Code Composer Studio Workshop

议程

概括介绍
- 什么是Code Composer Studio
- 产品路线图
- Code Composer Studio V5 的新功能（对比 V4）

研讨会
- 熟悉CCSv5.1的环境
- TMS320F28069 controlSTICK

什么是Code Composer Studio?

- TI嵌入式处理器的集成开发环境(IDE)
  - 包括调试器，编译器，编辑器，操作系统， 等等
  - IDE基于Eclipse开源软件框架(v4+)
  - 融合了TI设备的支持与功能
- CCSv5是基于原版的 Eclipse
  - CCS使用未修改的Eclipse版本
    - TI将直接向开源社区提交改进
  - 用户可以随意的将各种其他厂商的Eclipse插件或TI的工具拖放到现有的Eclipse环境
  - 用户可以享受到Eclipse中所有最新的改进所带来的便利
- 集成额外的工具
  - 操作系统的应用程序开发工具（Linux操作系统，Android的...）
  - 代码分析，源代码控制...

Code Composer Studio v5

- CCSv5分为两个阶段
  - 5.0
    - 不是为了替代CCSv4
    - 针对需要在设备上运行Linux的用户与多核C6000的用户
    - 增加了一些在CCSv4上没有的Linux的调试功能
    - 现已被CCSv5.1取代，不再提供技术支持
  - 5.1
    - 针对CCSv4的替换版，面向所有用户
- 支持Windows和Linux
  - 注意，并非所有的仿真器（emulator）有Linux版支持
    - SD DSK / EVM板载仿真器，XDS560 PCI没有支持
  - 大多数的USB / LAN仿真器将被支持
    - XDS100，SD510USB/USB+，560v2，BH560m/bp/lan
  - HTTP：/ / processors.wiki.ti.com/index.php/ Linux_Host_Support

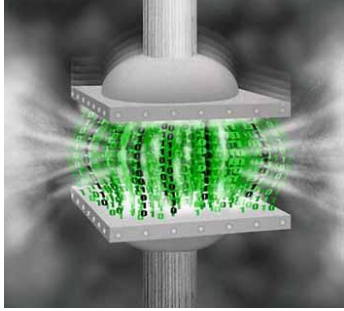
Code Composer Studio 产品路线图

What's New in CCSv5

用户对于CCSv4的反馈

- 需要更小
  - CCS DVD映像太大（下载> 1GB，安装4GB）
  - 需要下载很多不必要的东西
- 需要更快
  - CCS比较迟缓
  - 启动时间和调试器的响应速度有待提高
- 需要更简易
  - 用户界面过于杂乱
  - 很难找出如何开始
- 因此，对于5.1的目标是使CCS“更小，更快，更简易”

更小



- CCSv4x
  - 下载大小是1.2GB
  - 代码大小限制版和DVD镜像版分开，用户经常下载错镜像
  - 用户必须下载的内容超过他们的需要
- CCSv5.1使用动态下载
  - 用户下载一个小的初始安装包
  - 根据用户的选择，相应的软件包将被下载并自动安装
  - 用户可以在以后添加更多的功能
  - 或者用户可以选择下载完整的DVD映像

更快

- 加快常用任务
  - 启动CCS
  - 启动调试会话(Debug Session)
  - 创建一个新项目（良好的初始体验）
- 响应速度
  - 单步(Stepping)（在各种视图打开的状态下）
  - 在实时模式下，表达式（Expressions）和绘图（Graph）视图的连续刷新
  - 存储目标配置（Target Configuration）
  - 加载/烧录程序于闪存

简易用户界面模式

- 简易模式
  - 默认情况下，CCS将打开简单/基本模式
  - 简化菜单选项，工具栏按钮的用户界面
  - TI提供的编辑(CCS Edit)和调试(CCS Debug)透视图（perspective）
  - 简化构建选项（build options）
- 高级模式
  - 使用默认的Eclipse透视图（perspective）
  - 非常类似于CCSv4
  - 推荐给会添加其它Eclipse插件到IDE的用户使用
- 可以切换模式
  - 用户可以在两种模式间随意切换

常用功能

- 创建新项目
  - 必须能非常简单的使用模板创建对应任意设备的新项目
- 构建选项
  - 许多用户觉得 “构建选项”(build options)对话框很难使用，选项繁杂
  - 选项升级需基于编译器版本而不是CCS由更新
- 共享项目
  - 需要使用户更轻松地共享项目，包括版本控制的使用（可移植项目）
  - 设置链接资源的过程需要进行简化

帮助和文档

- 方便
  - 需要可以很容易地从IDE内部访问产品的帮助系统
  - Eclipse和TI的帮助必须进行综合
- 高质量的内容
  - 一个很好的教程！
  - 基于任务的帮助
  - Flash视频的链接
  - 改良的欢迎屏幕

CCS5.1 的更新与改进

- Eclipse
  - 更新的Eclipse及CDT (C/C++ Development Tooling)
  - 使用“原版”的Eclipse和CDT
- 提升可用性
  - 透视图(Perspectives)
  - 视图(Views)
  - 创建项目
  - 使用链接资源(Linked Resources)
- 项目管理的变化
- 调试器的变化
- Linux支持
  - Linux主机(host)支持
  - 支持Linux调试（内核/应用）
- 多内核调试
- 系统分析器（ System Analyzer ）

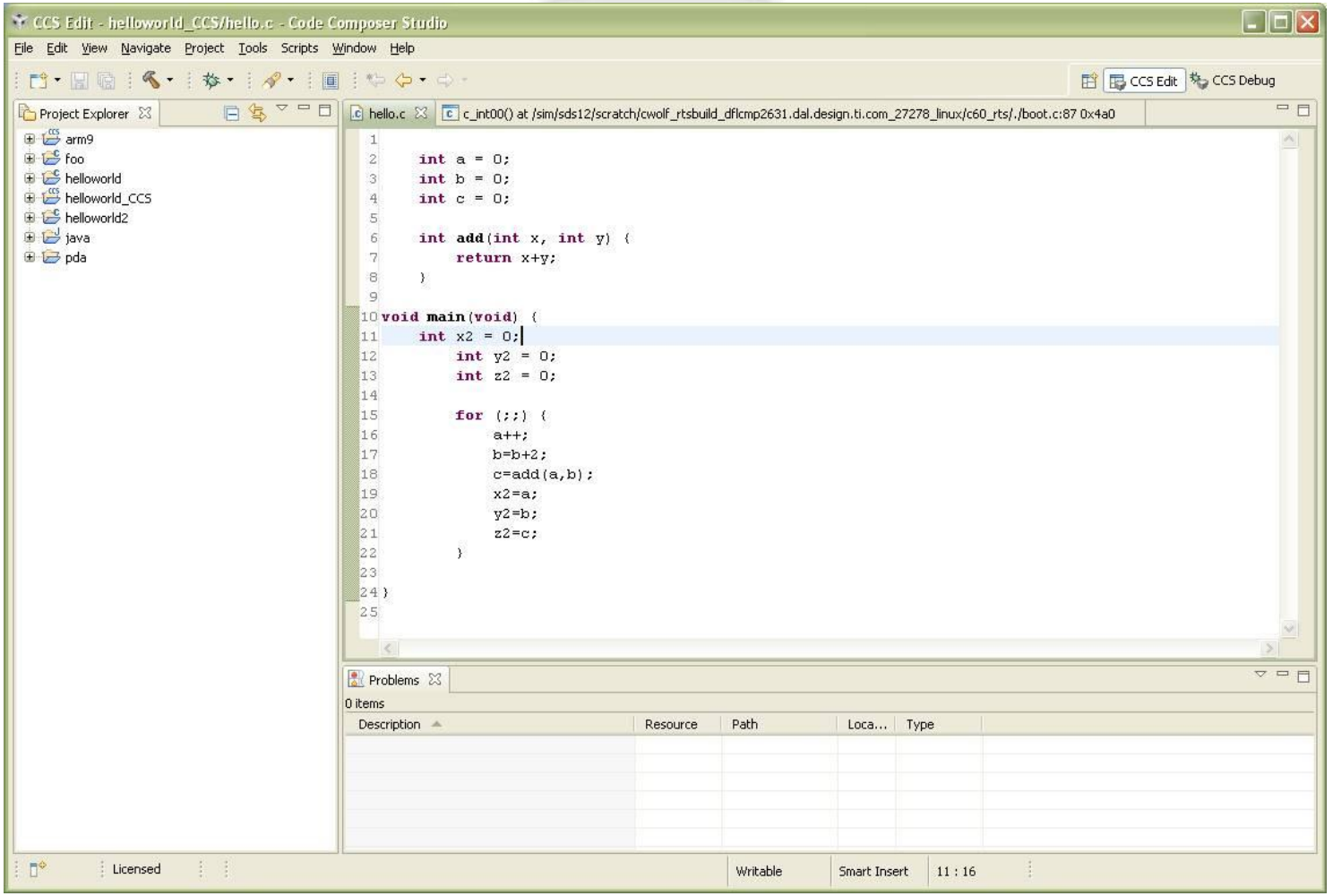
Eclipse的更新与改进

- CCSv5.1使用的Eclipse3.7，CCSv4使用Eclipse3.2
  - 5年间的修复和增强
- 关键项目
  - 编辑器，检索器(Editor/Indexer)改善
    - 在CCSv4中常见的Eclipse区域相关的问题
    - 更快
    - 更可靠
  - 拖放支持
  - 支持在链接文件时使用宏（可移植项目）
  - 动态语法检查
  - 从Eclipse内搜索插件
  - 警告和错误在控制台的输出上更明显

使用未修改版本的Eclipse

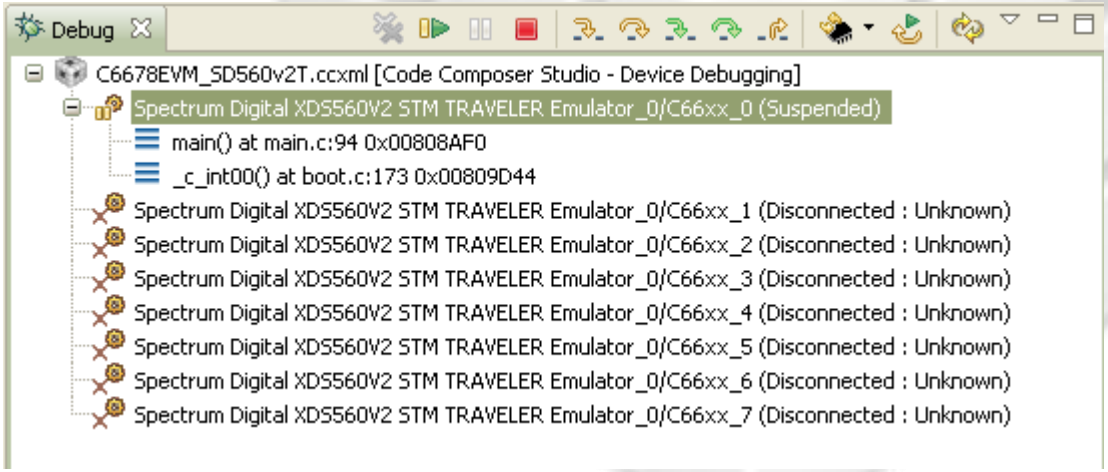
- CCSv5是基于原版的Eclipse
  - 使用未修改版本的Eclipse
    - TI直接把改进提交到开源社区
      - TI对Eclipse的贡献案例
        - » 打开多个调试会话视图的实例
        - » 使调试视图定在一个特定的调试上下文
        - » 在调试视图给CPU分组
        - » 给于“断点”视图更灵活的层次结构（添加更多的列，等等）
- 把其他供应商或TI工具的Eclipse插件拖放到现有的Eclipse环境
  - 更好的Eclipse插件的兼容性
- 用户可以使用Eclipse的所有最新功能与改进
  - 随时更新到新的Eclipse

简化的透视图（Perspectives） – CCS Edit

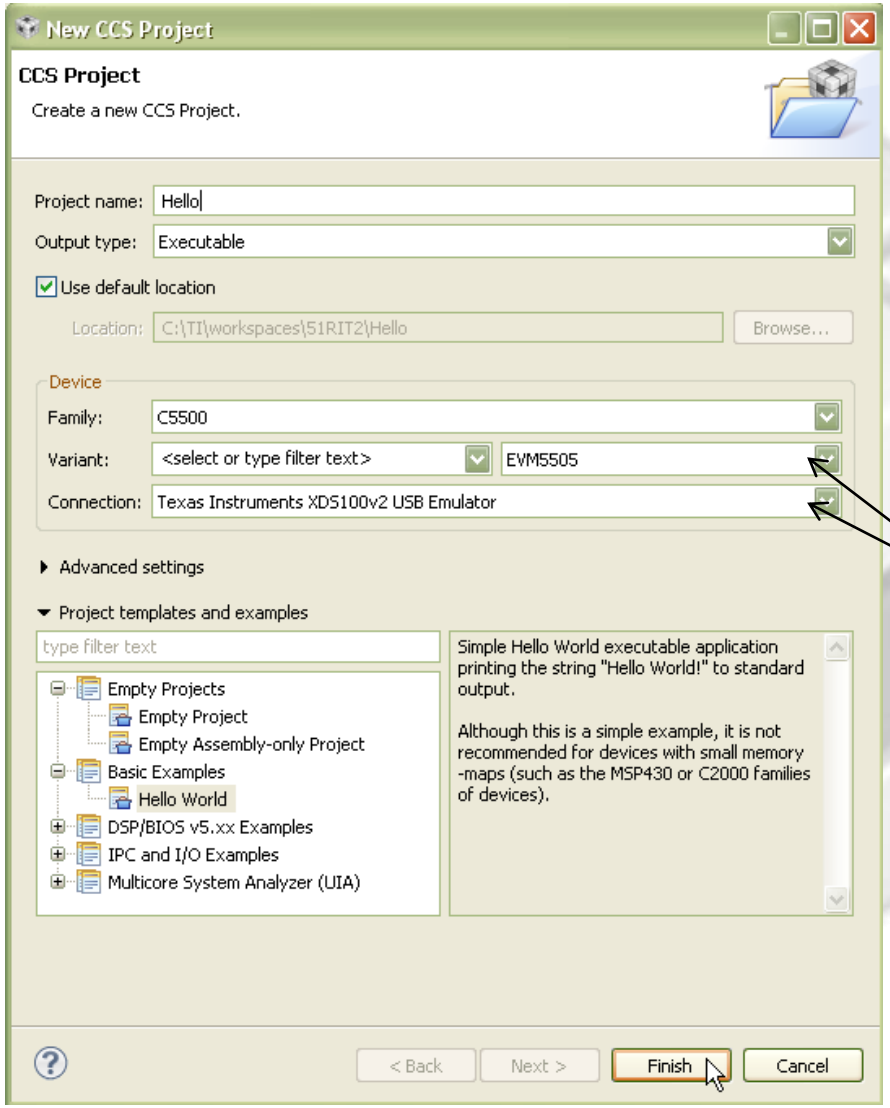


简化的视图 – 调试视图（Debug View）

- 清爽，简洁，更多自定义选项
- 没有额外的“线程”节点
- 没有“CIO/目标”(CIO/target)错误节点
- “项目/目标配置”(Project / Target Configuration)节点下直接显示多个CPU设备
- 可以自定义选择显示更多JTAG的层次结构
- 可以收合到单行来优化屏幕空间 （“面包屑”模式）

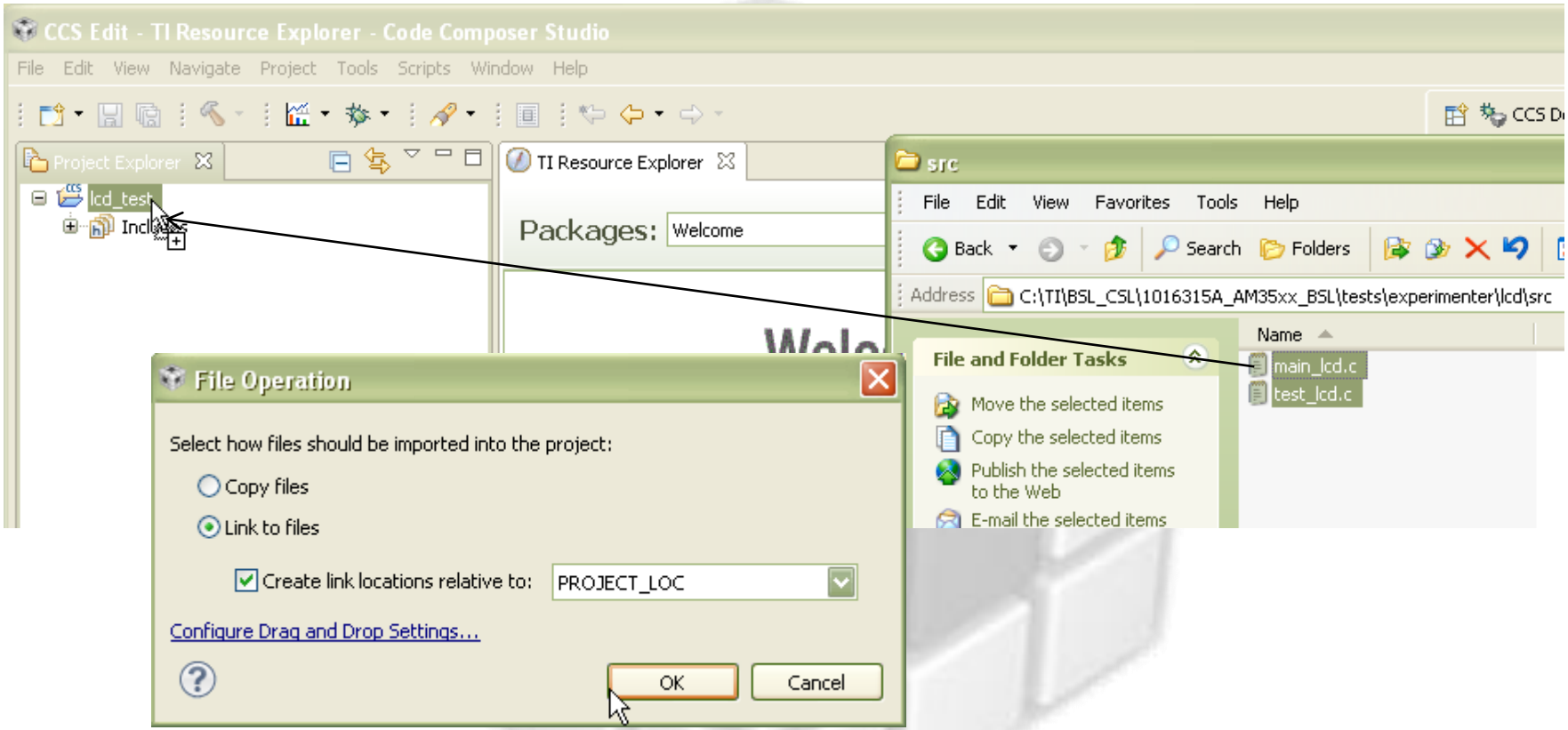


简化新项目向导 – 一页完成

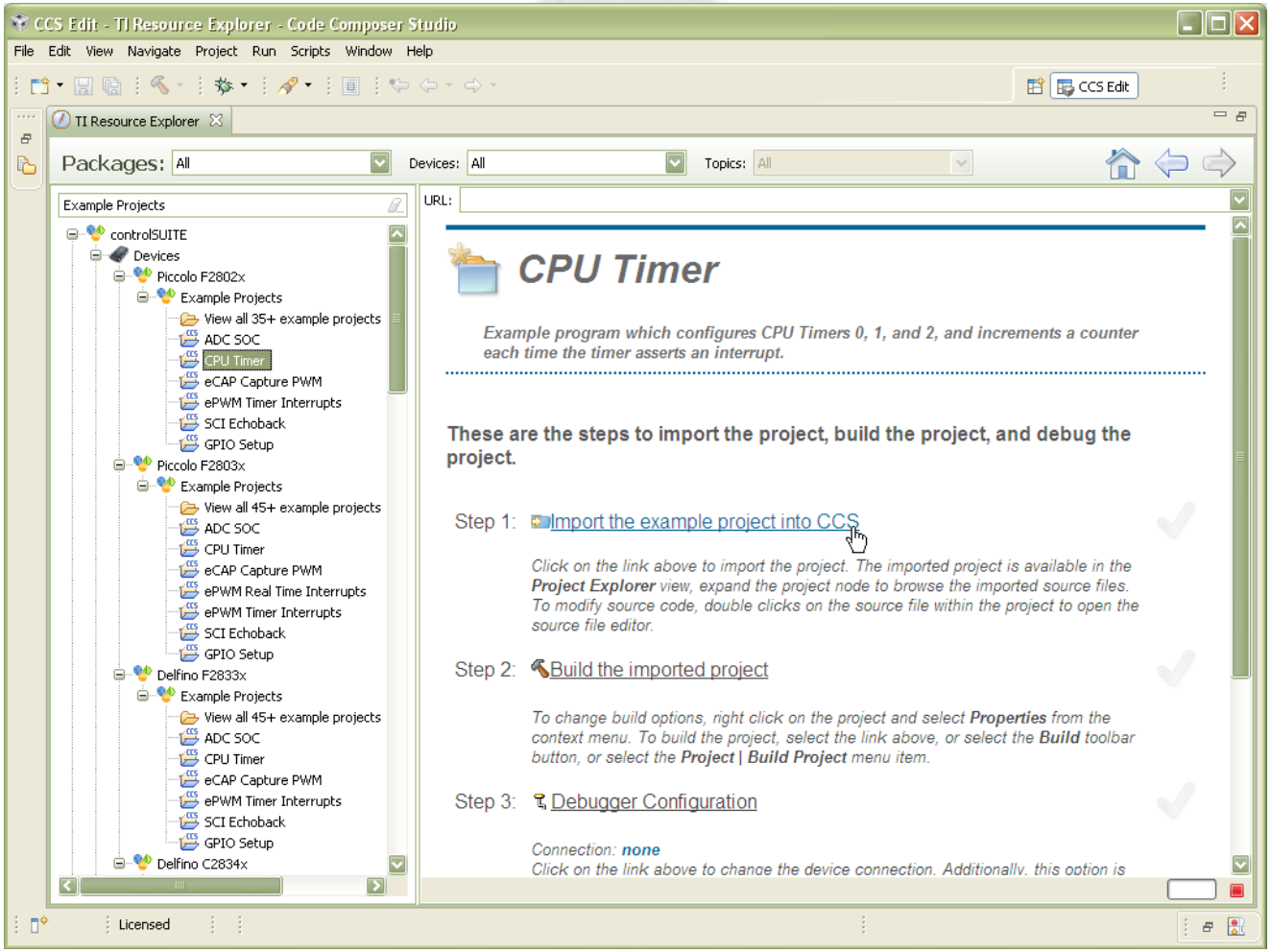


指定一个特定的设备型号和连接类型，向导将会自动创建目标配置文件

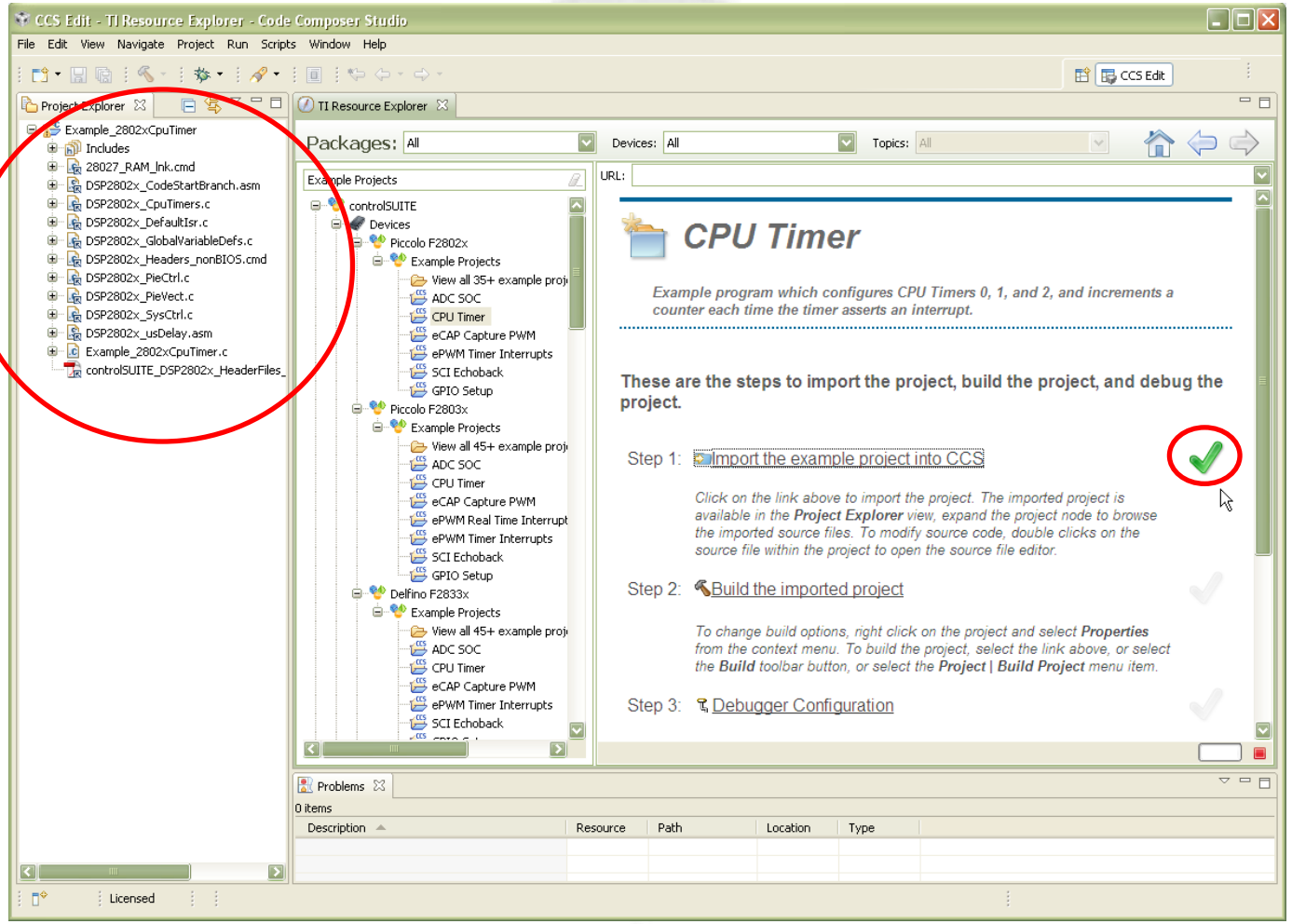
链接源文件到项目



资源管理器 - 教程



资源管理器 - 教程



视图: CCSv5 的变化

- 断点(**Breakpoint**):
  - 取消了在试图内列的就地编辑（使用断点属性页）
  - 过滤视图，只显示当前调试上下文 （debug context）的断点
  - 与调试视图链接： 突出显示当前调试上下文中，目前已经到达了的断点
- 寄存器(**Register**):
  - 无“网格”(Grid)模式的支持
- 变量(**Variables**)（V4 - > Local （本地））和表达式(**Expressions**)（V4 - > Watch（观察））：
  - 这些视图已被重命名为标准的Eclipse名称
  - 在表达式视图“添加全局变量”(Add global variables)的功能
  - 表达式格式(Hex, Dec, Bin, Char)可以在视图级别调整
  - 默认值因类型而定
- 反汇编(**Disassembly**)
  - 无反汇编显示风格上的变化支持
  - 可选择链接到激活的调试上下文：更新视图到当前调试上下文的地址
    - 按照堆栈帧
  - 跟踪表达式
- 控制台(**Console**)
  - 不支持从控制台运行GEL命令（使用脚本控制台）
  - 有一个可选 CI / O控制台对应所有CPU （多核调试）

