称	说明
cpuname	加载二进制文件的 CPU 的名字
filename	二进制文件的路径

描述：

对于需要使用x86 架构的覆盖率功能，需使用该命令加载执行的二进制文件。

load_app_file

命令：

load_app_file <cpuname> <filename> <address>

参数：

名称	说明
cpuname	加载二进制文件的 CPU 的名字
filename	二进制文件的路径
address	二进制文件的偏移地址

描述：

对于只需要统计某个应用程序覆盖率的情况，使用该命令加载应用程序。

7.3 符号解析

parse-symbol

命令：

parse-symbol <cpuname> <filepath>

参数：

名称	说明
cpuname	符号解析的 CPU 的名字
filepath	二进制文件的绝对路径

描述：

为了通过对二进制文件进行符号解析，获取软件中全局变量地址信息，可以对全局变量进行数据注入和输出监视。

7.4 设置同步周期

set-min-syntime

命令：

set-min-syntime <time> 1

参数：

名称	说明
time	同步周期

描述：

使两个目标系统可以按设定的同步周期同步执行。注：实际使用后面还有一个默认参数 1，如设置同步周期为 0.01 时命令为“set-min-syntime 0.01 1”。

7.5 使能覆盖率

enable-codecov

命令：

enable-codecov <machinename> <ismerge>

参数：

名称	说明
machinename	板子名称
ismerge	是否使能覆盖率融合

描述：

使能相应板卡的覆盖率统计功能，开启后将进行覆盖率统计。使能覆盖率融合后，每次运行工程后，都会将之前的覆盖率信息进行合并；若不使能，则每次运行将会重置覆盖率。

7.6 远程连接 GDB

remote-gdb

命令：

remote-gdb <cpuname> <port> <ip>

参数：

名称	说明
cpuname	需要调试的 cpu 名称
port	端口号
ip	主机 IP 地址

描述：

配置远程连接 GDB 服务器，可进行 GDB 调试。

7.7 初始化

init-ok

命令：

init-ok

参数：

无

描述：

初始化所有的硬件环境，准备运行。

7.8 执行运行控制脚本

run-pyfile

命令：

run-pyfile <filename>

参数：

名称	说明
filename	Python 脚本文件名

描述：

执行 python 脚本。

7.9 开启设备 debug 功能

enable-skyeye-debug

命令：

enable-skyeye-debug <soc_name>

参数：

名称	说明
soc_name	打开 debug 的板卡名称

描述：

打开设备调试模式，可以查看二次开发外设时的一些调试信息。

8 联系方式

浙江迪捷软件科技有限公司

迪捷软件 DIGIPROTO

Zhejiang Digiproto Software Technology Co., Ltd

公司地址：浙江省绍兴市越城区中关村水木湾区科学园

北京市海淀区永丰路 9 号院

上海市松江区九亭镇九亭中心路 1158 号

中国（四川）自由贸易试验区成都高新区天府三街 288 号

西安市高新区天谷七路 996 号

固定电话：0575-88361699

移动电话：19957518153（Sales）/13260299730（合作）/13501153049

联系邮箱：contact@digiproto.com

公司网址：www.digiproto.com

微信公众号：迪捷软件