项目管理的变化

- “激活的项目”(**Active project**) 是选定的项目上下文
  - 在“项目资源管理器”(Project Explorer)点击另一个项目会自动将其变为“Active project”
  - 在没有选定的项目上下文的情况下点击“调试”(Debug)按钮，将会开启最后一个被调试的项目的调试会话
- 在项目级别管理链接资源（项目属性页面）
  - 在项目级别创建链接资源路径变量
  - 编辑/转换/删除一个项目的链接资源
- “C/ C+ +项目”视图更名为“项目资源管理器”
- 项目“宏”更名为“构建变量”(**Build Variables**)
- 更多的项目模板支持
- 命令行对于项目的“创建/导入/构建”命令语法改变

  CCSv4：命令的调用由使用一个特定的.jar执行Java来完成

  > jre\ bin\ java -jar startup.jar<restOfCommand>

  CCSv5：使用“eclipsec.exe”或“eclipse”

  Windows ： > eclipsec -nosplash<restOfCommand>

  Linux的： > Eclipse -nosplash<restOfCommand>

调试器的变化

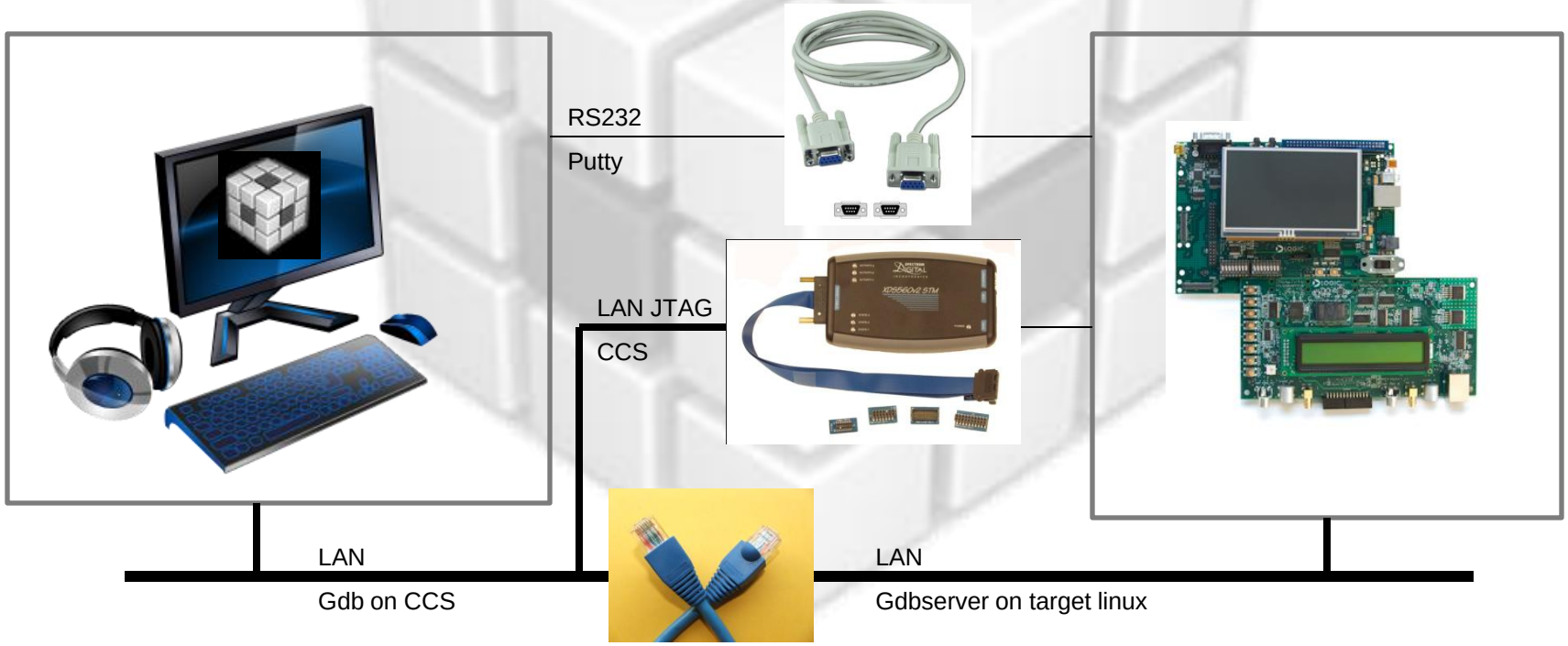
- 没有全局（工作区）级的 “调试选项”(Debugger Options)
- 大多数的调试视图支持”钉住”（pin）和“克隆”（clone）
- 调试配置(Debug Configurations)
  - 指定在启动调试会话时运行的调试初始化脚本（JavaScript文件）
    - 使用DSS初始化目标
    - 一个脚本可以在多核的调试环境下初始化多个CPU
    - 可以替换启动GEL文件
  - 在“程序 /目标/源“标签里能以每一个CPU独立指定设置
- CCSv4的“目标”(Target)菜单被v5的“运行”(Run)菜单所取代

Linux主机支持

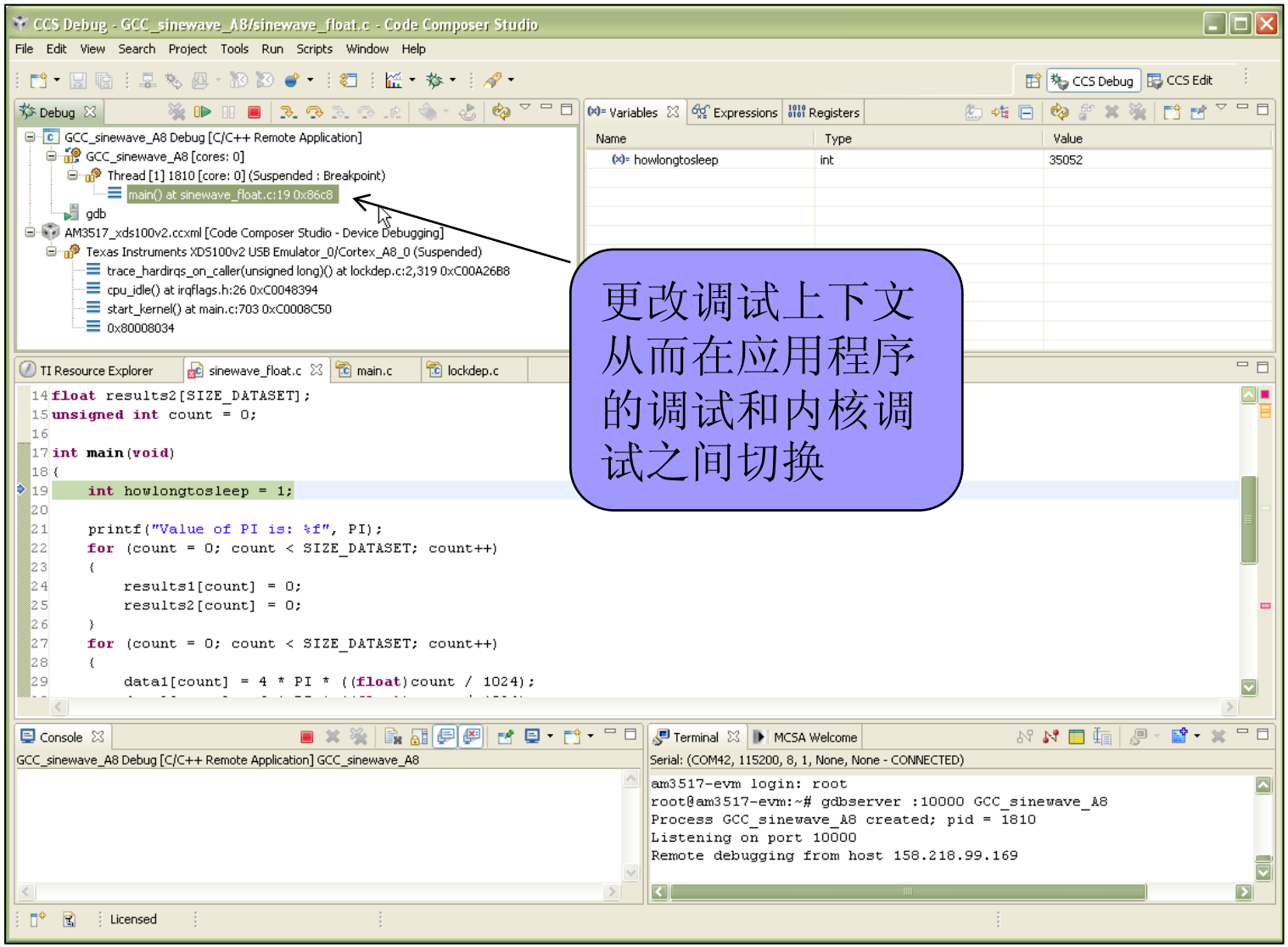
- 在以下环境中已测试：
  - Ubuntu 10.04的32位
  - Ubuntu 10.10的32 / 64位
  - SUSE 11 32位
- CCS依赖于一些库（因分布版本而异）
- 并非所有的功能在Linux上都有支持
  - 跟踪（Trace）
    - 内核跟踪（XDS560T）
    - 嵌入式跟踪缓冲（ ETB (Embedded Trace Buffer) ）
    - 系统跟踪模块（ STM (System Trace Module) ）
- 请注意，并非所有的仿真器在Linux上都有支持
  - SD DSK / EVM板载仿真器，XDS560 PCI不支持
- 大多数USB/ LAN的仿真器都支持
  - XDS100，SD510USB/USB+，560v2， BH 560m/BP/LAN
    http://processors.wiki.ti.com/index.php/Linux_Host_Support

Linux 开发(目标)

- CCS支持Windows和Linux主机电脑
- 通过集成的GDB调试Linux应用程序
- 通过JTAG调试Linux内核



Linux应用/内核调试



多核调试

- 指定一个JavaScript来执行多CPU初始化
  - 用于在各个CPU上执行GEL命令，加载程序，运行过去的初始化例程
- 一些设置在相同CPU可见的调试会话之间持续存在
  - CCSv4中这一行为由脚本完成，V5中有明确的GUI来控制这种行为。
- 没有“同步模式”(Synchronous Mode)按钮
  - 命令可以被发送到动态创建的自定义CPU组
- 调试会话之间保存创建的自定义CPU组
- 状态视图(Status view)在同一视图中显示所有CPU的状态信息（包括路由器（ routers ））

系统分析器

- 在任何时间，观测整个系统的应用程序，操作系统和硬件
- 把多个内核的软硬件仪器的输出关联到一个全局的时间轴
- 系统的分析器是由两个核心组件构成
  - UIA（统一仪表体系结构 - Unified Instrumentation Architecture ）：目标端日志记录(logging)，运行控制(runtime control)和数据运动(data movement)工具包
  - DVT（数据分析和可视化技术 - Data Analysis and Visualization Technology ）：主机端运行控制，数据采集(data collection)，数据解码(data decoding)，数据分析(data analysis)和数据可视化(data visulization)工具

升级到CCSv5.1

- CCSv5.1
  - 替换CCSv4
  - 需要CCSv5许可证
    - 订阅有效的用户将免费获得CCSv5.1
    - 订阅已过期的用户可以更新订阅来获取升级

迁移

- 从CCSv3迁移到CCSv4有一定难度
  - 完全不同的环境
  - 新项目系统
  - 目标配置(Target Configuration)更改
  - CCS的整体面目都大不一样
- CCSv4 CCSv5.1迁移会顺利得多
  - 环境非常相似
  - 项目系统是相同的（的简单的导入）
  - 目标配置是相同的
- CCSv3 CCSv5.1迁移
  - 加强V3到V5的项目转换向导
  - UI的简化将有助于用户缩短学习曲线
  - 详细的文档帮助人们更快熟悉v5

迁移推荐顺序 – CCSv4 -> CCSv5

- 工作区
  - 在v5时使用新的工作区
- 项目
  - 一个V4项目导入到V5后会被修改，使其不能再用V4打开

Questions?

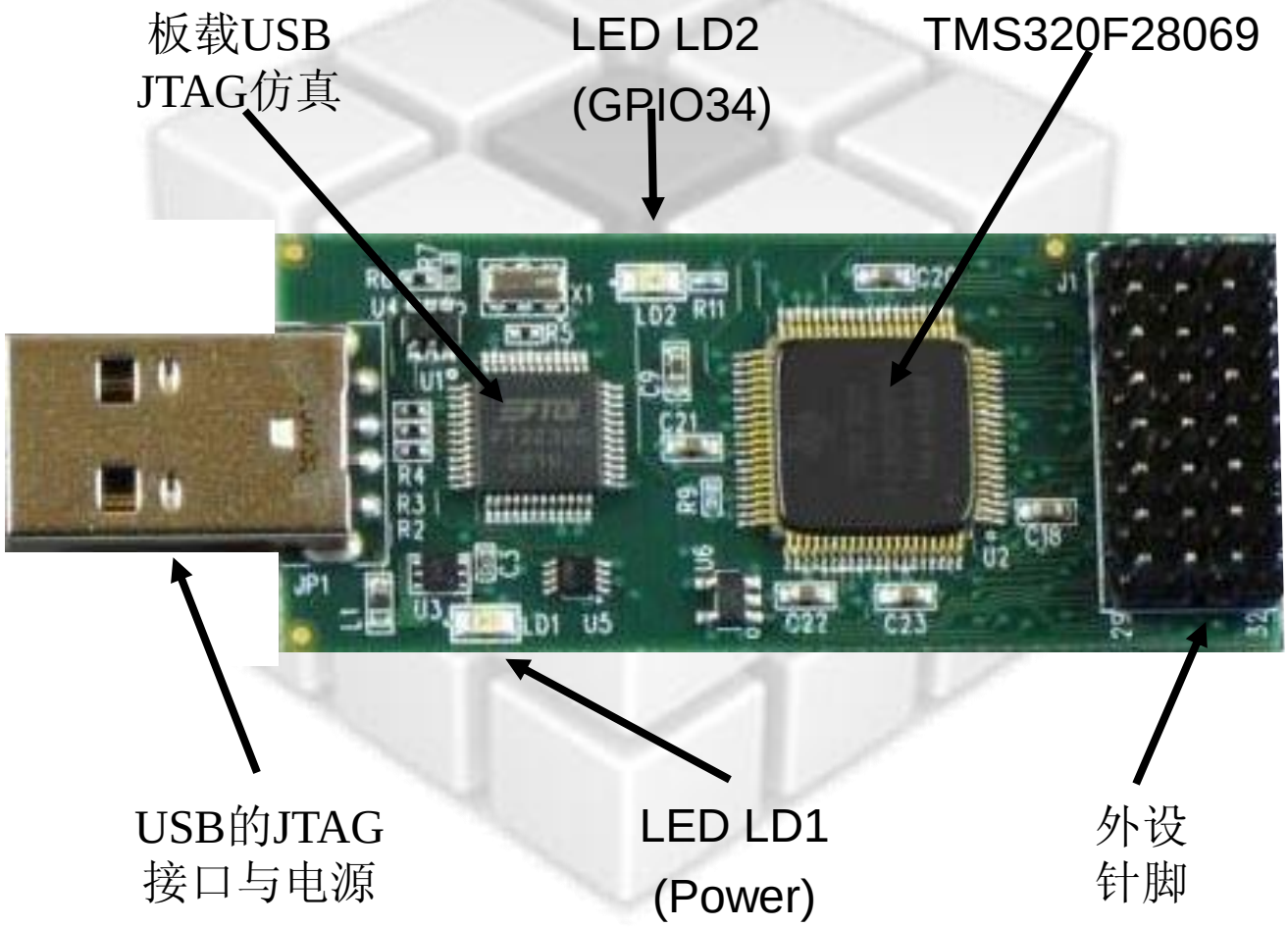
CCSv5和TMS320F28069 controlSTICK入门

Getting Started with CCSv5 and TMS320F28069 controlSTICK

什么是TMS320F28069 controlSTICK?

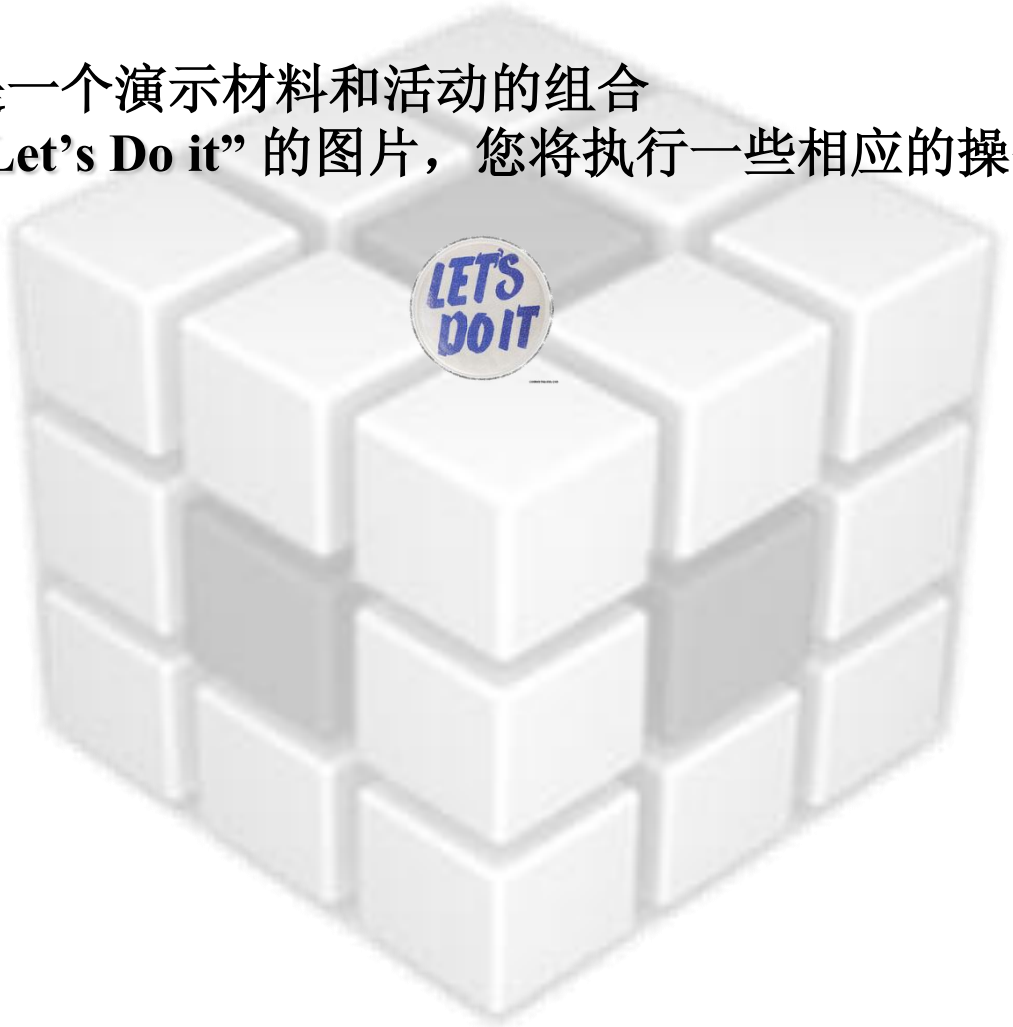
- Piccolo（短笛）F28069微控制器
- 快速，方便，具有TI的新型Piccolo F2806x微控制器的所有先进功能，只售$ 39
- 方便和易于使用的controlSuite GUI
- 提供简易的实验展示Piccolo F2806x MCU的浮点运算能力
- 比记忆棒稍大
- 板载仿真器，可访问所有I / O引脚
- 详细的范例软件和文档
- 完整的硬件原理图，Gerber文件等

TMS320F28069 controlSTICK



研讨会说明

- 本次研讨会是一个演示材料和活动的组合
  每当你看到“Let’s Do it” 的图片，您将执行一些相应的操作



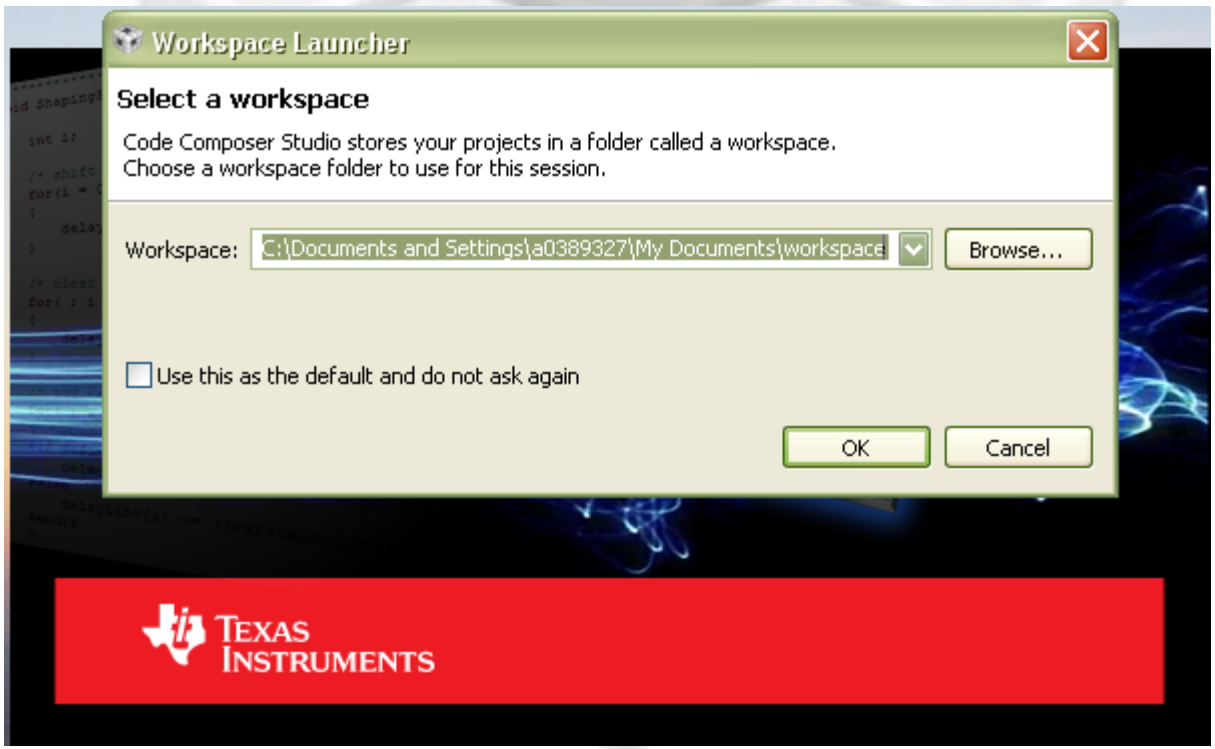
闪烁LED的范例
BLINKING LED EXAMPLE

闪烁的LED：练习的摘要

- 主要目标
  - 创建和构建一个简单的程序blinkLD2
  - 启动调试会话和加载程序到controlSTICK
  - 运行程序blinkLD2
- 工具和概念
  - 工作区(Workspaces)
  - 欢迎屏幕/资源管理器(Welcome screen / Resource Explorer)
  - 项目概念
  - 视图的基础
  - 调试启动
  - 调试控制
  - 分析器时钟(Profile Clock)
  - 本地历史
  - 调试闪存内现有的代码（加载符号调试）
  - 构建属性(Build Properties)
  - 更改编译器版本

工作区

- 启动CCS，选择一个工作区文件夹
  - 默认为您的用户文件夹

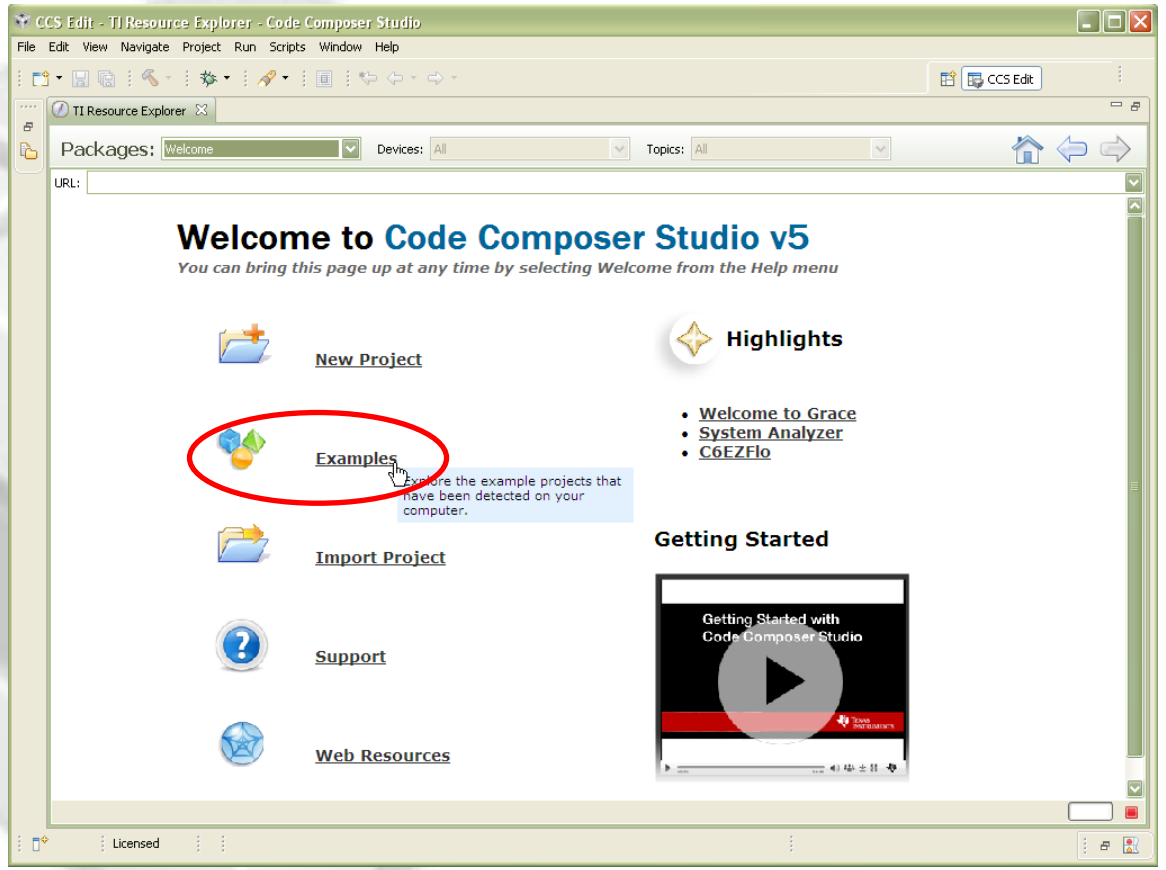


Eclipse概念：工作区（Workspace）

- **CCS的主要工作文件夹**
- **包含被其定义的所有项目的管理信息**
  - 创建新项目时所使用的默认地点
- **用户首选项(User preferences)，自定义的透视图(custom perspectives)，插件的缓存数据(plug-in cache)，等等，都存储在工作区**
- **工作区概念不可与CCSv3工作区文件（*. WKS）混淆**
- **可维持多个工作区**
  - 每个CCS实例仅可以同时打开一个工作区
  - 相同的工作区不能共享多个CCS的运行实例
  - 不建议用户之间共享工作区

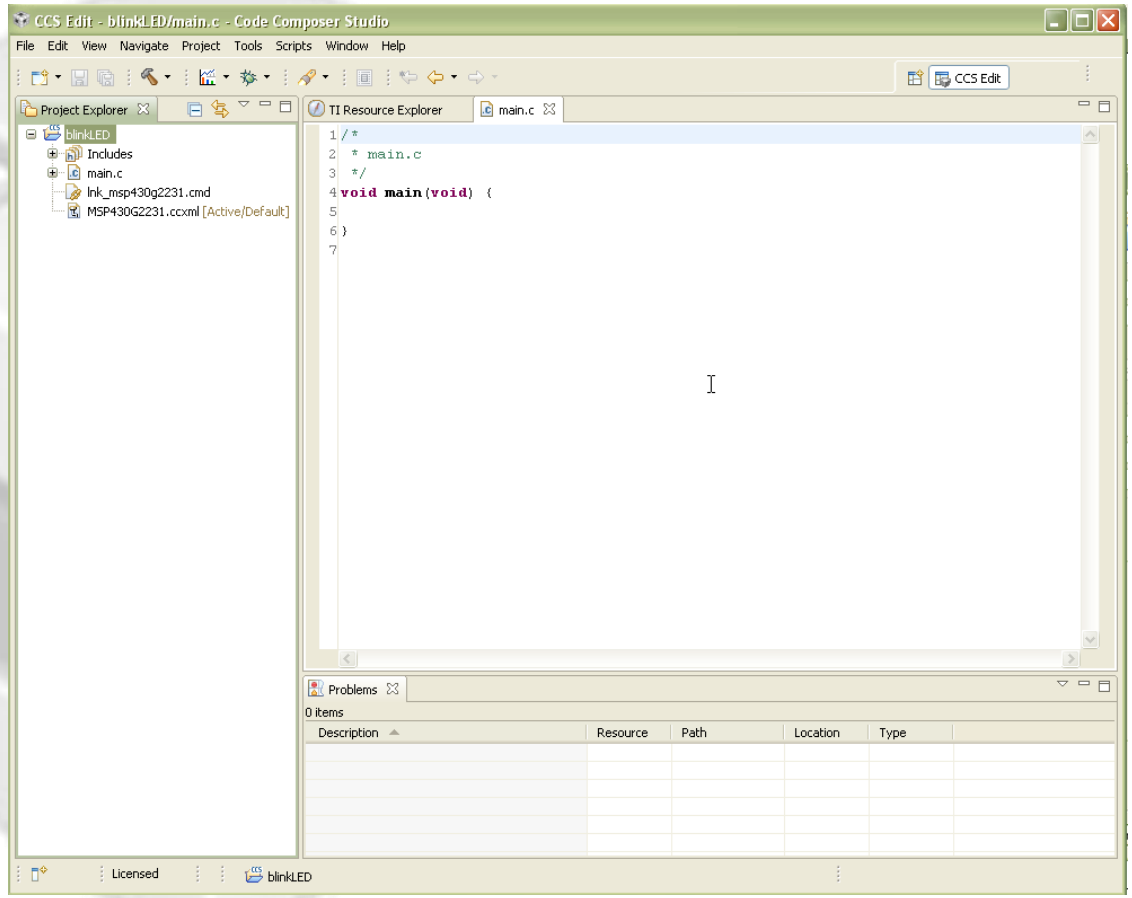
资源管理器

- 在首次使用一个新的工作区时，“资源管理器”将显示“欢迎”页面
- 视频介绍CCS入门知识
- 包含文档链接，例如，支持资源
- 按钮来创建一个新的项目，或导入一个现有的项目到工作区
- 可以随时从“帮助 - >欢迎使用CCS”(Help->Welcome to CCS)返回到资源管理器



Eclipse概念：工作台（workbench）

- “工作台”是指主要CCS 图形界面窗口，
  - 相当于在CCS3.x中的“控制窗口”(control window)
- 工作台包含所有用于开发和调试的各种视图(view)与资源

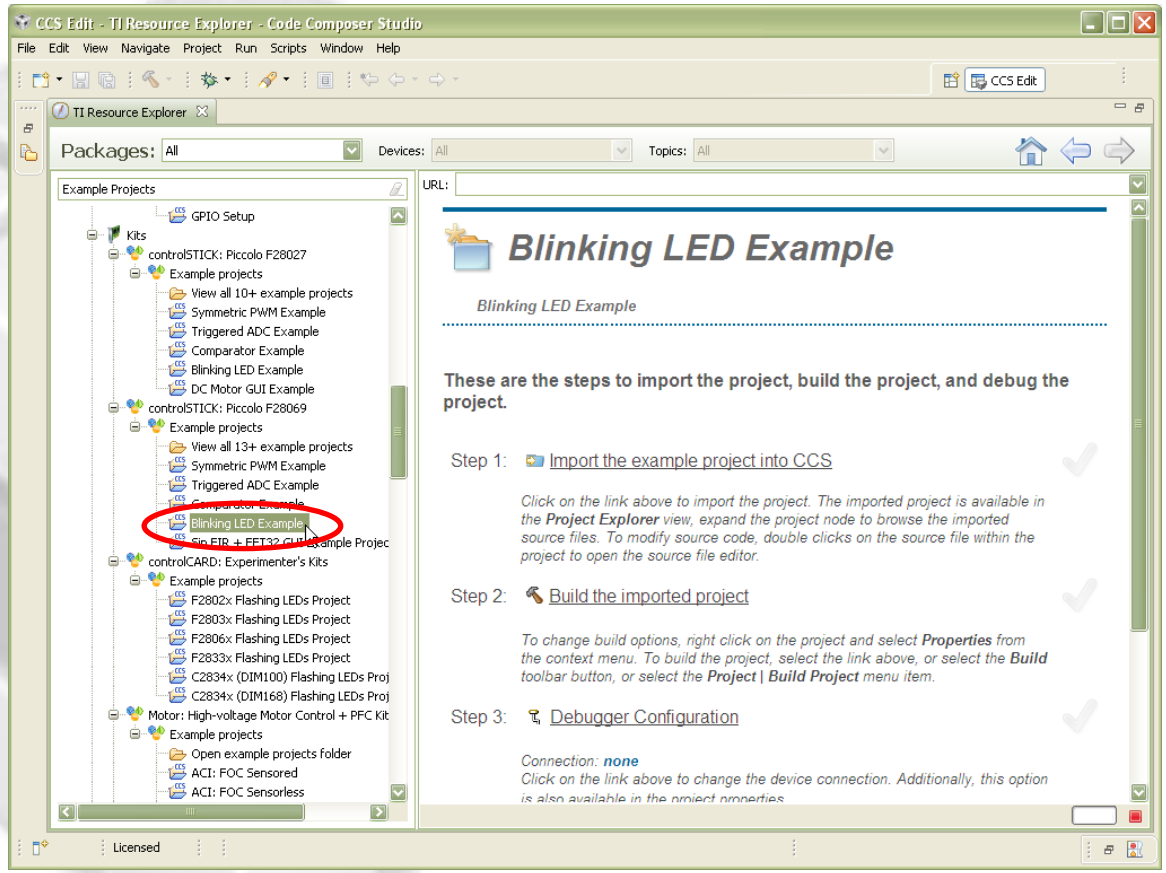


工作台（与CCSv3.x比较）

- **CCS的3.x版**
  - 每个调试的CPU只可以拥有一个打开的控制窗口
  - 每个控制窗口(Control Window) 之间的信息不能共享
- **CCS的4 / 5**
  - 可以打开的多个工作台窗口（“窗口” - >“新窗口”）（ 'Window->New Window ）
  - 每个工作台窗口可以有视觉上的不同（ 视图， 工具条的位置安排等），但是对应同一个CCS实例中的工作区
    - 从一个工作台打开的项目将能在所有的工作台窗口中看到

资源管理器：示例

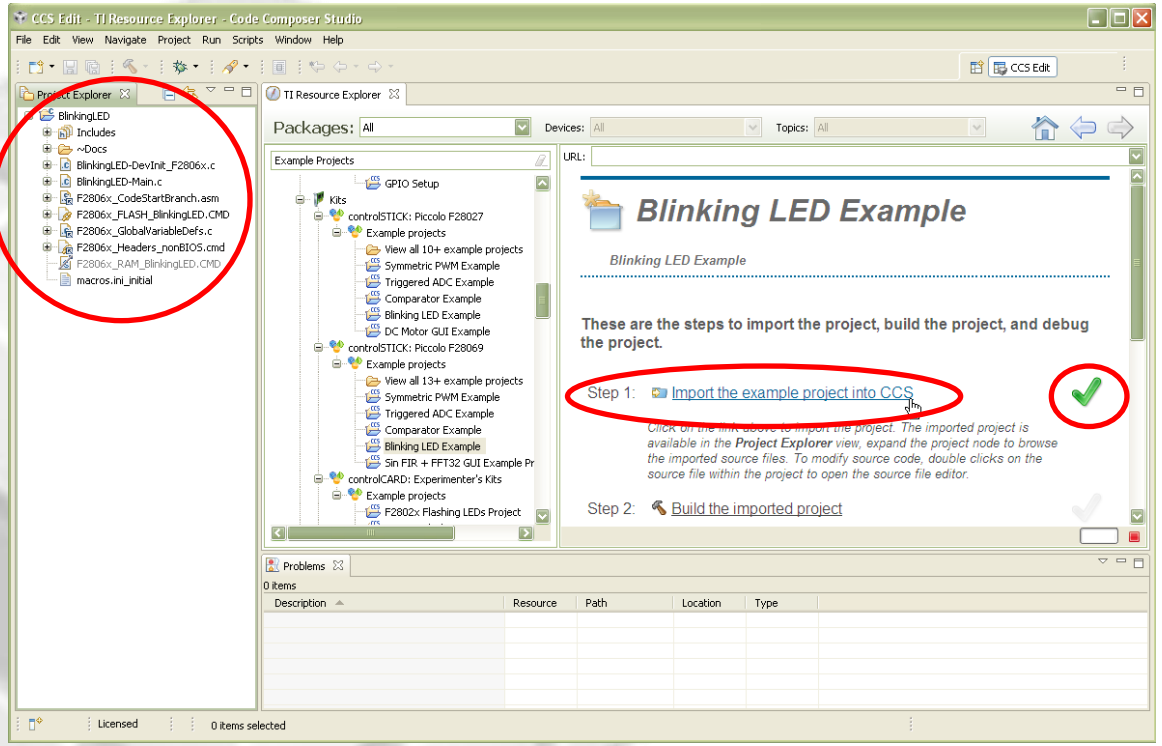
- 显示资源管理器中发现的所有示例项目
- 说明如何建立/运行示例的步骤
- 在“controlSUITE - >kits - > controlSTICK：Piccolo F28069” 下选择“Blinking LED Example”项目



第1步：导入“闪烁LED(Blinking LED)”项目

- 点击“导入CCS示例项目”(Import the example project into CCS)链接，按照步骤1把“闪烁LED”(Blinking LED)项目导入CCS的工作区

- 如果导入成功的话，该项目将出现在“项目资源管理器”。并且一个绿色的复选标记会出现在链接的旁边，以表示成功

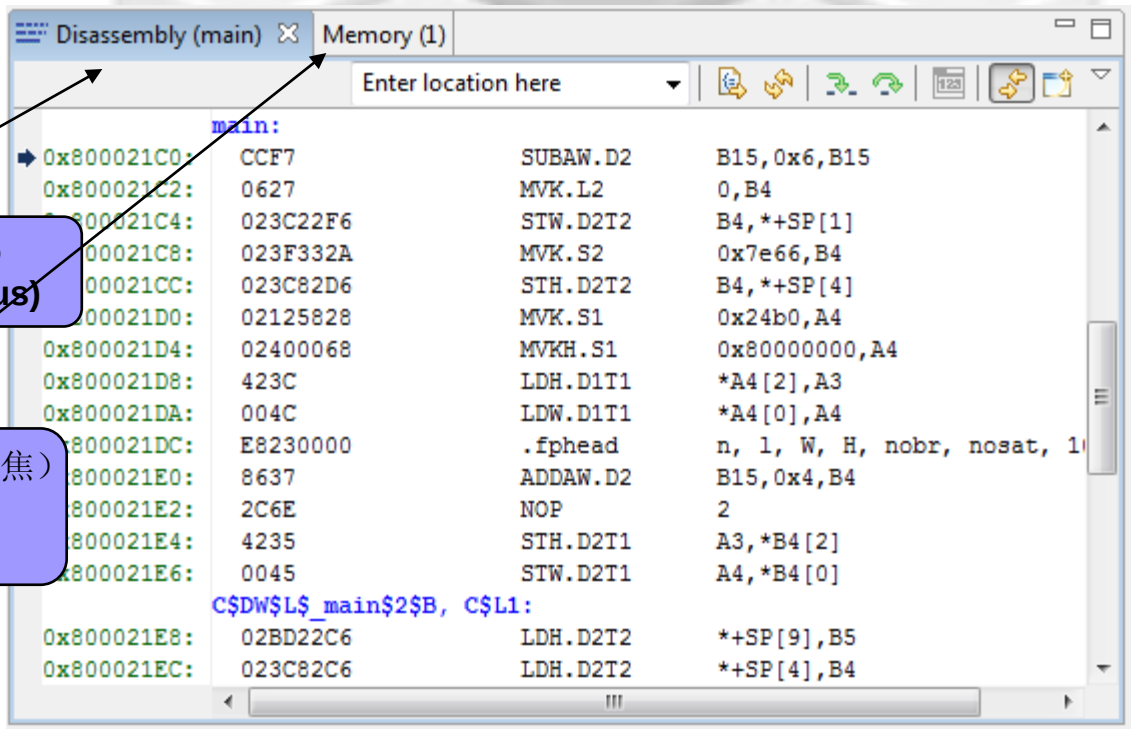


Eclipse概念：项目(projects)

- 项目映射到文件系统中的目录
- 可以添加或链接文件到项目
  - 添加文件到项目
    - 复制文件到项目文件夹中
  - 链接文件到项目
    - 在您的项目中引用该文件
    - 文件停留在原来的位置
    - CCS3.x的项目只有这个文件管理概念
- 项目一定是在打开或关闭的其中一种状态
  - 关闭的项目：
    - 仍在工作区中，但它不能由工作台修改
    - 关闭的项目资源不会出现在工作台，但资源仍然驻留在本地文件系统
    - 关闭的项目需要较少的内存，并不会被频繁的扫描
    - 这不同于CCS3.x中，关闭的项目不会出现在CCS3.x的项目视图窗口。
- 项目必须被导入工作区后才可以打开
  - CCSv4/5，CCE的项目和传统CCSv3 项目均可导入到工作区中

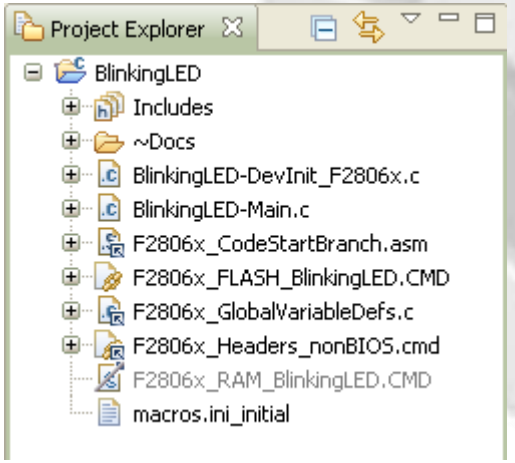
Eclipse概念：视图

- 视图是工作台窗口内的窗口，提供一些信息的视觉化表示
  - 大多数的视图可以在菜单“视图”(View)中访问
  - 可以从整理标签(organization tabs)来识别



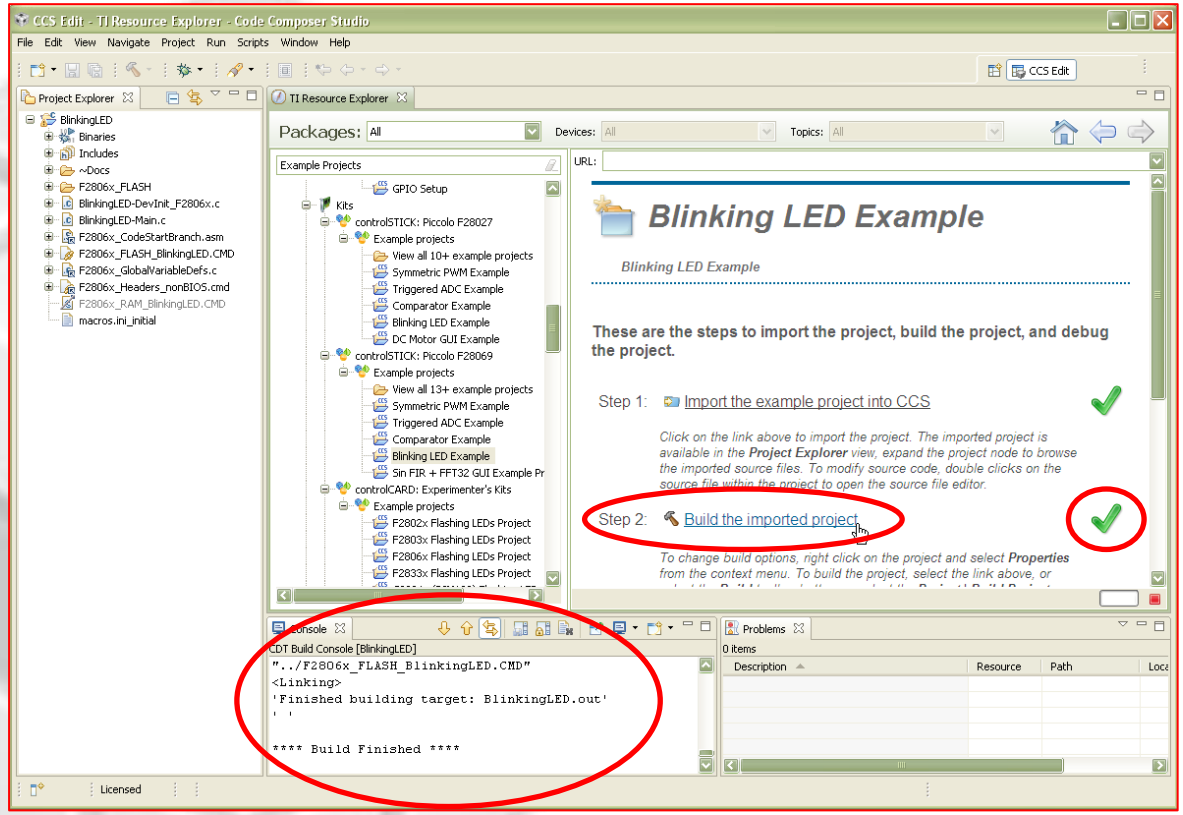
视图：项目资源管理器(Project Explorer)

- 显示所有当前工作区中定义的项目
- 主要是以文件系统的形式来展示项目文件夹中的内容
  - 链接文件都标有一个特殊的连接图标
- 可以使用过滤器来隐藏的各种文件类型，以减少在视图混乱
  - 默认下会过滤CCS生成的项目文件（.*）



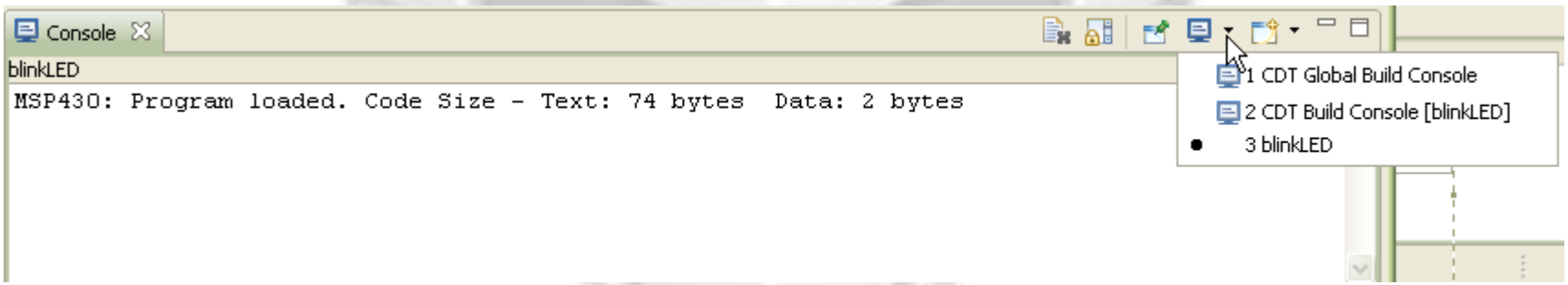
第2步：构建“闪烁LED”项目

- 按照第2步，点击“构建导入的项目”(Build the imported project)链接，构建“闪烁LED”(Blinking LED)项目
- 构建“控制台”(Console)视图将出现在底部，输出项目构建消息
- 如果构建成功，一个绿色的复选标记会出现在链接旁边表示成功



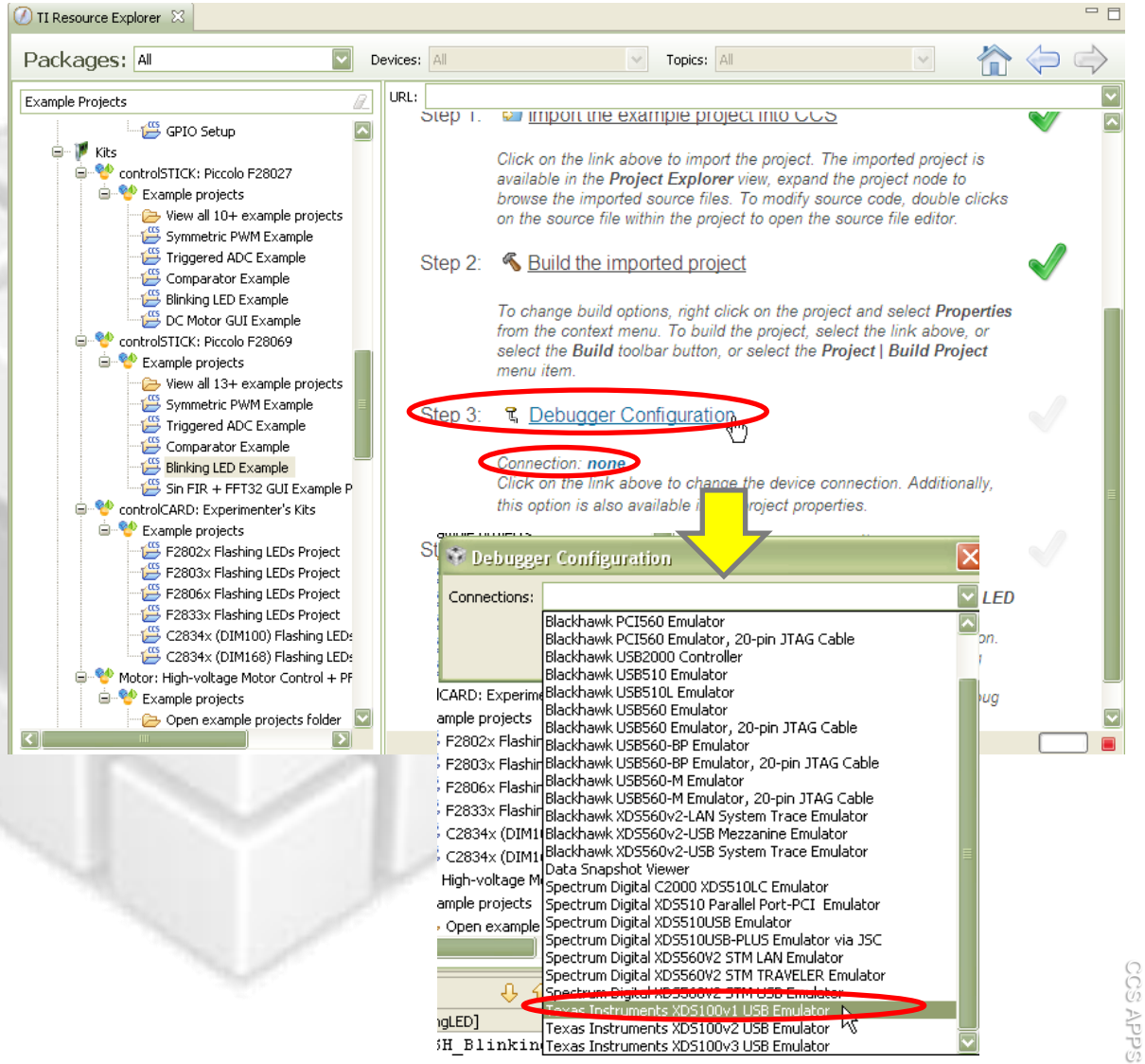
视图：控制台(console)

- 多个上下文 (context)
  - 可以显示构建(build)消息或调试(debug)消息（包括CIO），取决于所选定的控制台
  - 当一个新的消息产生时上下文会自动切换
    - 可以使用“钉” (pin) 功能定住窗口来避免自动切换
  - CCS-3.x 中，构建输出，CIO的输出和调试器的消息都有专门的视图，
- 您也可以打开多个控制台窗口
  - 比如，一个对应CIO的输出和另一个对应构建消息



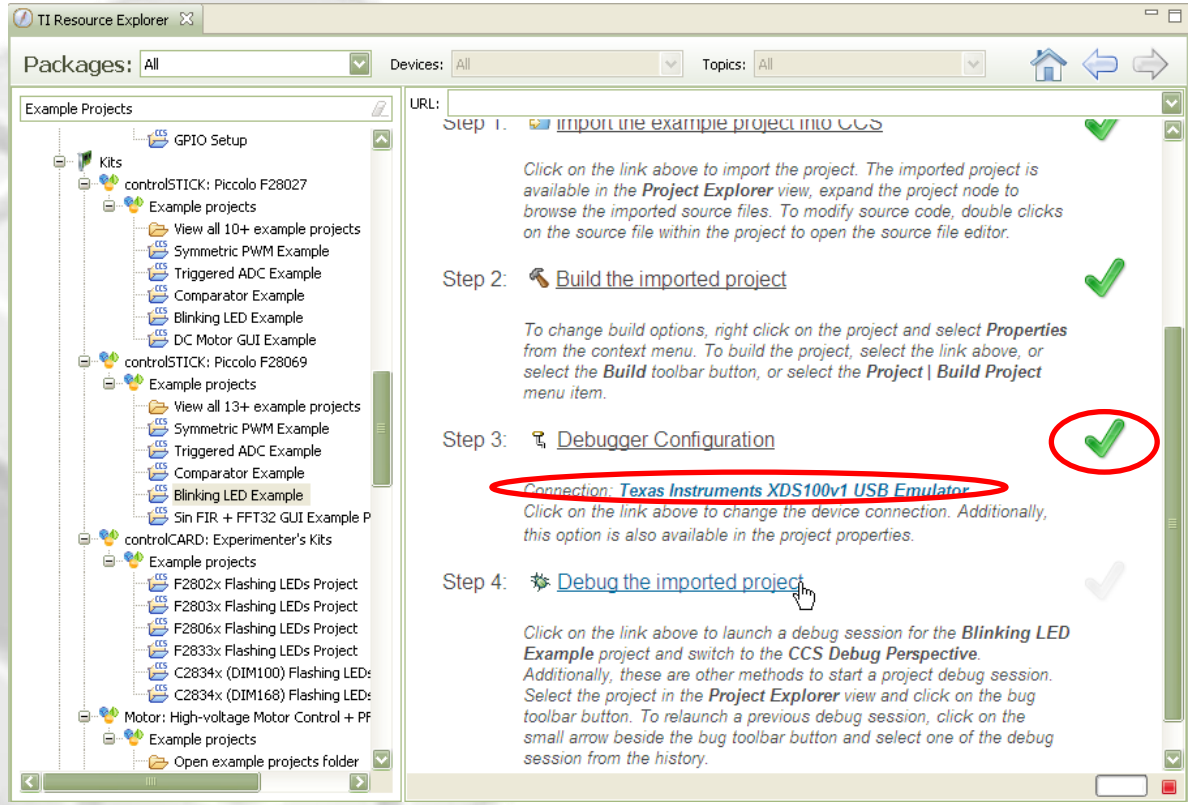
第三步：指定“调试器配置” (Debugger Configuration)

- 需要注意第3步的“连接”(Connection)类型是没有定义的（无）
- 按照第3步，然后单击“调试配置”(Debugger Configuration)链接
- 一个对话框将会出现，显示连接列表
  - 选择“‘Texas Instruments XDS100v1 USB Emulator”
  - 完成时，选择“OK”



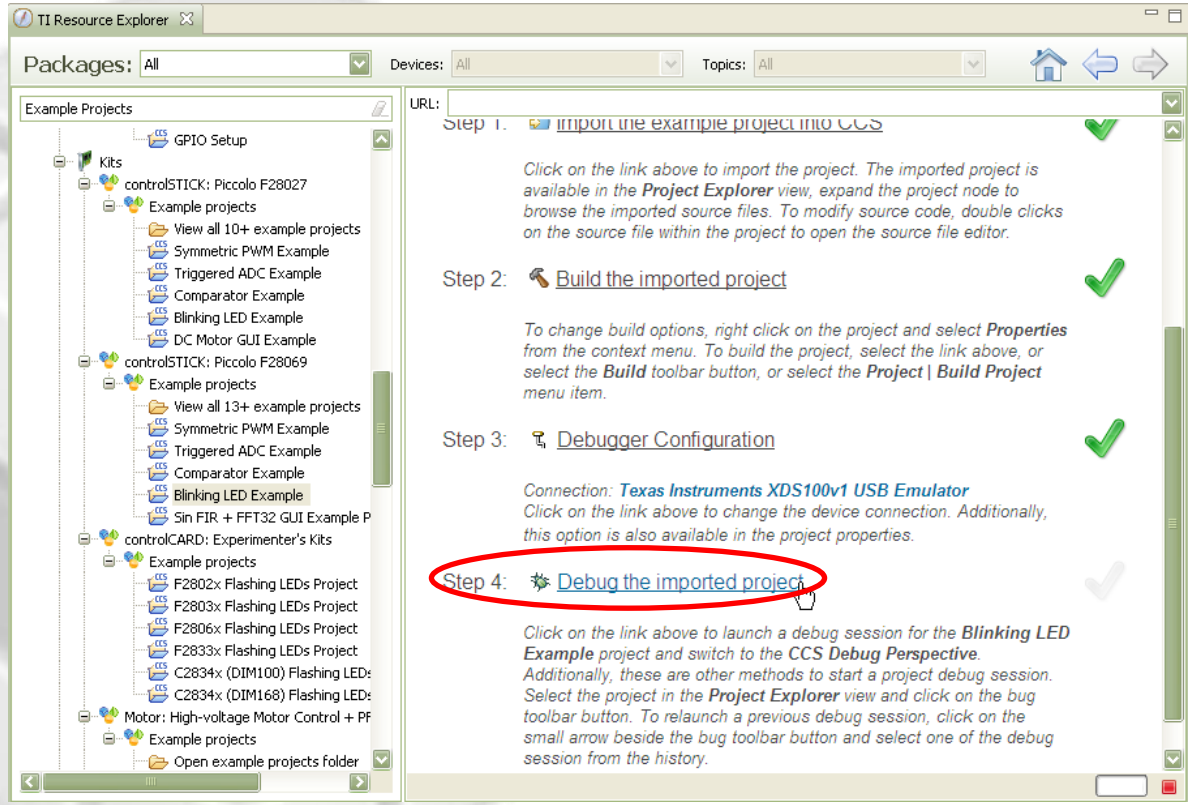
第三步：指定“调试器配置”

- “连接”（Connection）类型的字段将被更新，以显示所选择的连接
- 链接的旁边会出现绿色复选标记指示步骤完成

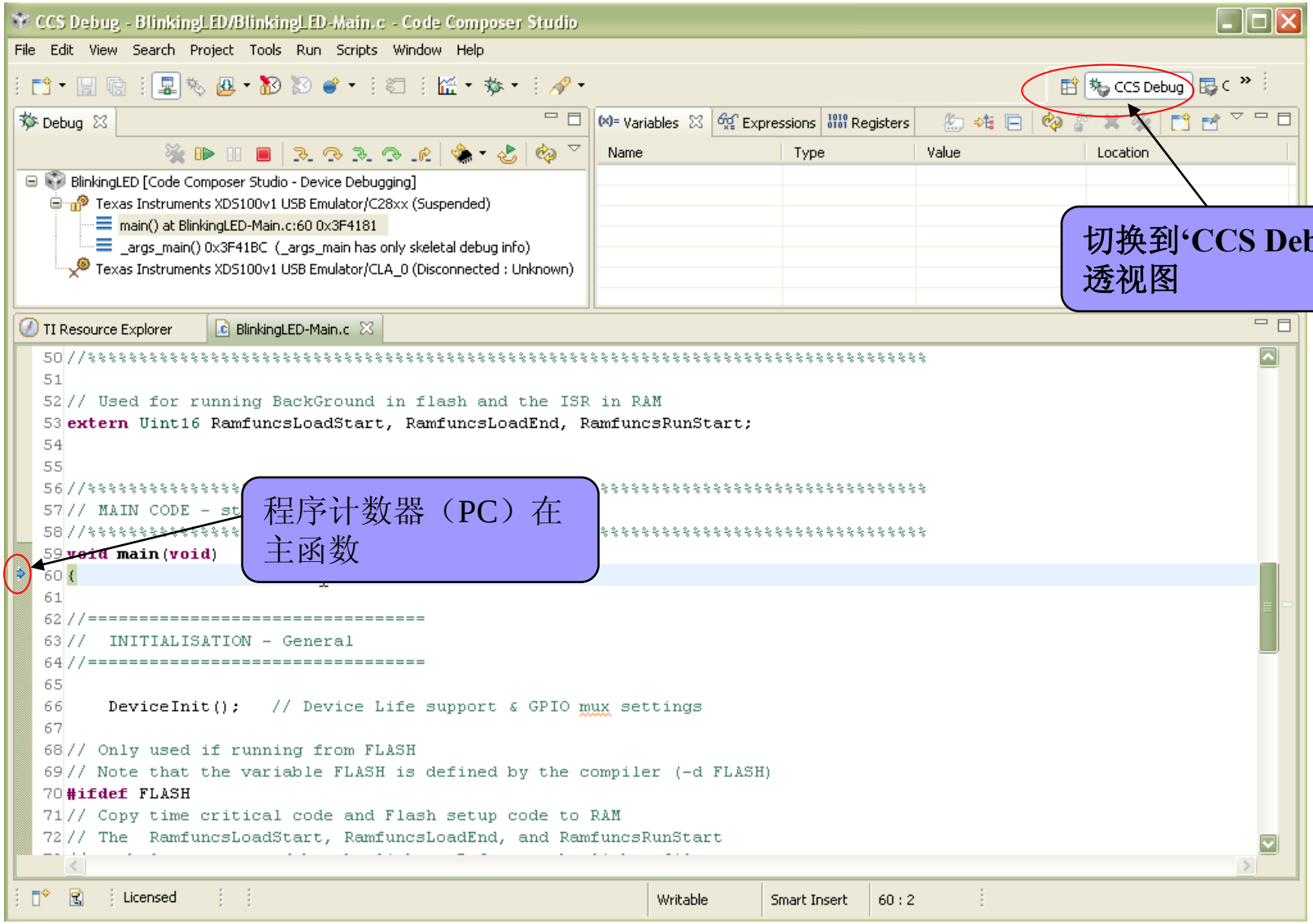


第4步：调试“闪烁LED”项目

- 按照第4步，通过点击“调试导入的项目”(Debug the imported project)链接来调试项目
- 自动完成的一些操作
  - 提示保存源文件
  - 构建项目（增量）
  - 启动调试器（CCS将切换到“CCSDebug”透视图）
  - 连接CCS到目标
  - 加载程序到目标（闪存）
  - 运行到主函数
- 不希望它一次做以上所有的操作？您可以配置它跳过一些步骤（调试器选项）

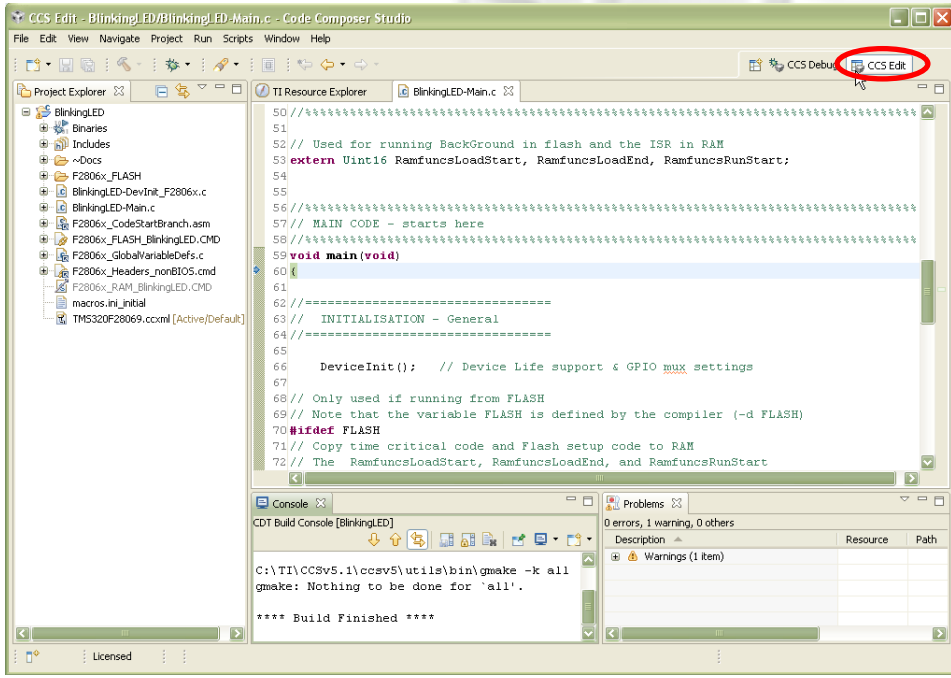


第4步：调试“闪烁LED”项目



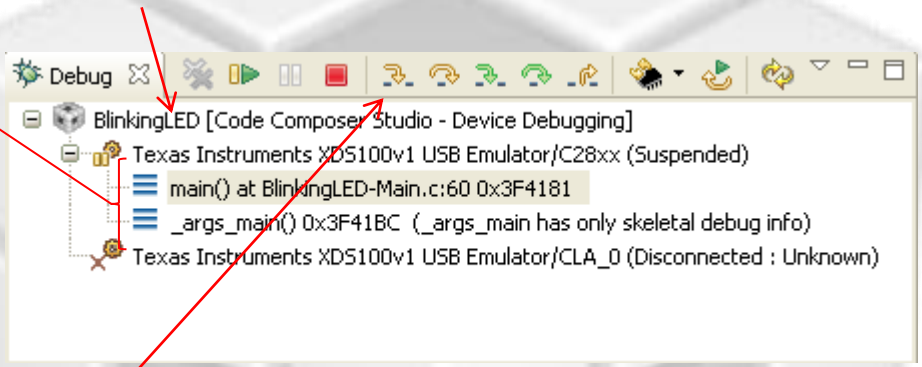
Eclipse概念： 透视图（perspectives）

- 定义工作台窗口内视图的初始设置和布局
- 在概念上类似于CCSv3“工作区”（*. WKS）（虽然可以在工作台上访问到多个透视图，但是同时间内只能展开一个）
- 每个透视图提供了一套完成一个特定任务类型的功能（“CCS Edit”用于项目开发，“CCS Debug”，用于项目调试等）
- 可以创建自定义的透视图



视图：调试

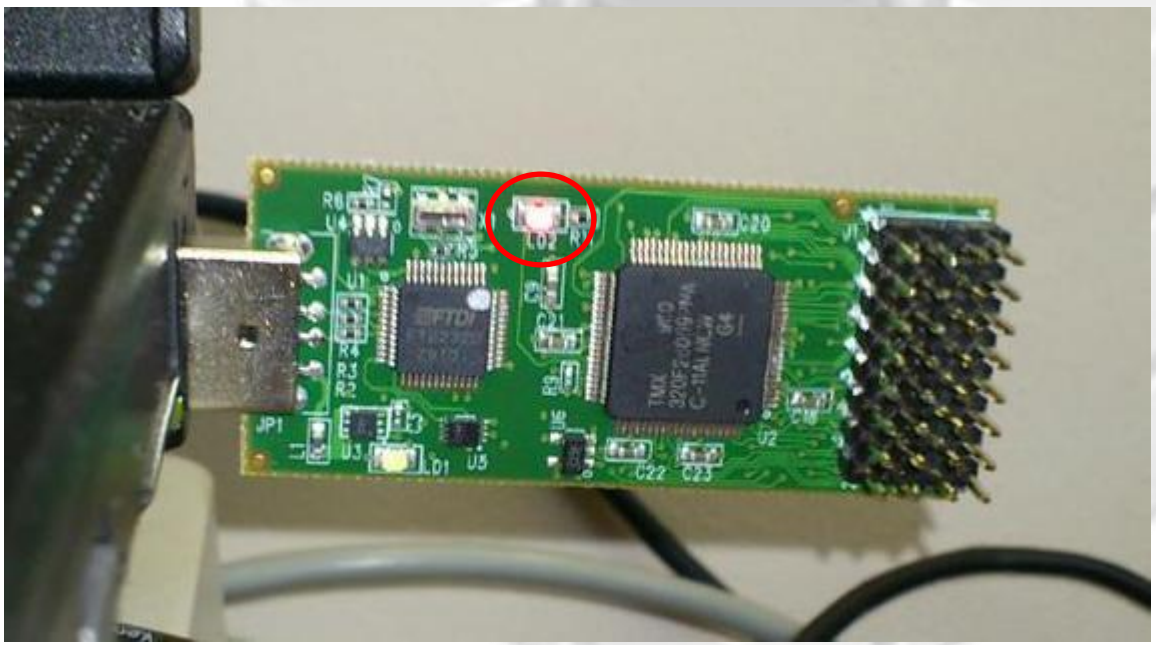
- 调试视图显示：
  - 目标配置或项目
  - 调用栈



- 按钮来运行，暂停，终止调试会话，源和ASM单步，复位CPU，重新启动程序

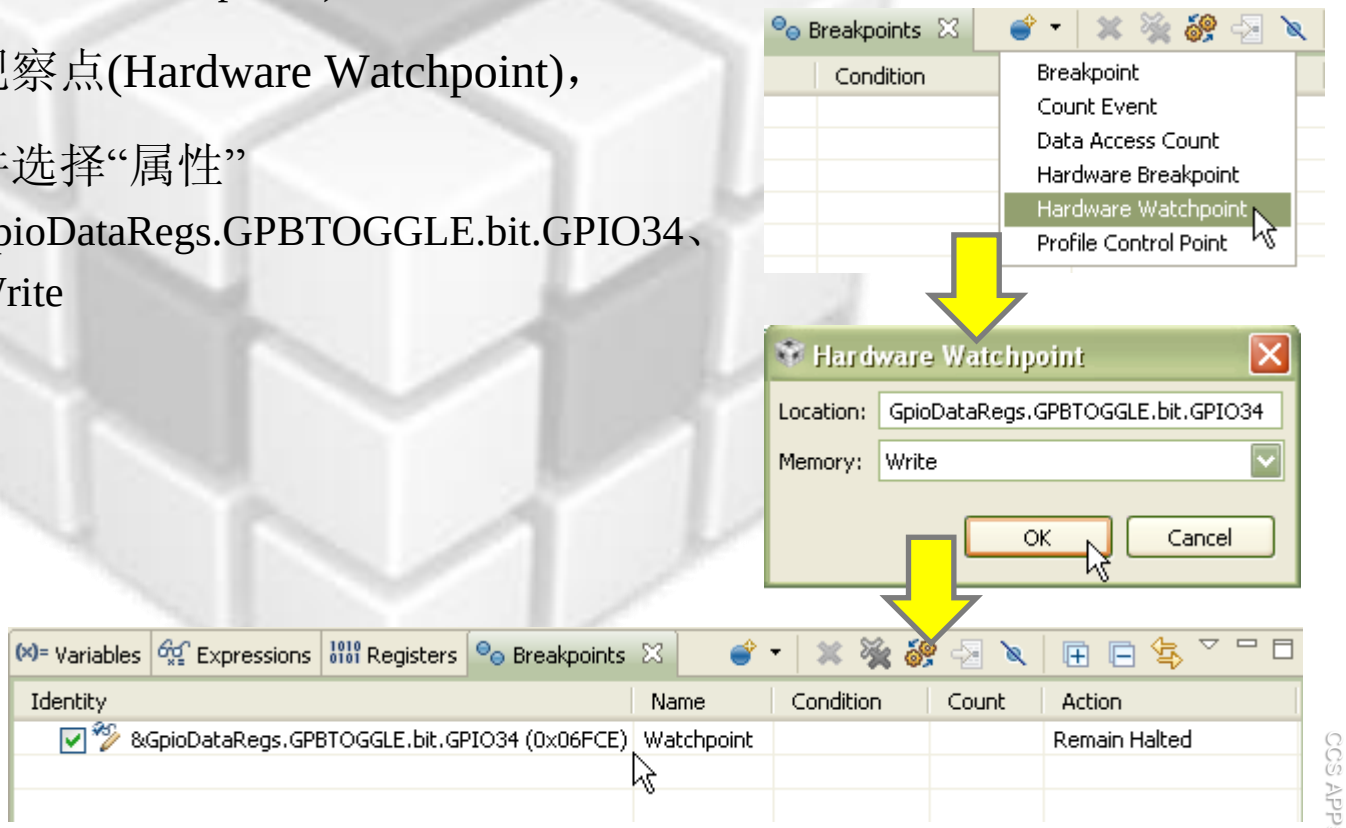
Blink LD2

- 按“运行”按钮运行程序 
  controlSTICK的LD2现在应该闪烁



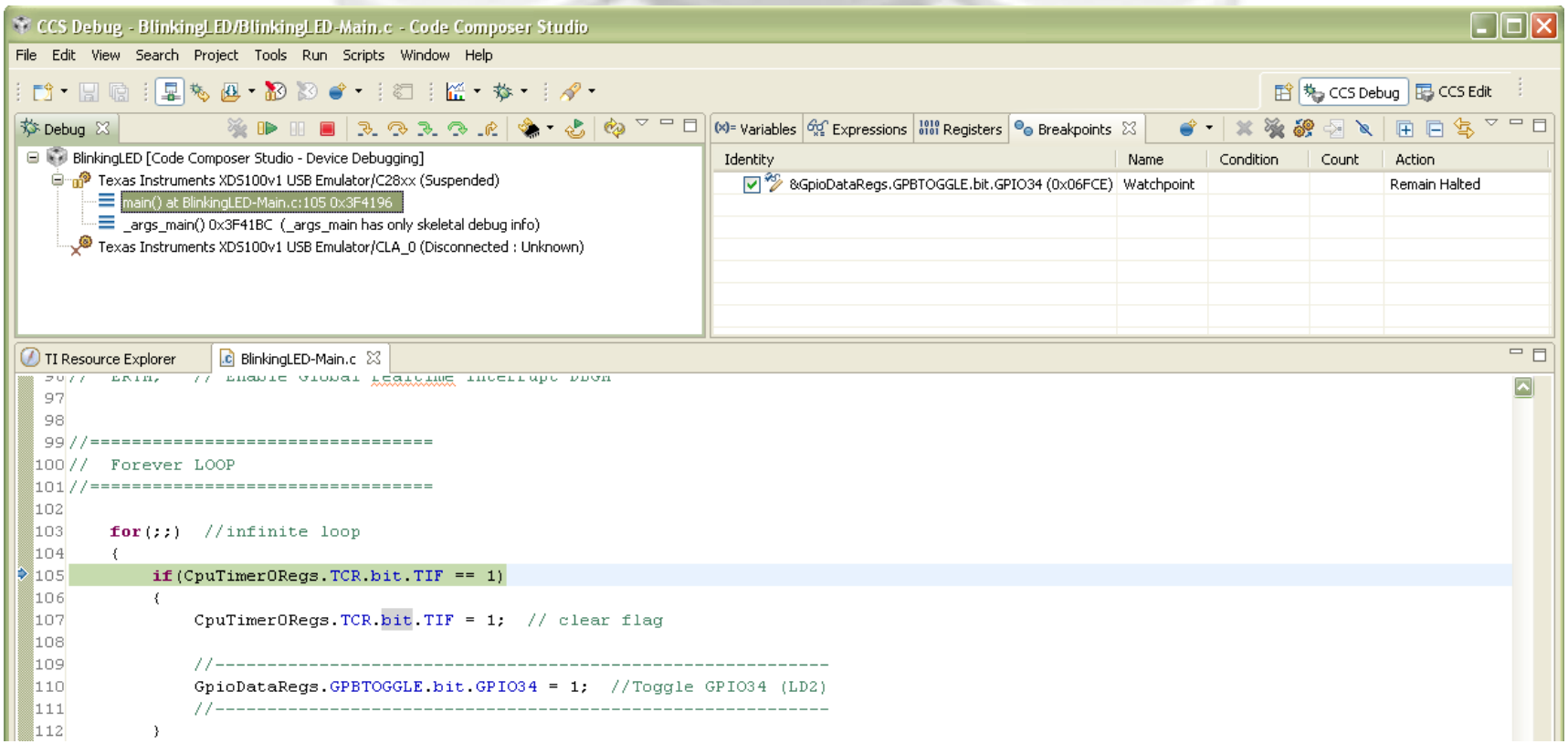
调试：使用观察点(watch points)

- 使用“停止\暂停“(halt\suspend)按钮来停止目标运行
- 打开“断点”视图
  - “视图 - >”断点“ (View -> Breakpoints)
- 创建一个新的硬件观察点(Hardware Watchpoint)，
- 右键单击观察点，并选择“属性”
  - 位置(Location)：GpioDataRegs.GPBTOGGLE.bit.GPIO34、
  - 内存(Memory)： Write



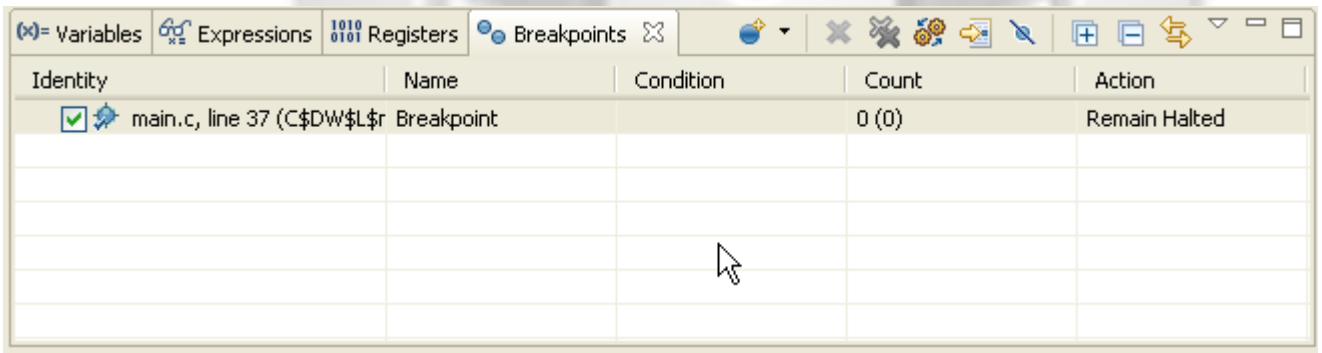
调试：使用观察点(watch points)

- 再次运行目标。指示灯LD2切换时执行会自动停止（GpioDataRegs.GPBTOGGLE.bit.GPIO34写入操作）



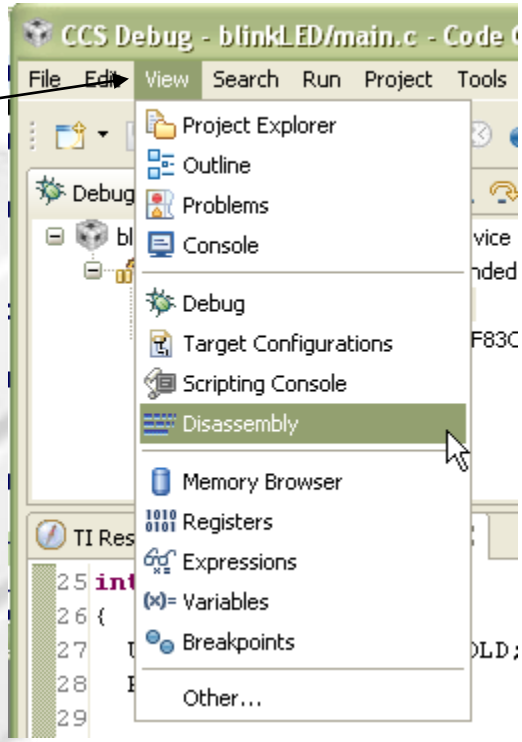
视图：断点

- 查看所有可用的断点
- 可以通过CPU将断点分组（多核设备）
- 断点触发时可以指定不同的行动
  - 刷新所有的窗口或更新一个特定的视图（取代CCS3.3中的“动画(animate)”）
  - 控制性能分析（设置分析停止/恢复点）
  - 文件的I / O（CCS3.3的探测点(probe points)）
  - 运行GEL表达式
  - 设置观察点
  - 控制CPU跟踪(trace)（在个别的ARM和DSP器件上）



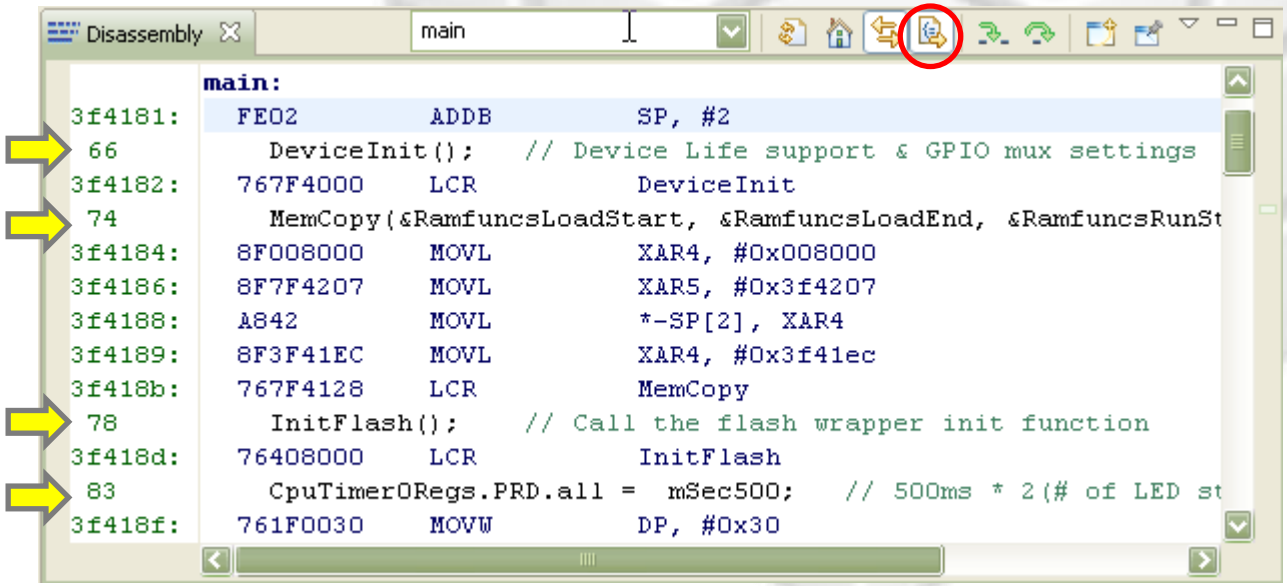
更多的调试

- 调查其他调试视图（通过“视图”(View)菜单中打开）
  - 内存浏览器(Memory Browser)
  - 寄存器(Registers)
  - 反汇编(Disassembly)（接下页）
- 设置断点
  - 双击源代码行进行设置/清除断点
  - 用“断点”(Breakpoints)视图来查看断点列表
  - 注 28x 的设备只支持两个硬件断点
    - 观察点使用一个硬件断点资源
- 可以使用“调试”(Debug)视图中按钮来进行以下操作：
  - 重新启动程序
  - 源单步
  - 汇编单步



视图：反汇编

- 在地址栏中输入“main”并按回车键，反汇编视图会找到“main”符号的地址
- 切换“显示源代码”(Show Source)按钮
- 注意反汇编码与源代码的交错

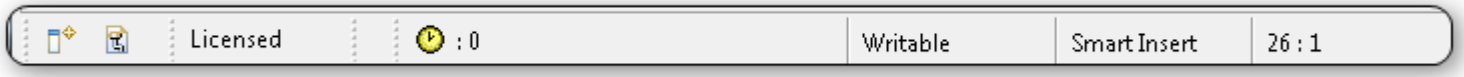


调试器选项

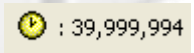
- 许多调试功能都可以通过调试器选项启用/禁用
- 调试会话中“CCS Debug” 透视图：
  - “工具 - >调试选项 - >通用的调试器选项”(‘ Tools->Debugger Options->Generic Debugger Options)
- 配置各种调试选项，例如启用/禁用：
  - 自动运行到主函数
  - 自动连接到硬件目标
  - 实时选项（需要硬件支持 例如C2000）
  - 加载程序时进行核查
  - 等等... ...
- 使用“记住我的设置(Remember My Settings)”选项来保存当前设置，方便未来使用

计数周期 - 性能分析时钟(Profile Clock)

- 性能分析时钟
  - 在大多数设备上都有，可用于计数周期
  - 在一些目标上，它可用于计数其他事件
- 时钟的启用
  - 运行 - >时钟 - >启用 (Run -> Clock -> Enable)
  - 时钟现在将显示在状态栏上



- 确认观测“GpioDataRegs.GPBTOGGLE.bit.GPIO34”写入的观察点已经启用
- 点击运行按钮来运行到这个断点
  - 时钟应该显示~40M个周期



增加LD2闪烁速率

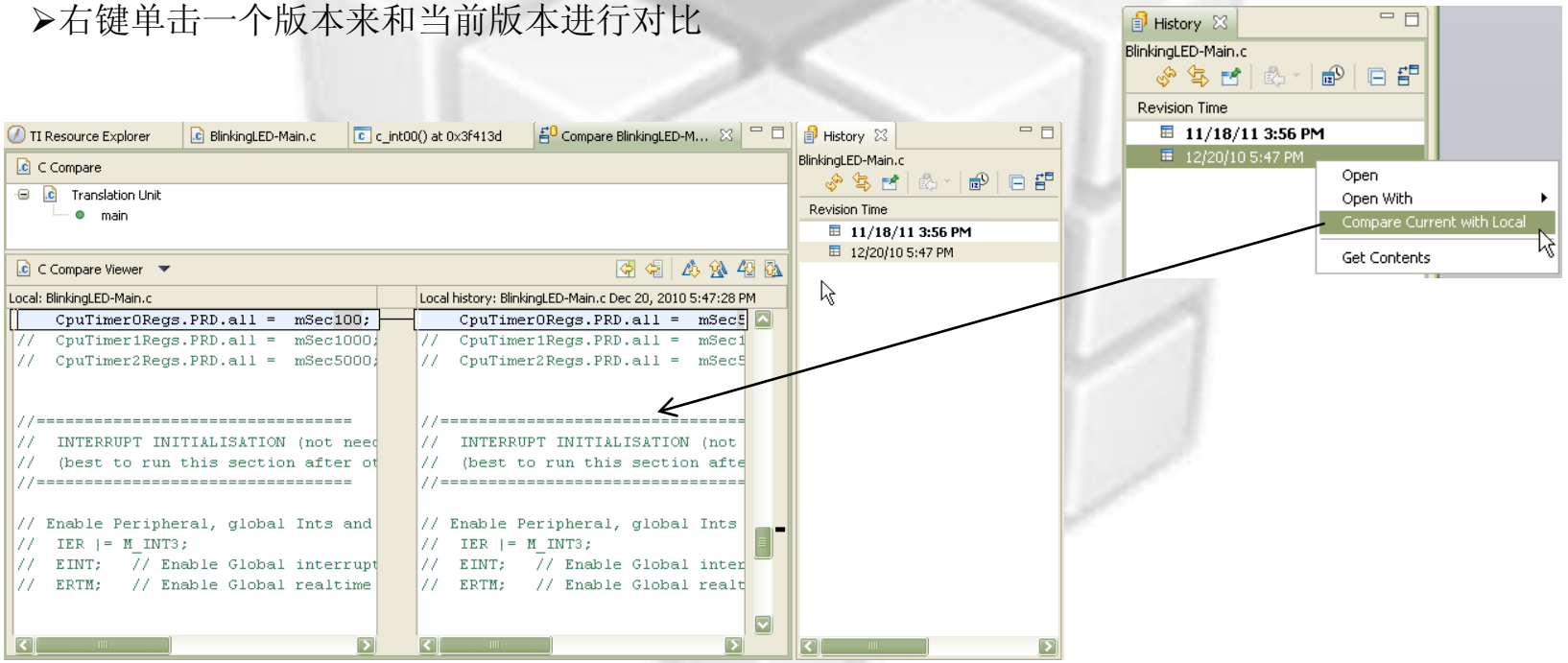
- LED L2的闪烁速率可通过改变定时器的值来提高
- 修改BlinkingLED- MAIN.C83行
  - 从：CpuTimer0Regs.PRD.all= mSec500;/ /500毫秒* 2（LED状态＃）=1秒闪烁速率
  - 到：CpuTimer0Regs.PRD.all= mSec100;/ /100毫秒* 2（LED状态＃）=200毫秒的闪烁速率
- 保存文件

TI Resource Explorer | BlinkingLED-Main.c | c_int00() at 0x3f413d

```
77// This function must reside in RAM
78    InitFlash();    // Call the flash wrapper init function
79#endif //(FLASH)
80
81// Initialise Period of Cpu Timers
82// Timer period definitions found in PeripheralHeaderIncludes.h
83    CpuTimer0Regs.PRD.all =  mSec100;   // 100ms * 2(# of LED states) = 2
84//  CpuTimer1Regs.PRD.all =  mSec1000;
85//  CpuTimer2Regs.PRD.all =  mSec5000;
86
87
```

视图：本地历史

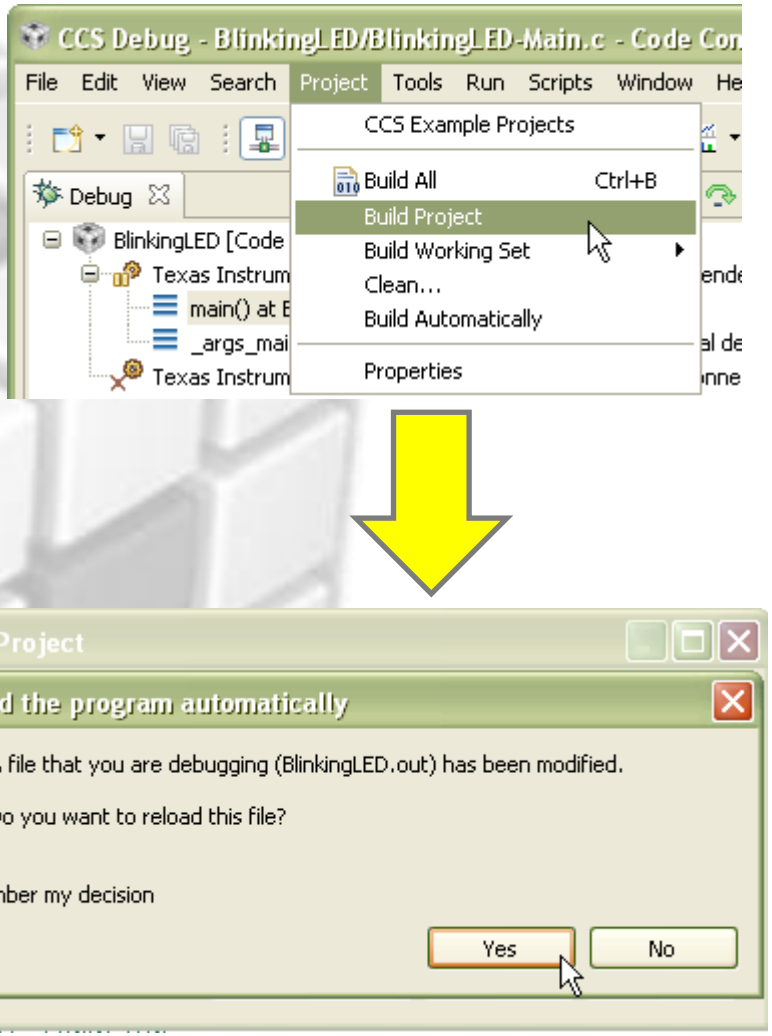
- CCS保留了本地源的变化历史
  - 切换到的CCS编辑的透视图
  - 在编辑器中右键单击一个文件，选择“团队 - >显示本地的历史”(Team -> Show Local History)
- 您可以把当前的源文件对任何以前的版本作比较或回滚到以前的版本
  - 在编辑器中双击一个版本将其打开
  - 右键单击一个版本来和当前版本进行对比



- CCS还保留项目历史
  - 恢复从项目中删除的文件
  - 右键单击该项目，并在菜单中选择“从本地历史恢复”(Recover from Local History)

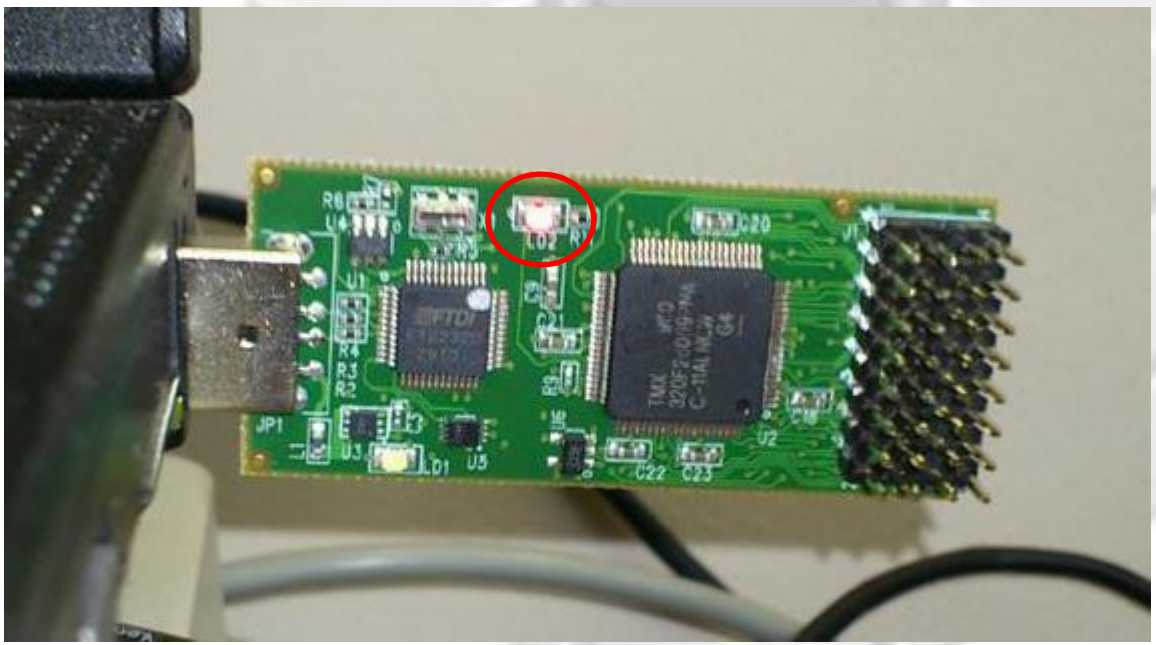
重建项目

- 切换到“CCS的调试”的透视图
- 重建项目
  - 项目 - >构建项目(Project -> Build Project)
- CCS会自动检测目前正在调试的程序是否已经改变/重建，并询问是否要重新加载该文件
  - 如果选择‘Yes’，CCS将重新加载/烧录程序，并运行到主函数



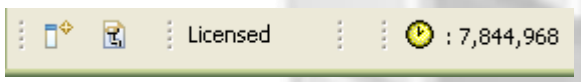
Blink LD2

- 按“运行”按钮运行程序 
  controlSTICK上的LD2现在应该以更快的速度在闪烁



计数周期（性能分析时钟） - 第2部分

- 暂停执行程序
- 双击在状态栏上的时钟图标重置其为0
- 启用在'GpioDataRegs.GPBTOGGLE.bit.GPIO34'的观察点
  - 观察点在程序重载时复位
- 点击“运行”按钮
  - 时钟现在应该显示一8M周期（40M的1 /5 ）

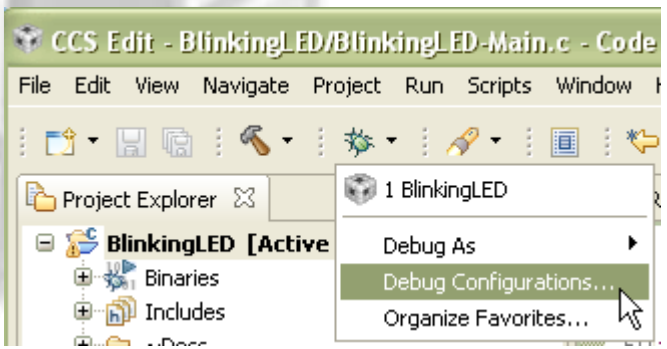


终止调试会话

- 进入到调试视图
- 点击“终止”按钮
- 这项操作会关闭调试器，返回到CCS编辑透视图

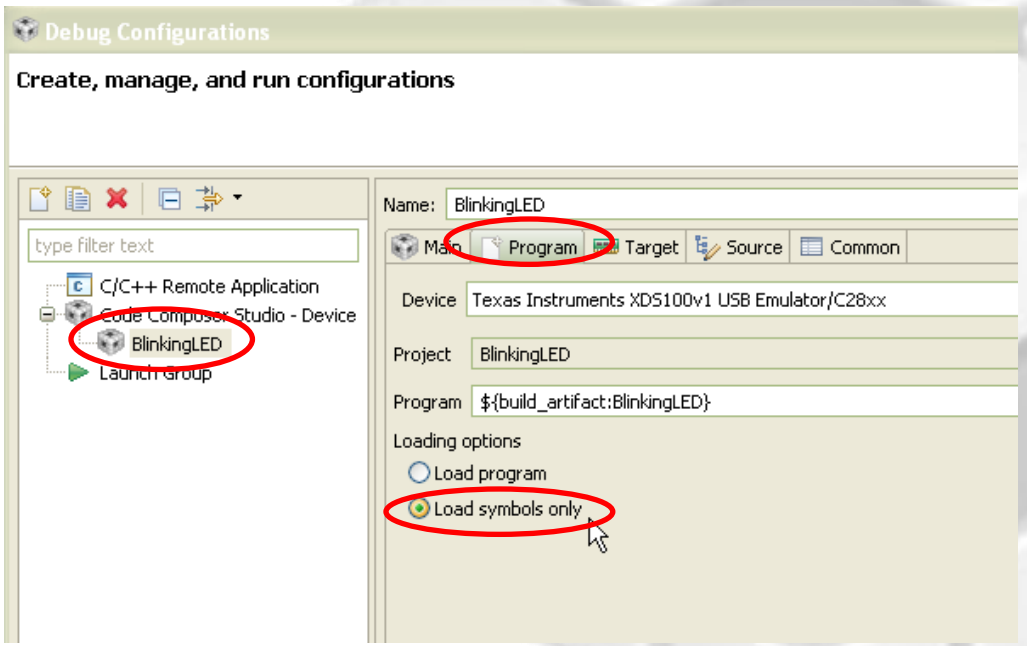
加载闪存上的程序符号

- 如果程序已经通过CCS烧录成功，你只想调试目标上闪存里现有的代码，你可以配置CCS只加载项目符号来进行调试
- 选择“调试”按钮旁边的下拉菜单，然后选择“调试配置..”
- Eclipse概念：调试配置- 当一个项目或目标配置第一次启动调试会话后所创建的缓存信息。缓存的内容包括因该使用的目标配置，调试设置...



加载闪存上的程序符号

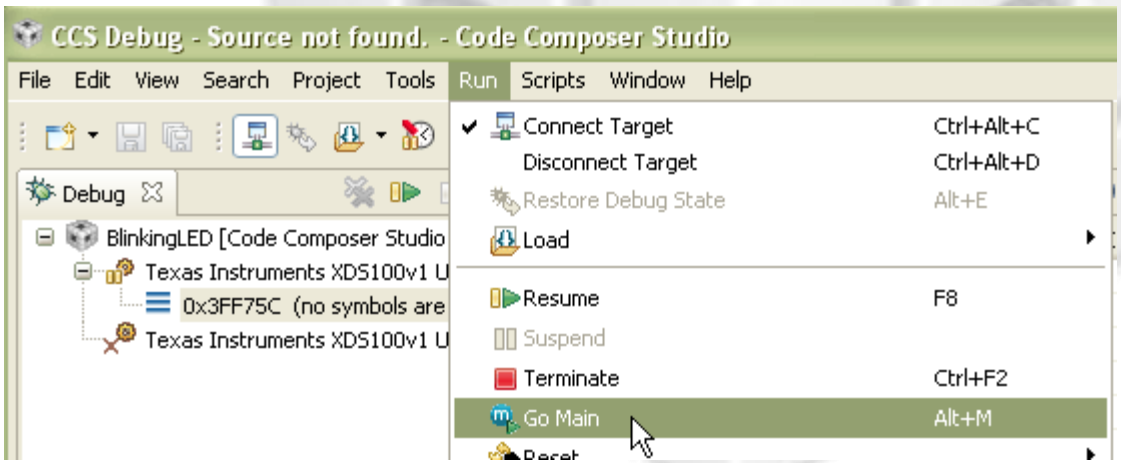
- 在左面板中选择标签 “BlinkingLED”；右面板中选择标签 “Program”
  - 在“载入选项”(Loading options)中，选择“只加载符号”(Load symbols only)



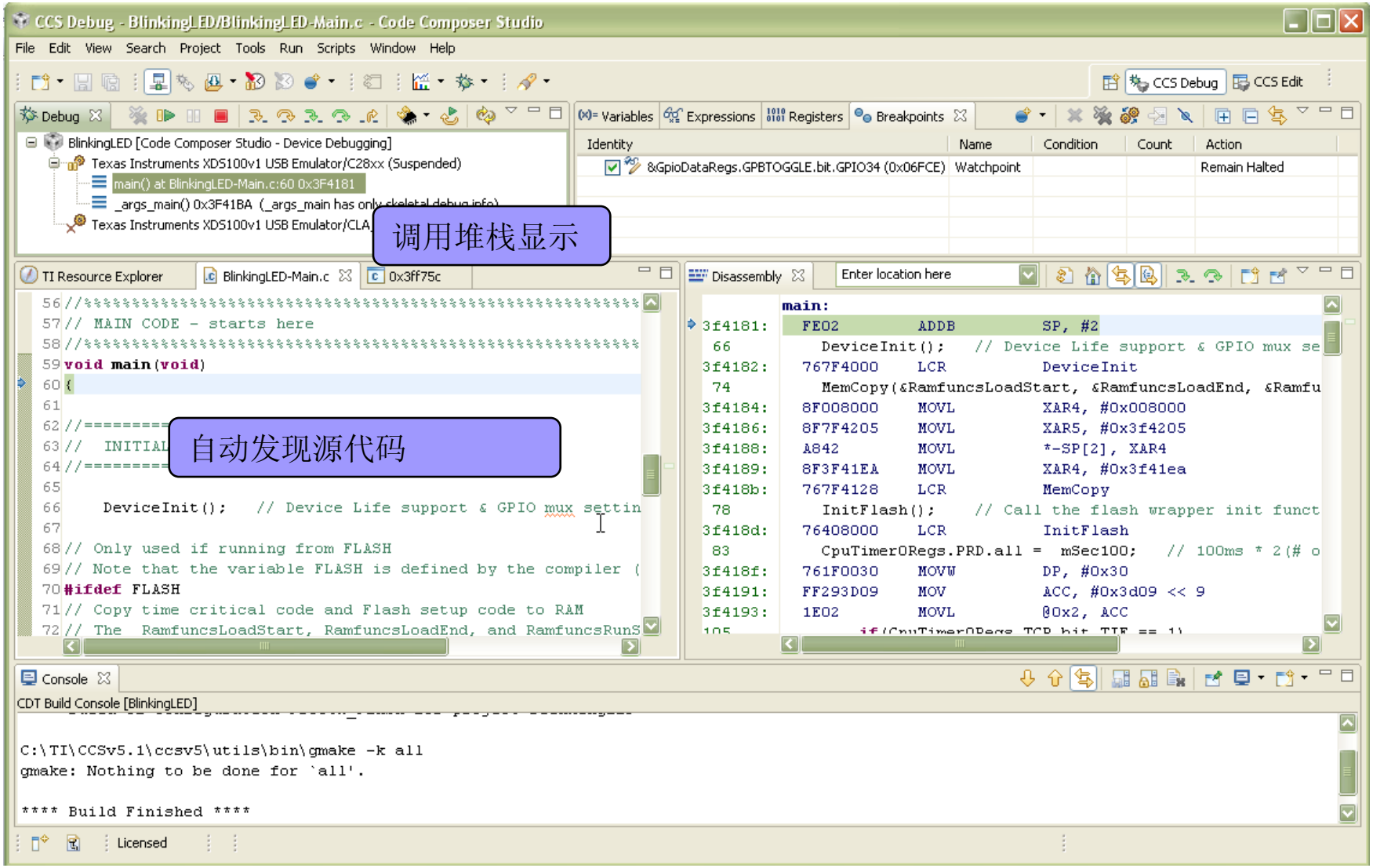
- 然后选择“应用”，然后“调试”

加载闪存上的程序符号

- 调试器将启动，连接到目标，并只加载符号来调试程序（没有对目标加载程序/烧录的操作）
- 程序计数器(program counter)将被设置在代码的入口点，而不是在主函数
  - “运行 - >运行到主函数” (Run->Go Main)



加载闪存上的程序符号



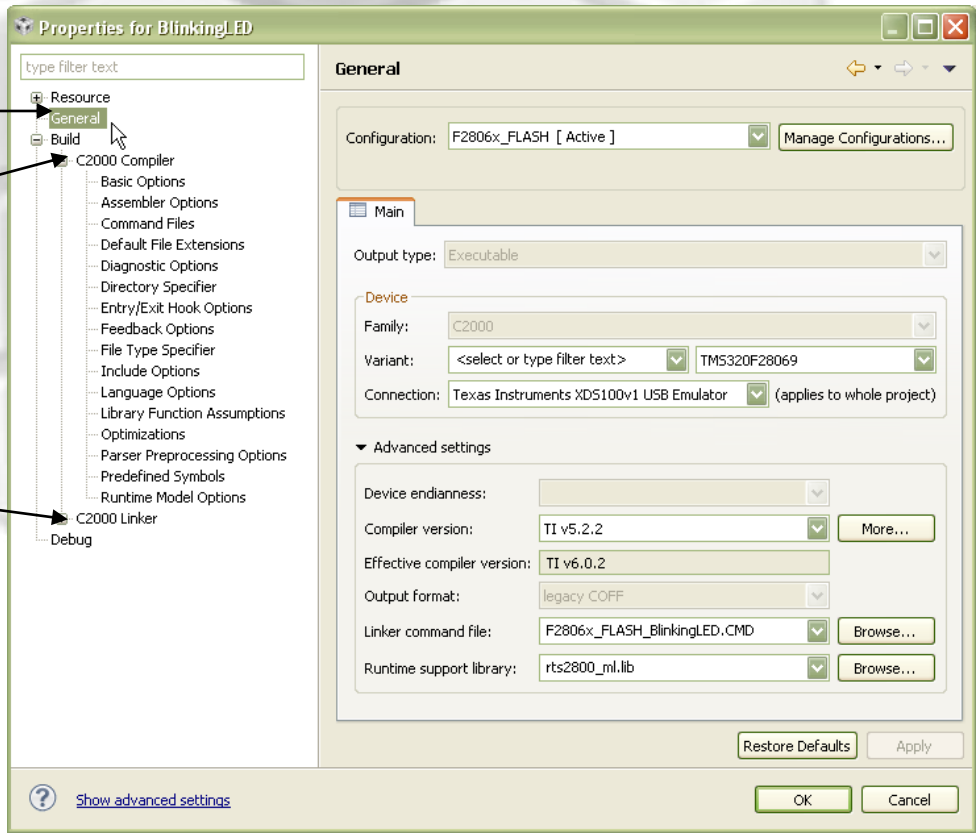
更改构建选项(build options)

- 终止调试会话
- “BlinkingLED”项目上右击，并选择“属性”

设备和高级的设置

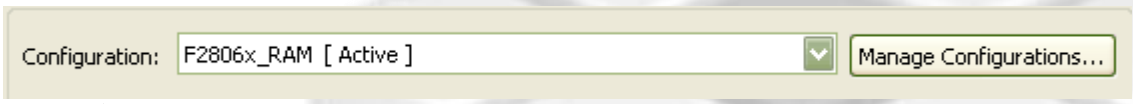
编译器选项

链接器选项

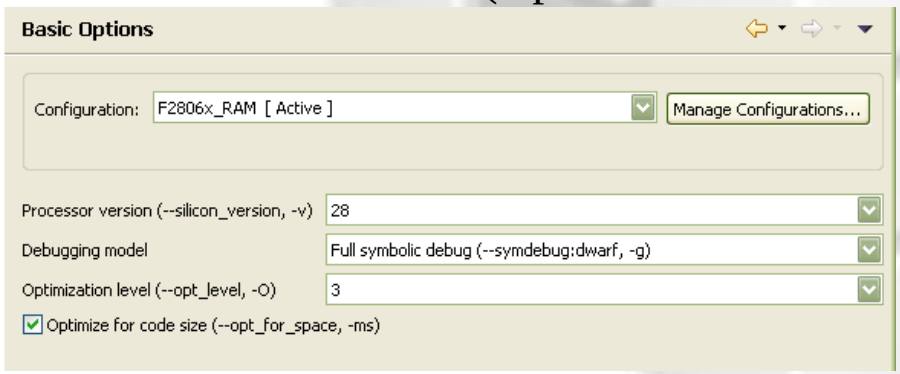


更改构建选项

- 每套构建选项对应一个构建配置(build configuration)
  - 更改您的配置为“F2806x_RAM”



- 更改优化设置
  - 基本选项(Basic Option)组
  - 更改优化级别为3。
  - 启用“优化代码大小”(Optimize for code size)



- 单击“确定”

更改构建选项

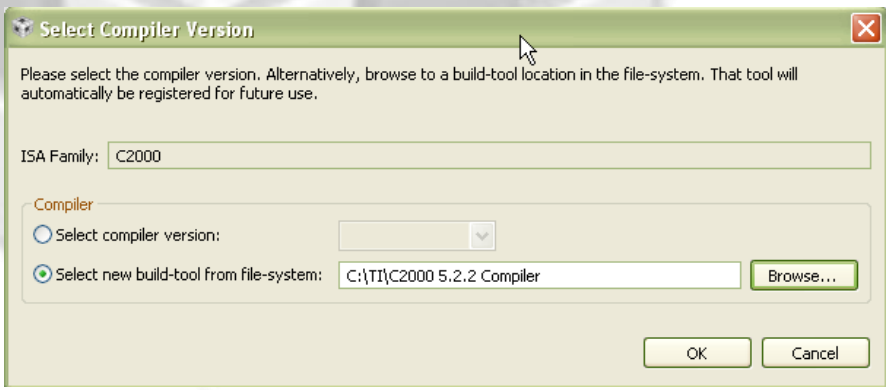
- 更改激活配置为“F2806x_RAM'
  - 右键单击项目
  - 选择构建配置 - >设置激活 - > F2806x_RAM (Build Configurations -> Set Active -> F2806x_RAM)
- 点击“构建”(build)按钮
  - 在控制台视图中，您会看到“F2806x_RAM”配置已建成
- 您还可以更改配置和然后构建：点击“构建”按钮旁边的箭头，然后选择你想用的配置
  - 选择“F2806x_RAM”，它会构建F2806x_RAM配置
  - 激活配置由复选标记表示

编译器版本

- 每个项目都需要指定使用的编译器版本
  - 实际上这是在每一个配置级别上的设置
  - i.e.调试版可以使用一个版本，发行版另一个
- 如果你想使用一个新下载的编译器，你必须改变你的项目所指定的编译器版本
- CCS将允许您选择本地计算机上能找到的所有编译器，
- 当您通过更新管理器安装在一个新的编译器，CCS会自动获得该编译器的地址
  - 然而，你也可以告诉CCS一个版本的编译器的地址

改变编译器的版本

- 在您的项目上右击并选择“属性”(Properties)
- 点击“一般设置“(General)
- 点击编译器的版本:“TI 5.2.2”旁边的“更多”(More)按钮
- 浏览到在下图对话框中显示的路径，并单击“确定”
- CCS将找到路径下的编译器，并把它选择至当前配置。您会看到，现在指定的编译器是TI 5.2.2
- 构建您的项目



目标配置
TARGET CONFIGURATION

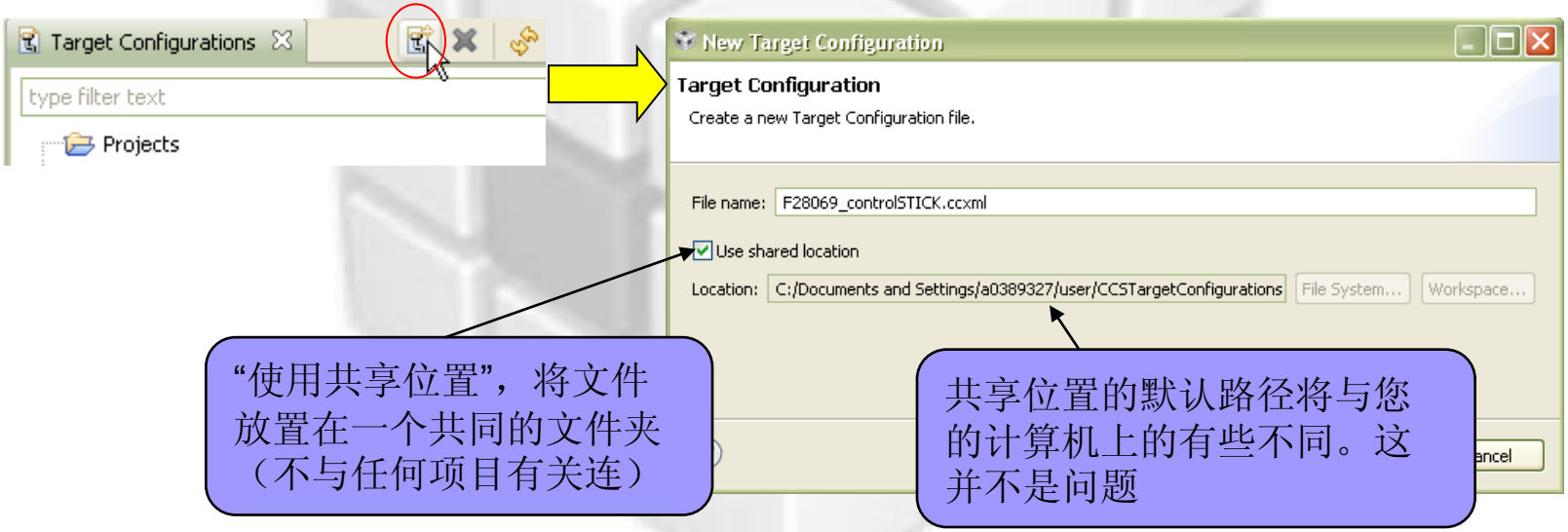
目标配置文件 - 基础知识

- 目标配置文件是定义设备与连接的XML文件（文件扩展名 *. CCXML）
  - 等于CCS3.x的配置文件（*. CCS）
- “目标配置(Target Configurations)”的视图是用于管理和维护目标配置文件
- 目标配置编辑器(Target configuration editor)用于创建/修改目标配置文件
- 基本(Basic)选项卡是为了大多数用户使用
- 高级(Advanced)选项卡是为了调整默认属性，初始化脚本，或建立新的板/设备的目标配置

创建目标配置

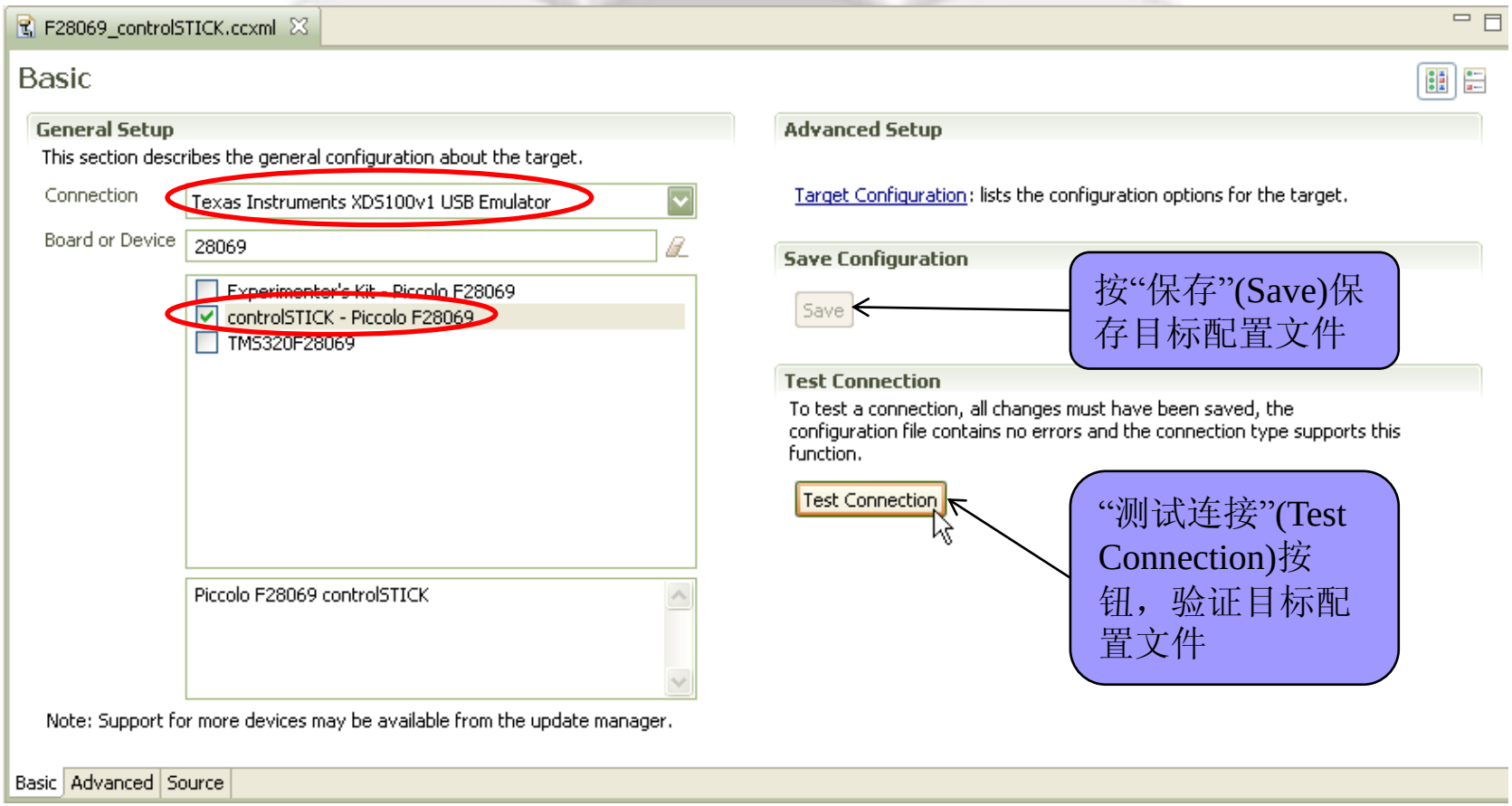
- 打开“目标配置”(Target Configuration)的视图
  - 视图- >目标配置 (View -> Target Configurations)

- 创建一个新的配置：



创建目标配置

- 在“基本”标签，选择设置：
  - 连接(Connection)：Texas Instruments XDS100v1 USB Emulator
  - 设备(Device)：controlSTICK - Piccolo F28069

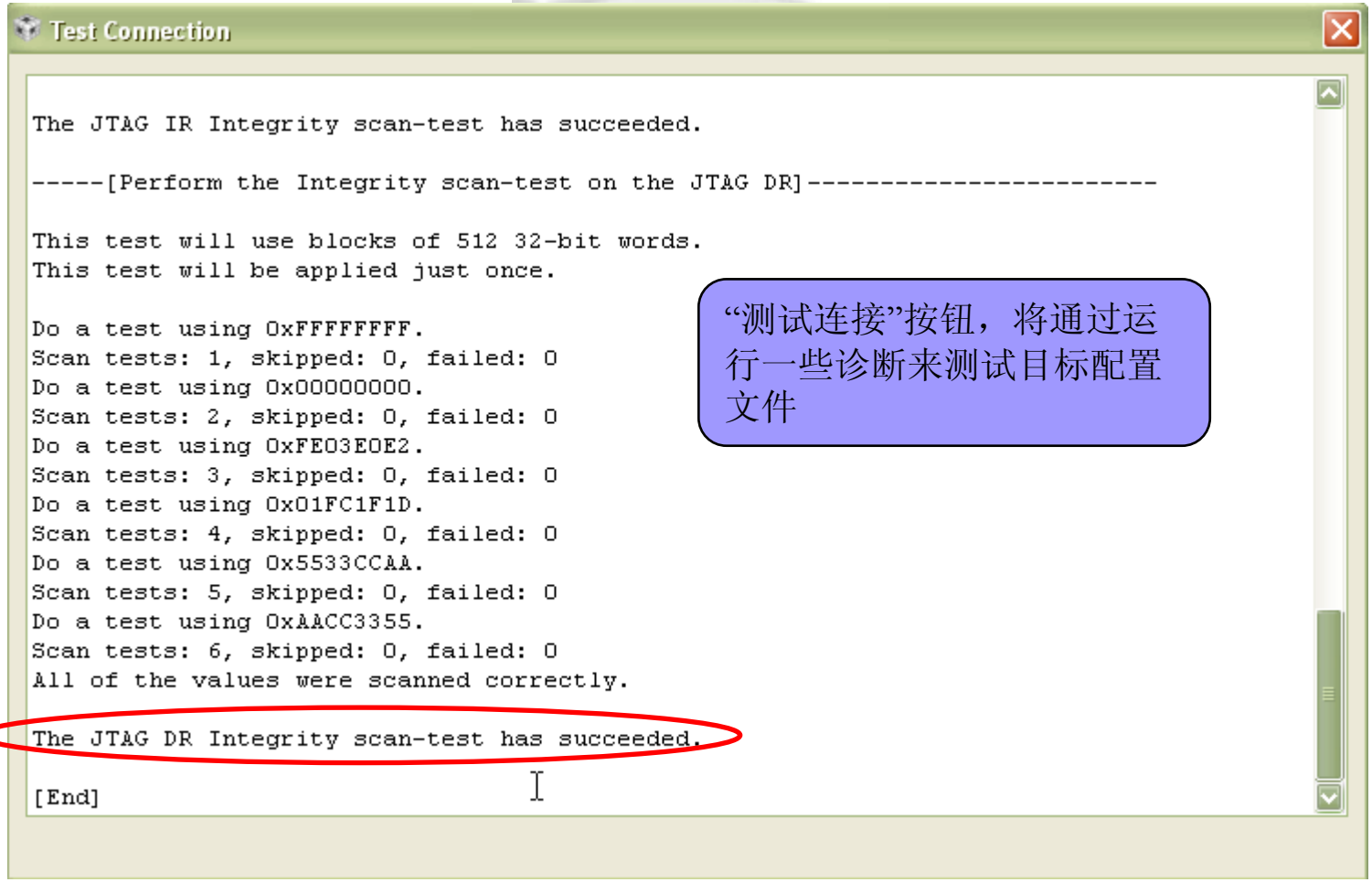


目标配置 - 高级

Test Connection

To test a connection, all changes must have been saved, the configuration file contains no errors and the connection type supports this function.

Test Connection



“测试连接”按钮，将通过运行一些诊断来测试目标配置文件

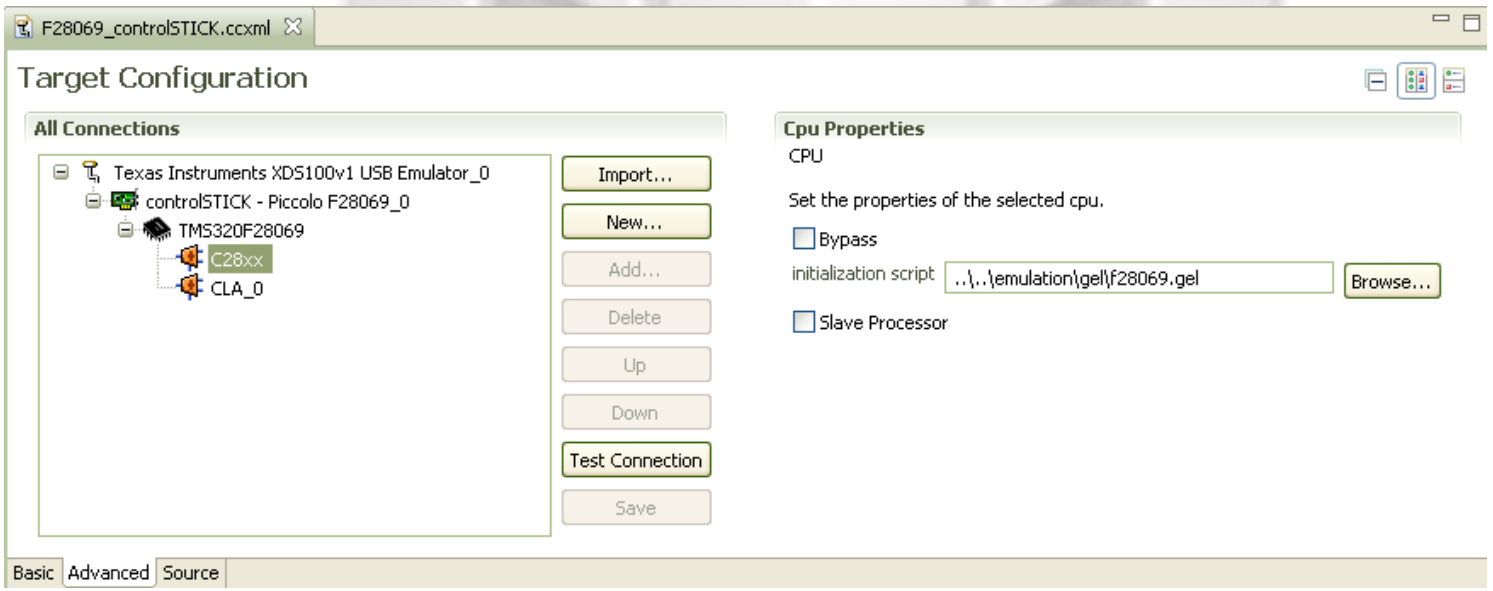
目标配置 - 高级

- 如果您从“基本”选项卡中找不到您的模拟器/目标，请使用“高级设置”(Advanced Setup)
  - 从“连接”('Connections)列表中选择指定的连接类型
  - 然后从组件列表中选择指定的组件（“设备(Devices)”，“处理器(CPUs)”，“路由器(Routers)”）添加到连接
- 使用“高级设置”，您可以创建一个使用两个仿真器的目标配置
- 若想正确的使用“高级设置”建立配置必须具备深入的相关目标设备的知识

目标配置 - 高级

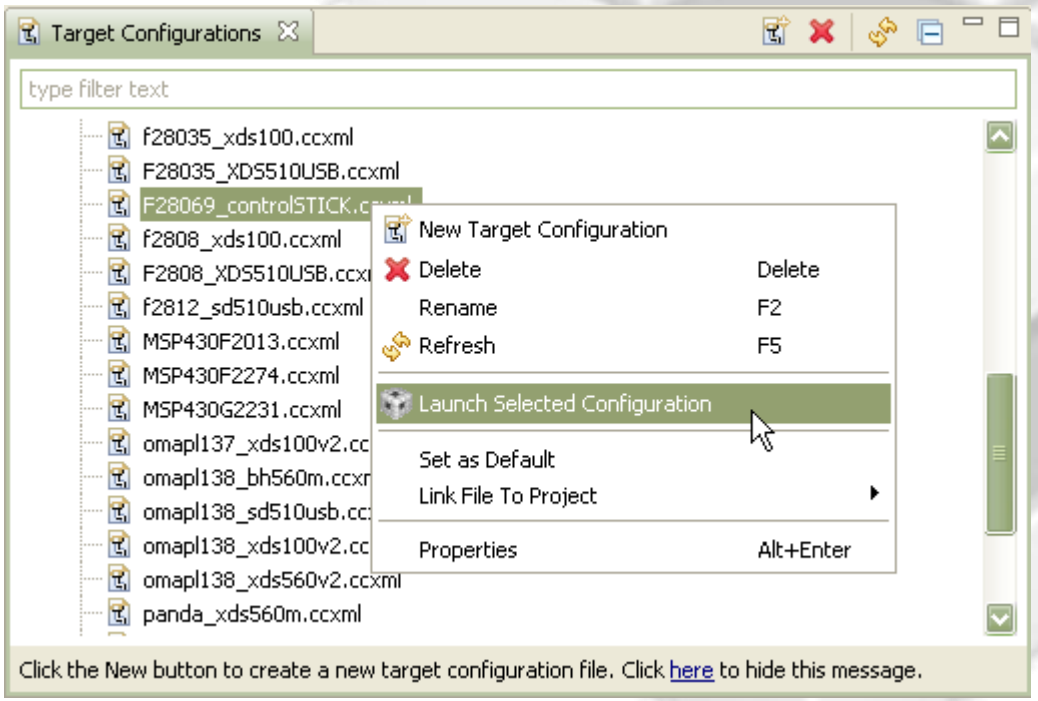
- 调整目标配置的默认属性：
  - 指定初始化文件（GEL启动文件）
  - 指定IcePick子路径的端口号
  - 绕行(bypass)CPU
  - 设置JTAG TCLK频率
  - 等等... ...

类似CCS3.x CCS设置工具



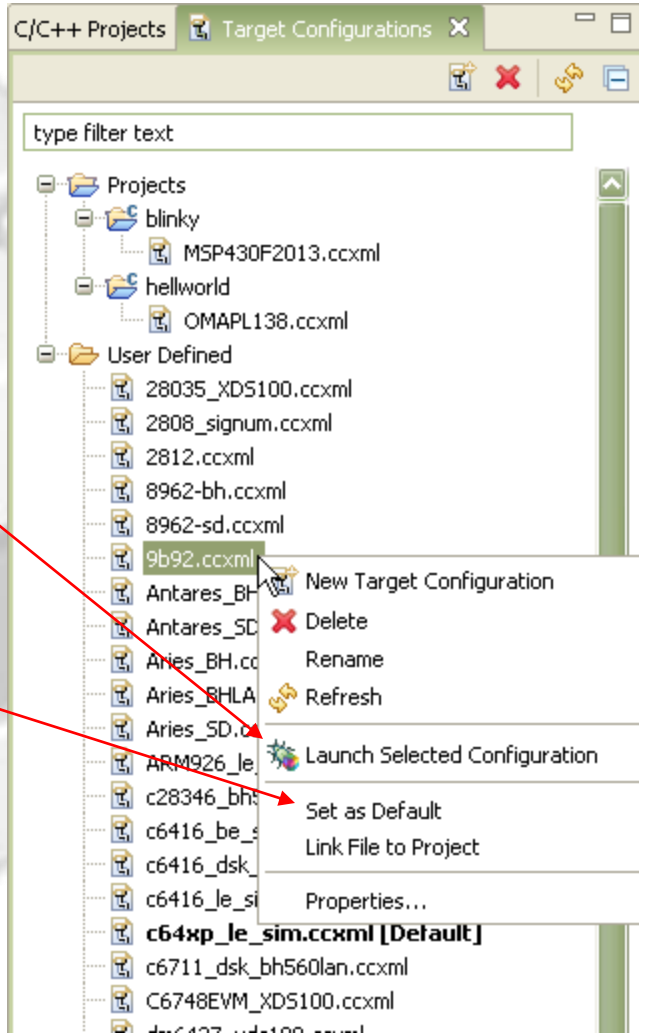
启动调试会话(Debug Session)

- 右键单击新创建的目标配置文件，并选择“启动选定的配置”(Launch Selected Configuration)启动调试会话



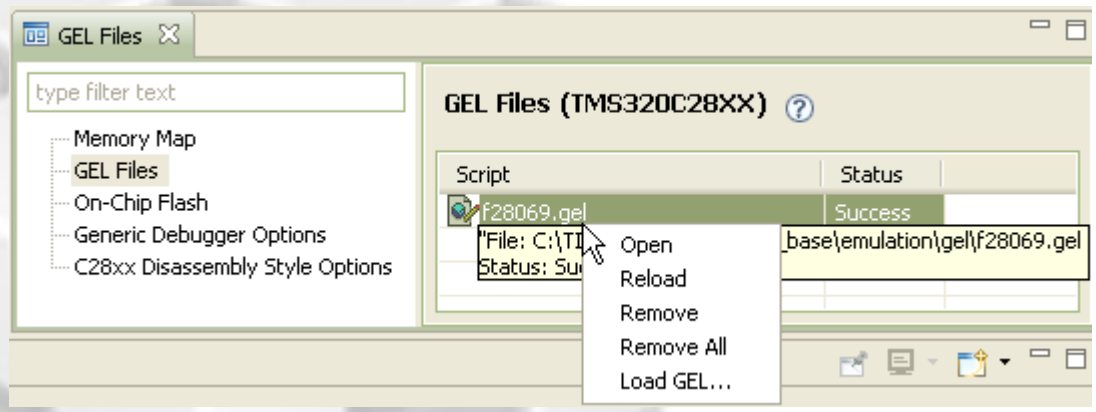
视图:目标配置

- 您可以轻松地打开，复制和删除目标配置（XML文件）
- 快速启动一个调试会话：右键单击目标配置，并选择“启动选定的配置(Launch Selected Configuration)”
- “启动TI调试器(Launch TI Debugger)”将使用目标配置视图里，[**Default**]（默认）选项卡中选定的目标配置
  - 右键点击文件并选择“设置为默认(Set as Default)”，使其成为默认目标配置文件
  - 如果没有项目指定的目标配置文件,“调试激活项目(Debug Active Project)”将使用默认目标配置

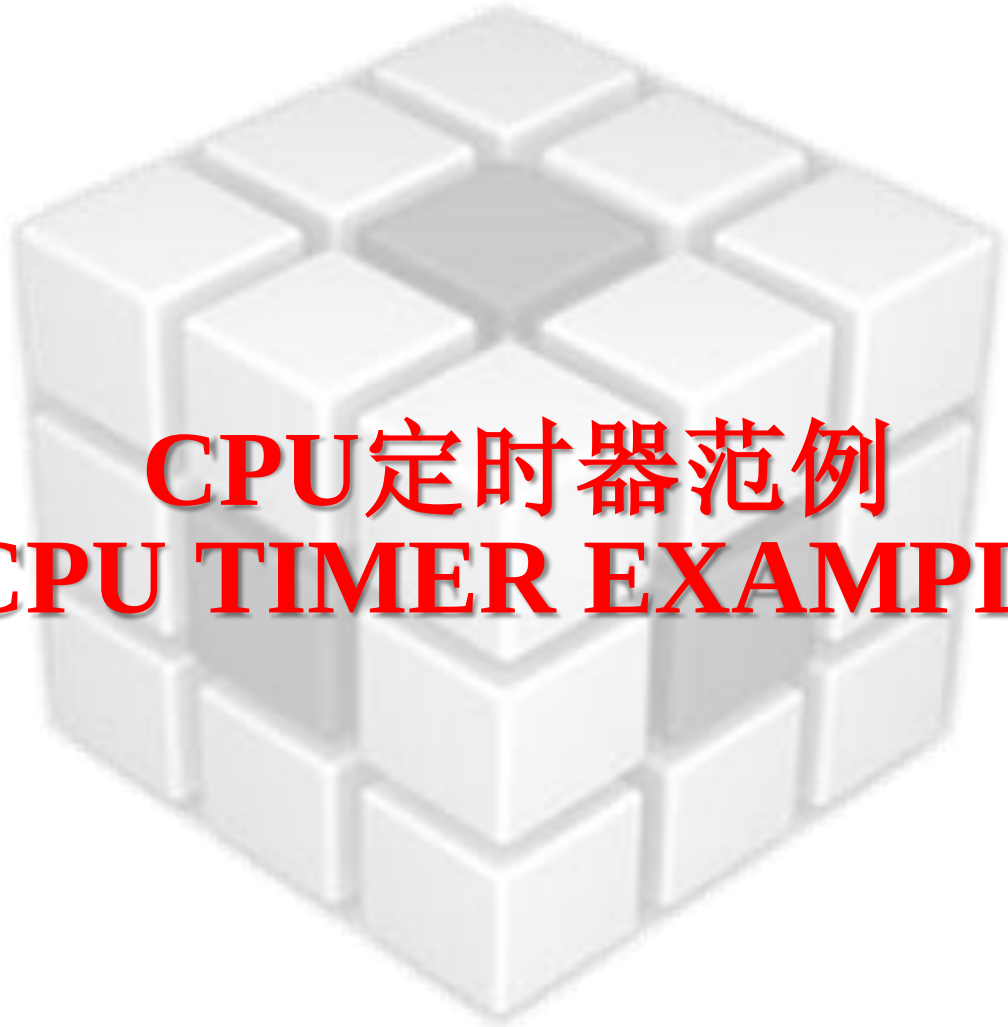


视图：GEL文件

- 显示当前的调试会话的所有加载的GEL文件和其当前的状态
- 可行的操作：
  - 在编辑器中打开文件
  - 刷新（重新加载）文件
  - 删除/卸载文件
  - 删除/卸载所有文件
  - 加载额外的GEL文件



- 新加载的GEL文件会在视图中出现
- 如果加载没有问题，状态将显示成功



CPU定时器范例
CPU TIMER EXAMPLE

CPU定时器范例：练习概要

- 主要目标
  - 基于CPU定时器的例子，创建一个新的可移植项目
  - 构建程序
  - 启动调试会话和加载程序到controlSTICK上
  - 运行程序
  - 启用实时模式来观察变量的实时更新
- 工具和概念
  - 可移植项目
  - 链接的资源
  - 链接的资源路径变量
  - 构建变量
  - 实时模式

创建一个新的可移植项目：简介

- 主要目标
  - 为项目创建工作区级变量
  - 创建一个新项目
  - 使用变量把文件链接到项目中
  - 使用变量配置构建属性
  - 通过项目构建，加载和运行来验证程序

共享项目
SHARING PROJECTS

共享项目

- 共享“简单”的项目（所有的源/头文件都在项目文件夹中）
- 共享包含“链接文件”(linked file)的项目和所有的源（项目使用链接文件）
- 共享只有“链接文件”的项目（没有源）
  - 只共享项目文件夹（项目的接收人已经有所有源）
  - 使项目“可移植”(portable)
    - 消除项目中的绝对路径

共享项目 - 简单的项目

- **使用案例：希望分享一个项目文件夹和需要构建该项目的所有源文件。所有的源文件都在项目文件夹内。**
- 没有链接文件的项目共享起来很容易：
  - 可以原封不动的发布整个项目的文件夹给各个对象
  - 收到项目的用户可以导入该项目到他们的工作区：使用“项目 - >导入现有的CCE / CCS项目”(Project -> Import Existing CCE/CCS Project)，并选择收到的项目文件夹
  - 适用于简单的项目，所有引用的文件都在项目文件夹内

共享项目 - 使用链接文件的项目（1）

- **使用案例：希望分享一个项目文件夹和需要构建项目的所有源文件。一些（或全部）项目的源文件是通过链接所指定的**
- 使用CCS导出项目(Export Project)，以创建一个归档文件（zip）。该文件包含项目的文件夹以及所有项目使用的链接源文件
  - 注意，如果项目中使用链接路径变量(Linked Path variables)的话，这种方法将失败！
- 导出您的项目：如何在源计算机上归档您的项目
  - 文件 - >导出 (File -> Export)
  - 展开“常规(General)”，然后选择“归档文件(Archive File)”。点击“下一步(Next)”
  - 弹出的对话框将允许您选择需要被导出到归档的项目。
  - 当您选择一个项目时，右边图框中会显示的所有会被出口的组件。点上项目旁边的复选框来指示希望出口该项目
  - 指定归档的名称，并选择ZIP或TAR，然后点击“完成”
  - CCS会在您的计算机上生成一个包含该项目的zip/tar文件。它包含了项目目录中的文件和链接/引用的所有资源

共享项目 - 使用链接文件的项目（2）

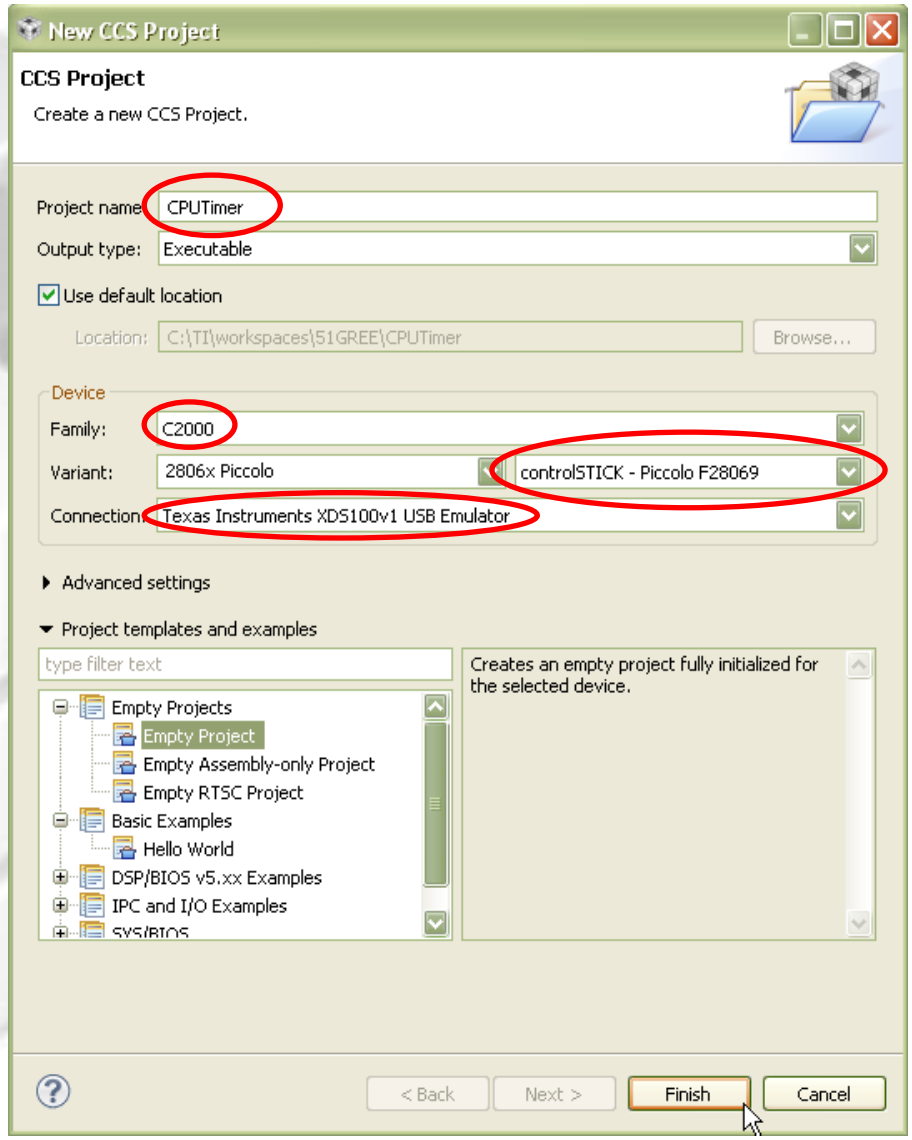
- 在另一台机器上导入项目：如何从归档导入项目-将项目导入到工作区
- 项目 - >导入现有的CCS/ CCE的Eclipse项目
  - Project -> Import Existing CCS/CCE Eclipse Project
  - 更改在顶部的单选按钮选择“选择归档文件(Select archive file)”
  - 浏览并选择您的归档文件
  - 它会列出所有在归档中发现的项目。默认情况下是全选；选择您想要导入的项目
  - 点击“完成”
  - 项目会被导入您目前在的工作区
- 链接/引用的资源将会从归档中被拷贝到相对于原来的计算机上同样的路径

共享项目 - 使用链接文件的项目（3）

- 使用个例：
  - 共享一个项目文件夹。项目的接收人已经有了所有源文件的副本
  - 需要把项目文件夹/文件等提交到源代码控制工具
- 大多数个例只是需要共享项目，而不是相关的所有源文件包
  - 用户有源文件的本地副本
- 需要使项目可移植化，以确保该项目能够被轻松共享
- 可移植项目必须避免任何绝对路径
- 理想的可移植项目必须能在不修改任何项目文件的情况下也能使用
  - 这对由在源代码控制工具上维护的项目是理想选择

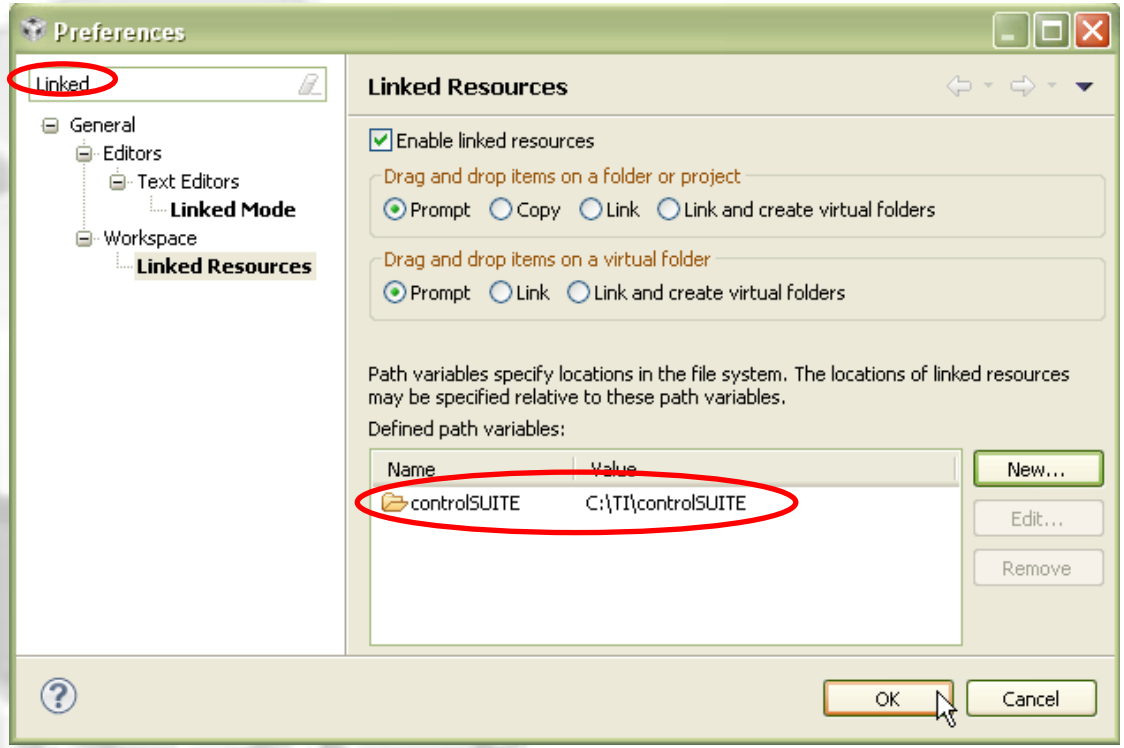
CPU定时器：创建一个新项目

- 使用“新的CCS项目”(New CCS Project)向导
  - 在“欢迎”(Welcome)页面选择“新建项目”(New Project)
- 如右图所示填写
- 完成后选择“完成”
- 生成的项目将出现在项目资源管理器(Project Explorer)视图
- 从项目中删除生成的“main.c”文件



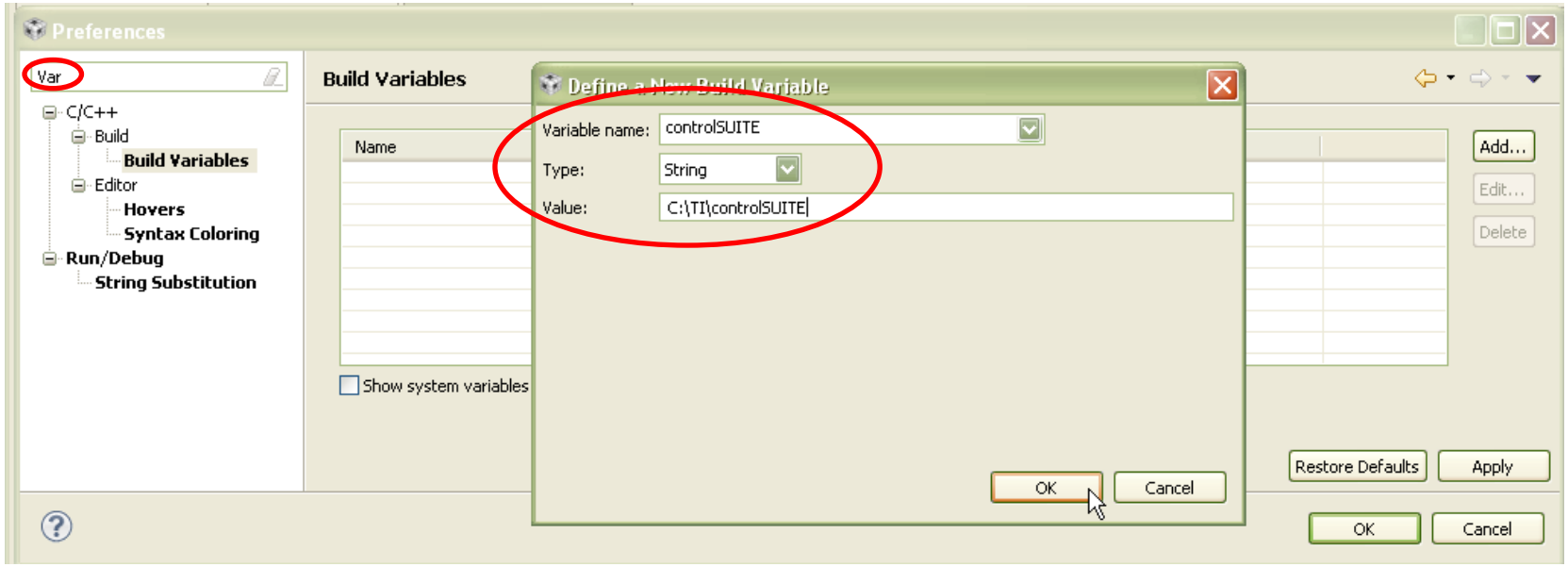
CPU定时器：创建一个链接资源路径变量

- 打开工作区的首选项(Preferences)
  - 窗口 - >首选项
  - Window -> Preferences
- 打开“链接资源”(Linked Resource)首选项
  - 在过滤器输入”Linked”，以便查找
- 使用“新建”(New)按钮来创建一个“链接资源的变量”(Linked Resource Variable)，指向controlSUITE根目录的位置
- 完成时，点击“OK”



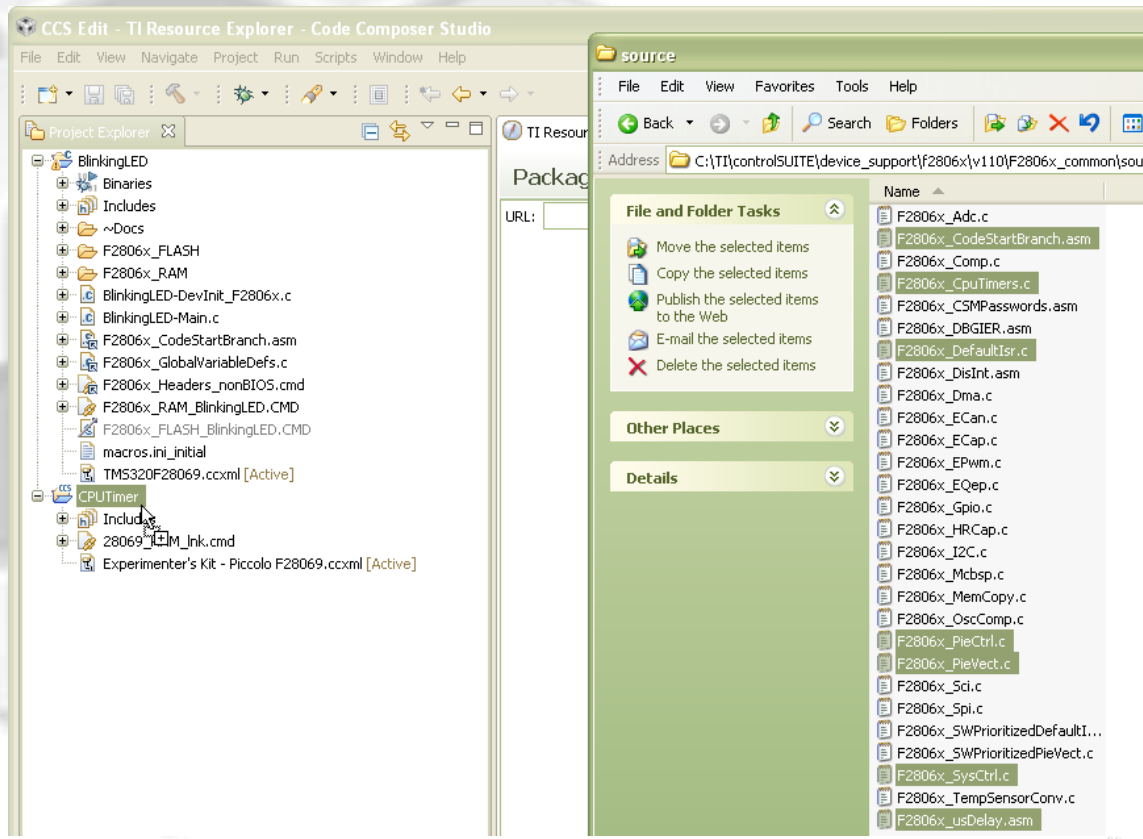
CPU定时器：创建一个构建变量

- 转到“构建变量”(Build Variables)首选项
  - 在过滤领域输入”Variables”，方便查找
- 构建变量允许您在项目属性中使用变量
  - 链接资源的路径变量仅适用于链接文件
- 使用“添加”按钮来创建一个“构建变量”，指向**controlSUITE**根目录的位置
- 完成时，点击“OK”



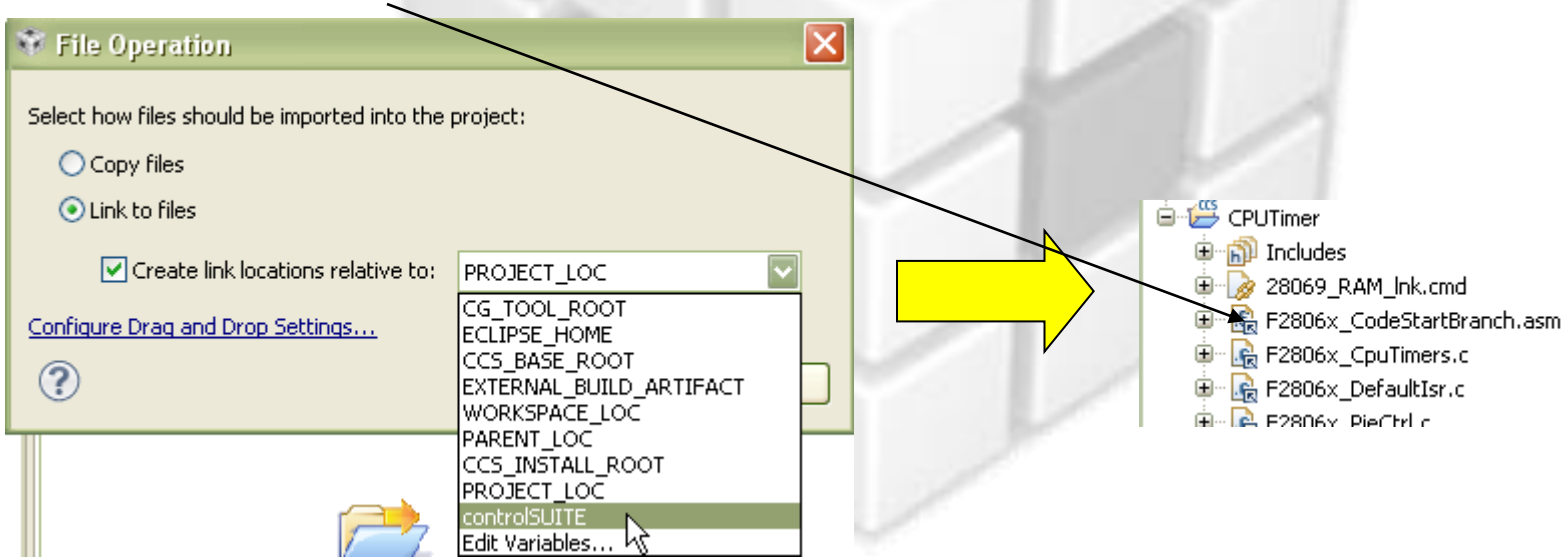
CPU定时器 :把源文件链接到项目

- 打开Windows Explorer并浏览到：
  - C:\TI\controlSUITE\device_support\f2806x\v110\F2806x_common\source
    - 把以下源文件拖放到CCS的Project Explorer视图中的新项目
      - F2806x_CodeStartBranch.asm
      - F2806x_CpuTimers.c
      - F2806x_DefaultIsr.c
      - F2806x_PieCtrl.c
      - F2806x_PieVect.c
      - F2806x_SysCtrl.c
      - F2806x_usDelay.asm



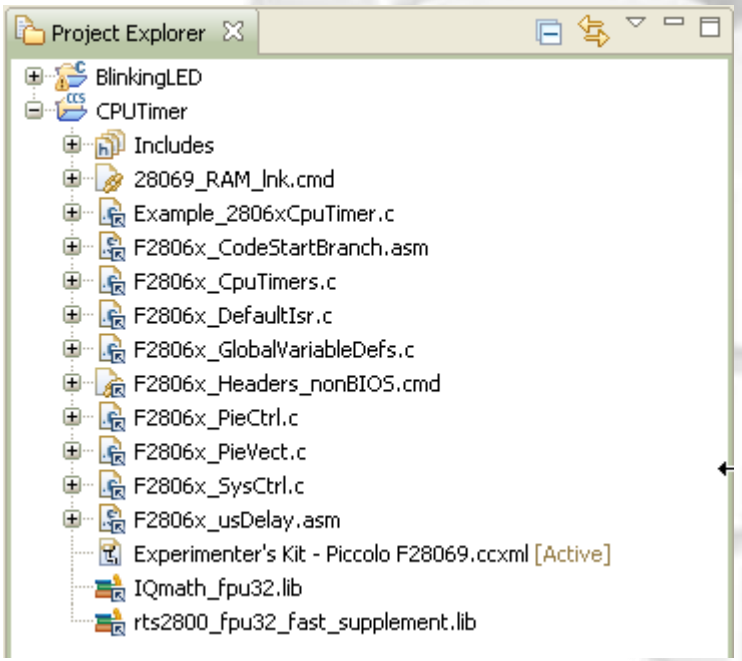
CPU定时器 :把源文件链接到项目

- 一个对话框将出现，确认您是否想复制或链接文件到项目
  - 选择“链接到文件(Link to files)”
  - 选择“创建链接的相对位置(Create link locations relative to)”
    - 使用我们新创建的链接资源的变量（controlSUITE）
  - 点击“确定”
- 有着“链接”(Link)图标的文件将出现在Project Explorer中



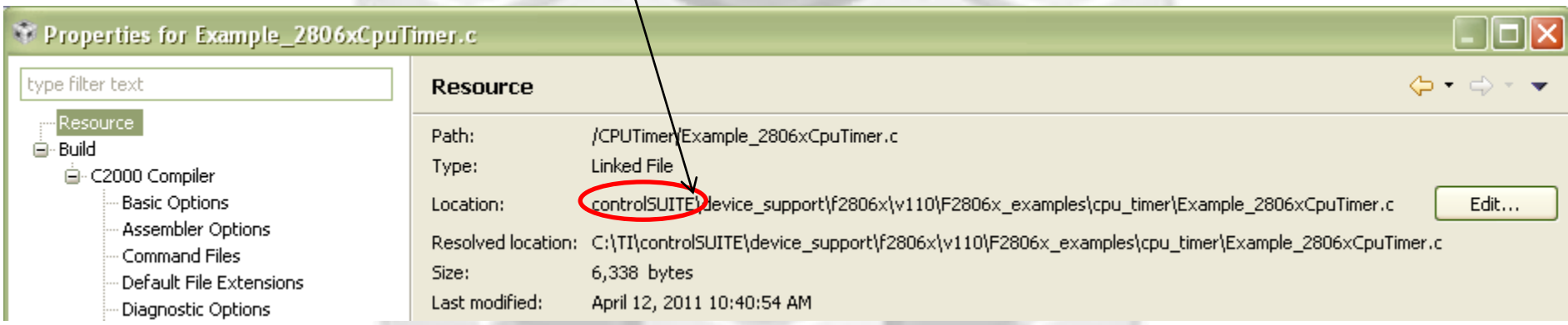
CPU定时器 :把源文件链接到项目

- 用相同的方式链接其余所需的文件：
  - C:\TI\controlSUITE\device_support\f2806x\v110\F2806x_examples\cpu_timer\Example_2806xCpuTimer.c
  - C:\TI\controlSUITE\device_support\f2806x\v110\F2806x_headers\source\F2806x_GlobalVariableDefs.c
  - C:\TI\controlSUITE\device_support\f2806x\v110\F2806x_headers\cmd\F2806x_Headers_nonBIOS.cmd
  - C:\TI\controlSUITE\libs\math\IQmath\v15c\lib\rts2800_fpu32_fast_supplement.lib
  - C:\TI\controlSUITE\libs\math\FPUfastRTS\V100\lib\IQmath_fpu32.lib



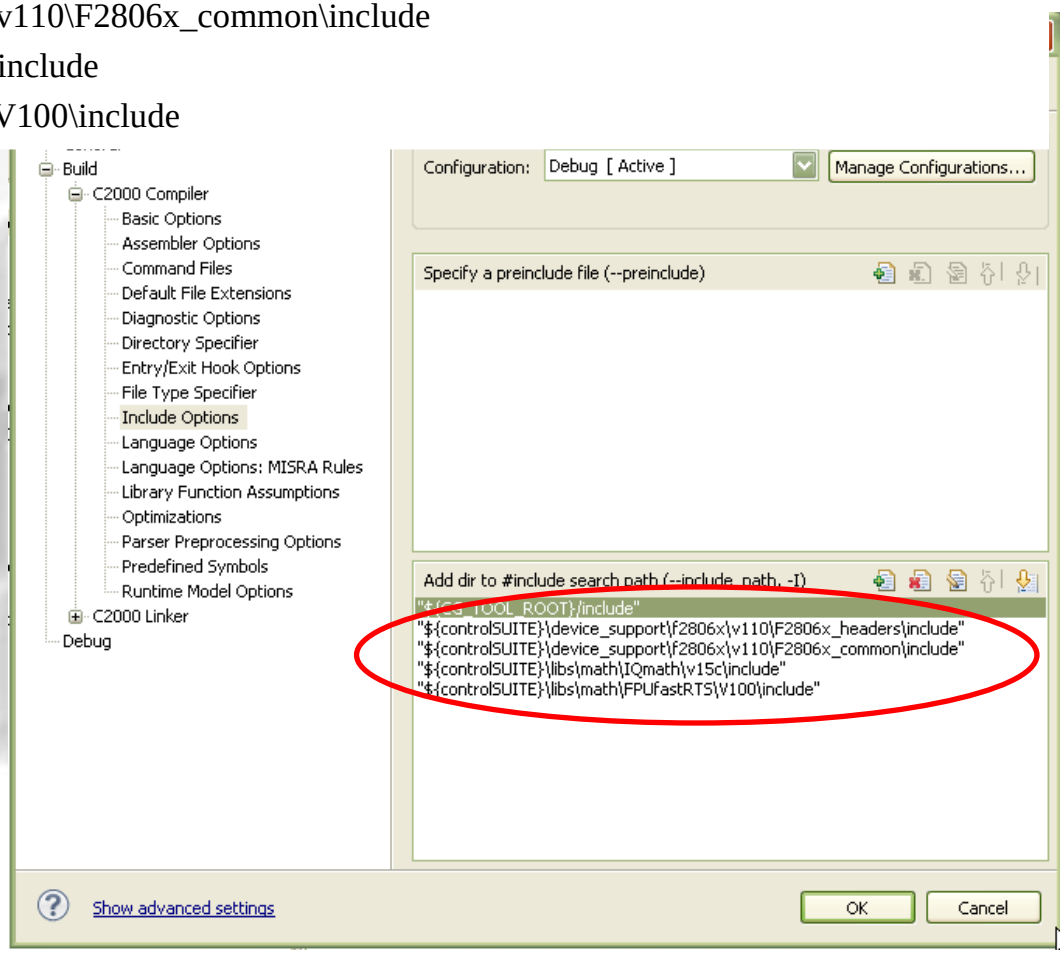
CPU定时器 :把源文件链接到项目

- 右键单击源文件，并检查“属性”(Properties)
  - 注意 “位置”(Location)参数如何引用链接资源的变量



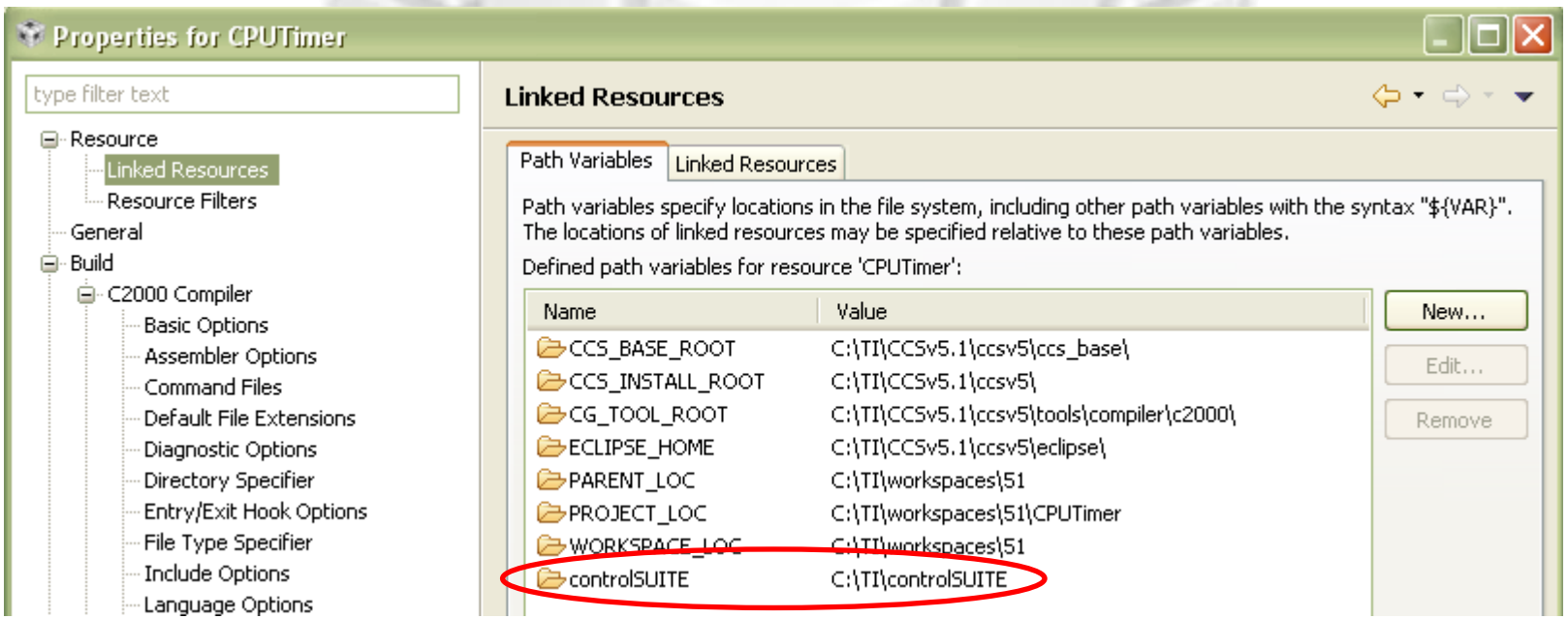
CPU定时器：修改项目属性

- 右键单击该项目并选择“属性”
- 在编译器“包含选项**(Include Options)**”中，把以下路径添加到搜索列表中：：
  - ${controlSUITE}\device_support\f2806x\v110\F2806x_headers\include
  - ${controlSUITE}\device_support\f2806x\v110\F2806x_common\include
  - ${controlSUITE}\libs\math\IQmath\v15c\include
  - ${controlSUITE}\libs\math\FPUfastRTS\V100\include
- **‘${<BUILD VARIABLE>}’**是在项目属性中使用构建变量的语法



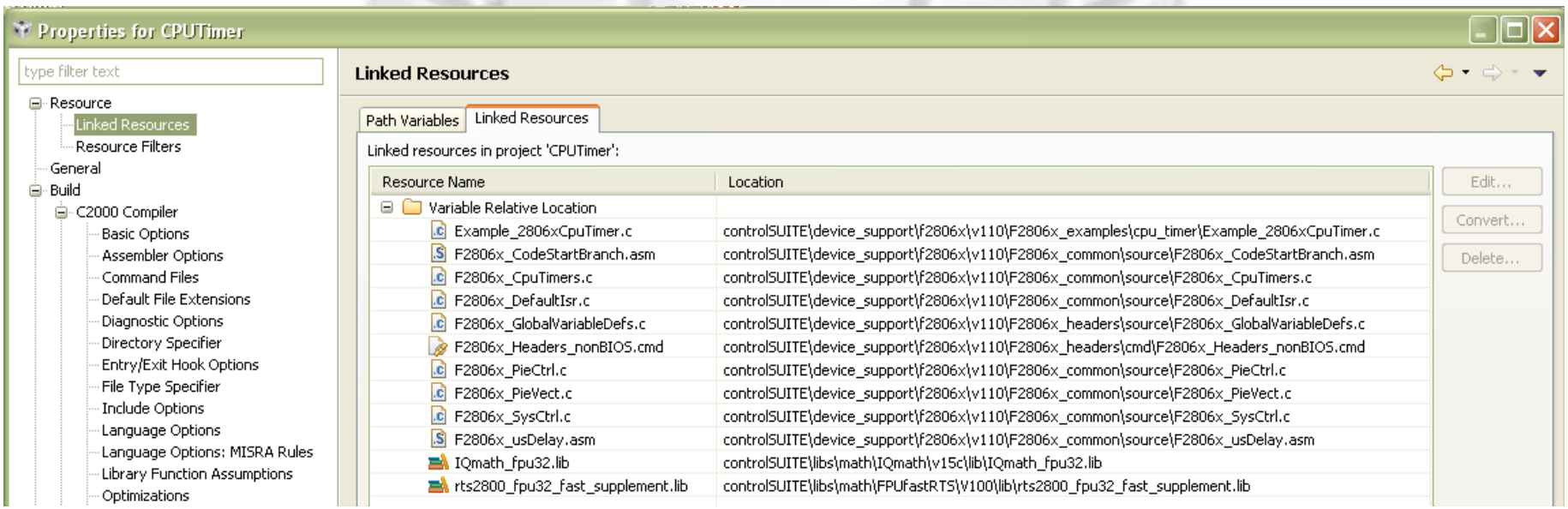
CPU定时器：项目属性

- 打开“资源 - >链接的资源(Resource -> Linked Resource)”能看到项目所有链接的资源路径变量
  - 这将显示所有在项目层面和工作区层面上所创建的变量
- 工作区层面上的链接资源的路径变量会出现在列表中
- 在这里可以编辑变量，但改变只会发生在项目层面（存储在项目文件中）



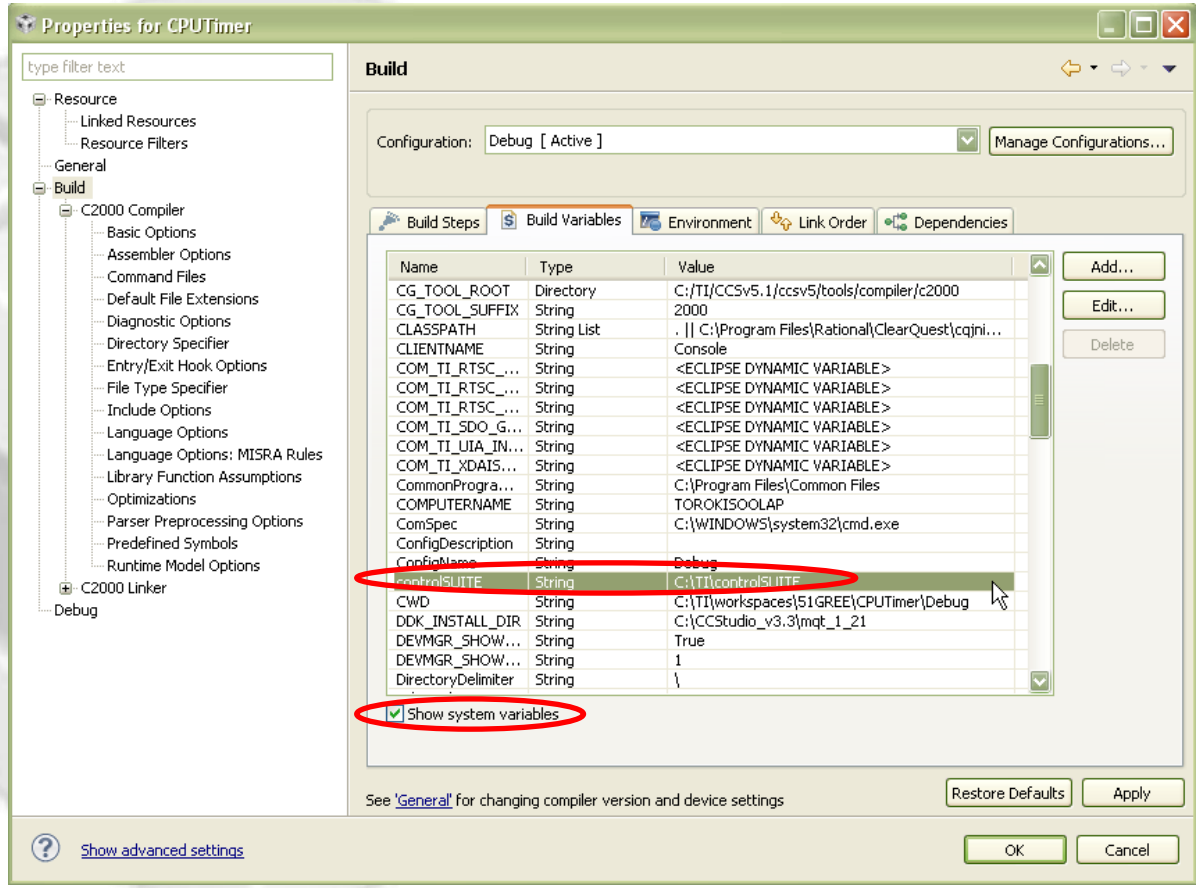
CPU定时器：项目属性

- “链接的资源”(Linked Resources)标签显示所有已链接到项目的文件
  - 链接文件会因使用链接变量和使用绝对路径分类排序
- 可以使用“编辑(Edit)...”按钮修改链接
- 链接可以通过“转换(Convert) ...”按钮，转换为使用绝对路径



CPU定时器：项目属性

- 打开“构建->构建变量”(Build->Build Variable)来查看所有的项目变量
  - 默认情况下，只有项目级别的变量将被列出
  - 启用“显示系统变量(Show system variables)”复选框来查看在工作区和系统级设置的变量
- 注意工作区级创建的构建变量出现在列表

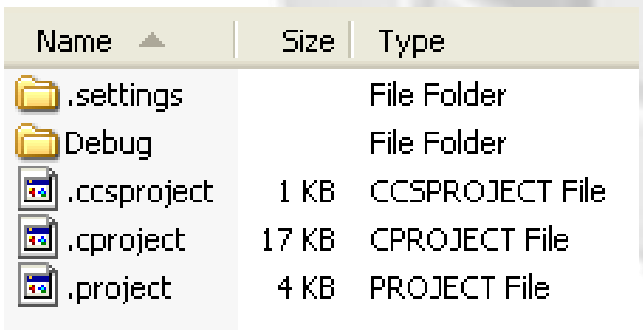


项目与工作区级的变量

- 在刚才的几页里，我们看到“链接资源的路径变量”(Linked Resource Path Variables)和“构建变量”(Build Variables)可以在项目级别设置
- 在工作区级别设置这些变量比较在项目级别设置的好处是什么？
  - 所有项目都可以重复使用相同的变量（设置一次）
  - 不需要修改项目！
    - 这对源控制工具管理的项目非常重要。以避免不断检出(checkouts)，从而该项目可写入！

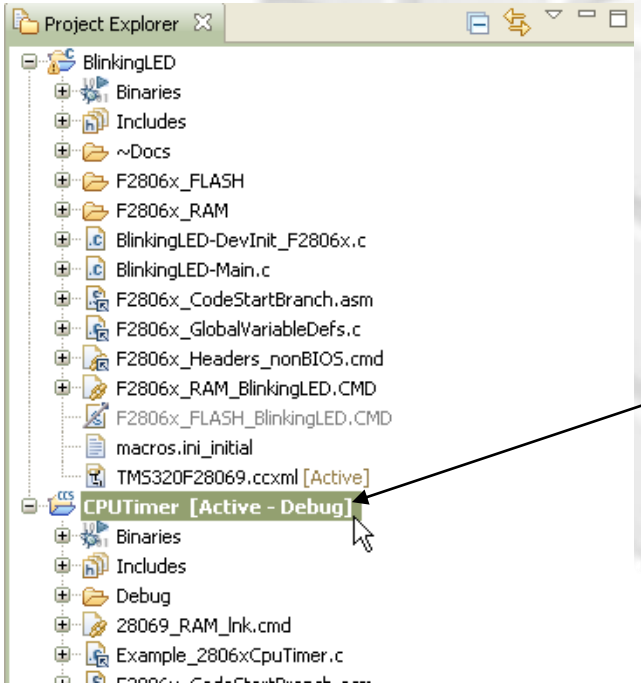
版本控制 - 提交(Check In)哪些文件？

- CCS项目文件夹内生成多个文件/文件夹
  - .CPROJECT 和.PROJECT是**Eclipse**项目文件，并应提交
  - .CCSPROJECT是CCS的具体项目文件，应提交。
  - .settings是一个标准的Eclipse文件夹，其中包含适用于该项目的设置。应提交此文件夹
  - .launches是在启动调试会话时生成的目录。它不涉及您的项目的构建，没有必要提交
  - 项目配置文件夹的内容（调试/发布） (Debug/Release)不需要进行提交



Eclipse概念：焦点(Focus)

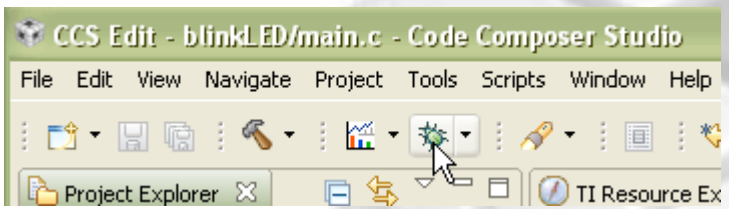
- 焦点是指工作台的选中的部件
  - 可已是编辑器，视图，一个项目，等等。
- 这个概念很重要，因为在Eclipse中的几个操作与中的焦点有关
  - 项目构建的错误，控制台，菜单和工具栏选项，等等。



'CPU Timer'项目，因为它已被选中，所以它是"焦点项目"。因此，按"调试"按钮将构建项目和开始调试'CPU Timer'项目. 请注意焦点项目用**黑体和"Active"**字样表示。

构建和加载程序

- 确认’CPU Timer’ 项目已是焦点项目，然后按下‘Debug’ 按钮



实时模式
REAL-TIME MODE

使用实时模式：概要

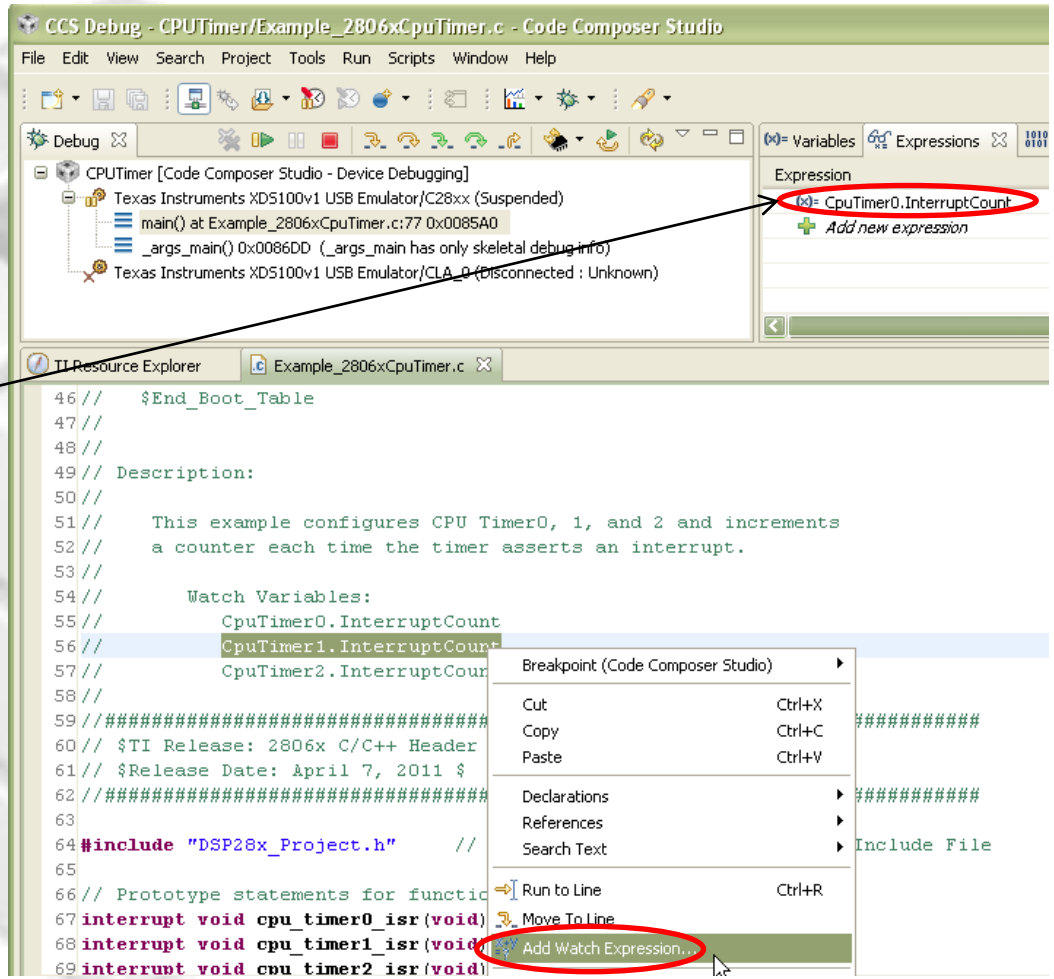
- 主要目标
  - 添加变量到“表达式”(Expressions)视图
  - 启用“表达式”视图中的“连续刷新”(Continuous Refresh)
  - 观察实时模式禁用和启用下的刷新行为，

什么是实时模式?

- 目标通常需要为了调试器访问而暂停（读/写存储器/寄存器等）
  - 许多情况下，调试器将“悄悄地停止”(silently halt)运行中的处理器 - 自动停止CPU，读取/写入内存，然后再次运行
    - 这是在CCS3.3中，当您在运行时单击“刷新”监视(watch)窗口的行为。
    - 如果调试一个实时音频例程，这将感觉象是音频的毛刺(glitch)。然而，如果是电机或调节电源的控制算法，暂时停止处理器是非常不可取的
- 实时模式是一种在CPU运行/执行代码中，可以修改内存/外设/ 寄存器的内容，和服务中断的运作模式，
  - 用户还可以在非时序关键（non-time critical）代码单步，同时无干扰地服务时序关键中断
  - 不需要软件监控（通过硬件支持）

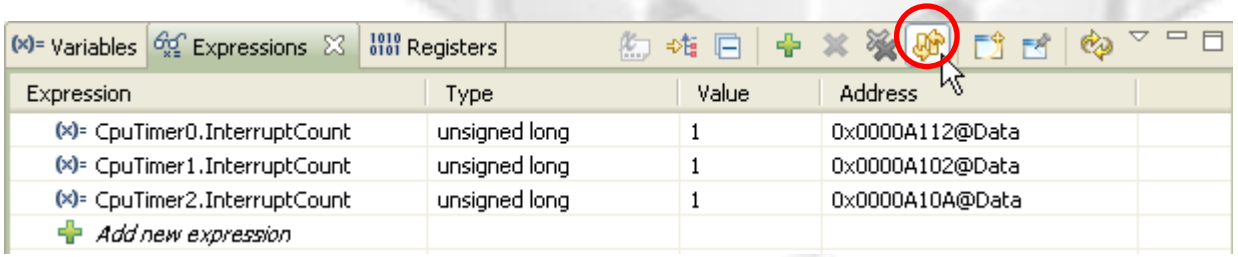
在监视窗口(Watch Window)添加观察变量

- v5 表达式试图(expression view)
- 转到第55行 “Example_2806xCpuTimer.c”
  - Highlight “CpuTimer0.InterruptCount”，右键单击，然后选择“添加观察表达式”(Add Watch Expression)
  - 变量出现在“表达式”视图
- 重复：
  - CpuTimer1.InterruptCount（第56行）
  - CpuTimer2.InterruptCount（57行）



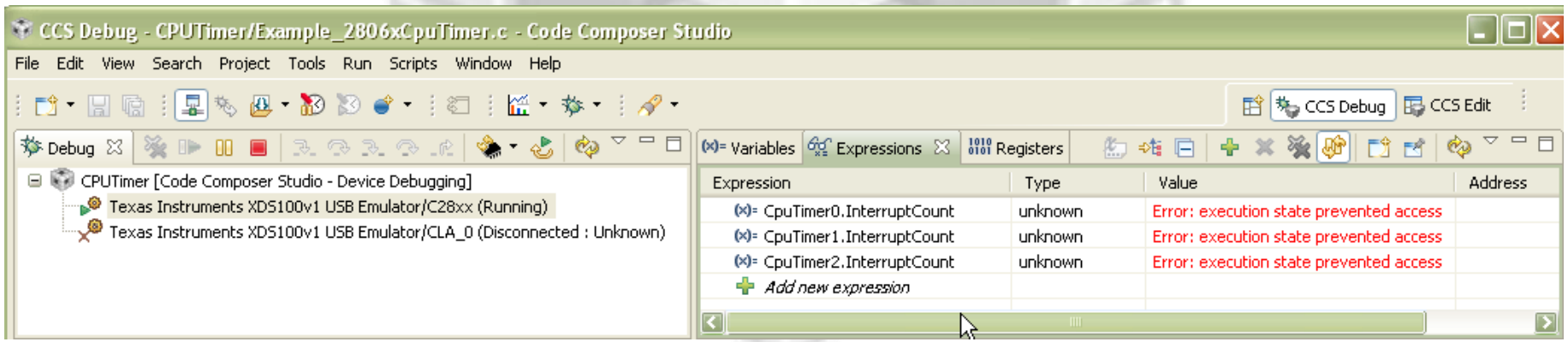
表达式视图：连续刷新 (Continuous Refresh)

- “连续刷新”将定期刷新表达式视图
  - 默认刷新间隔为500毫秒
  - 默认的刷新间隔是可配置的
- 启用连续刷新模式，使观测的变量会不断在程序运行时实时刷新



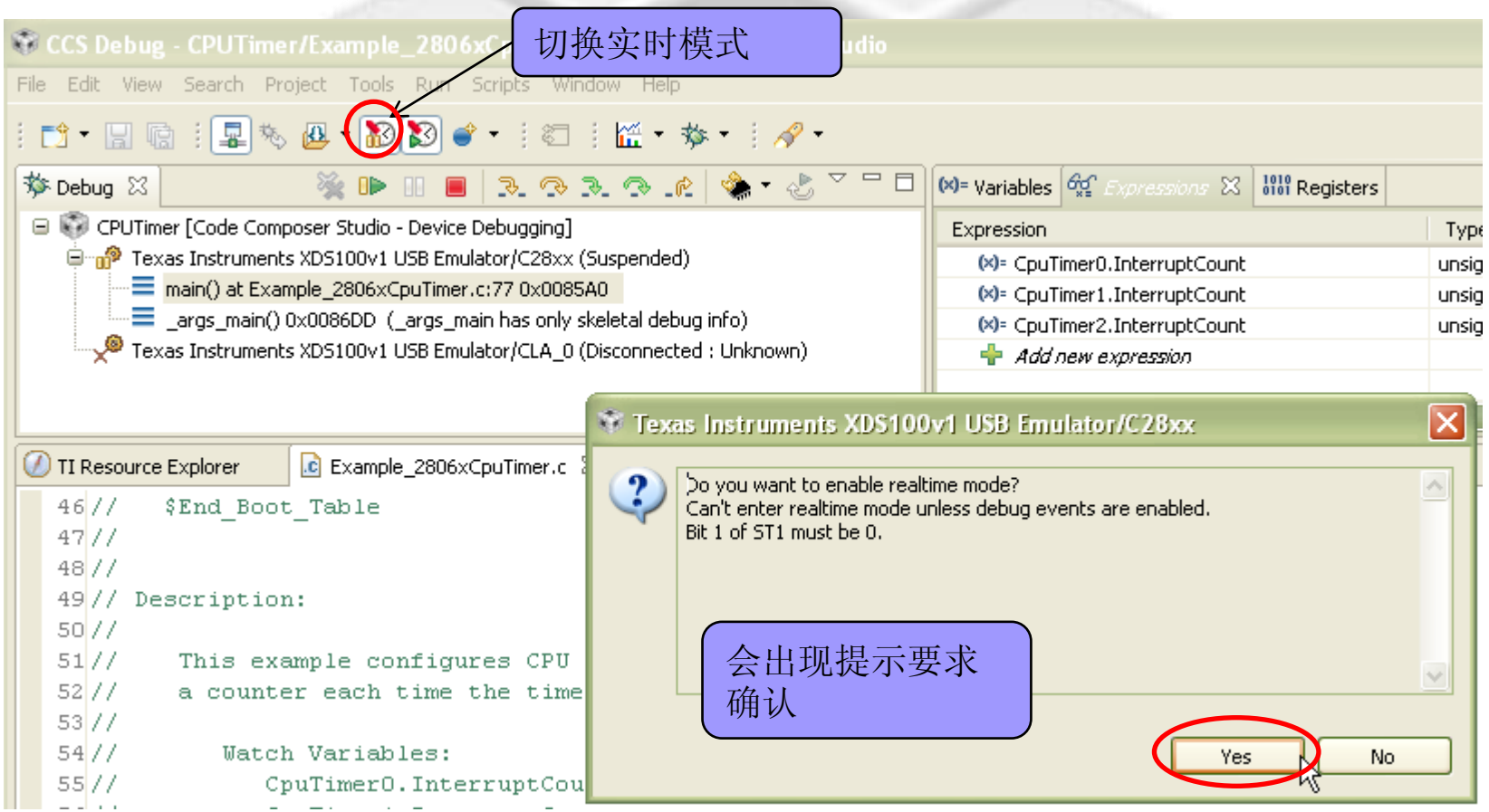
非实时模式运行

- 运行程序
- 请注意，“表达式”视图将无法更新连续刷新的值
- 非实时模式下，目标必须停止来让CCS访问目标存储器



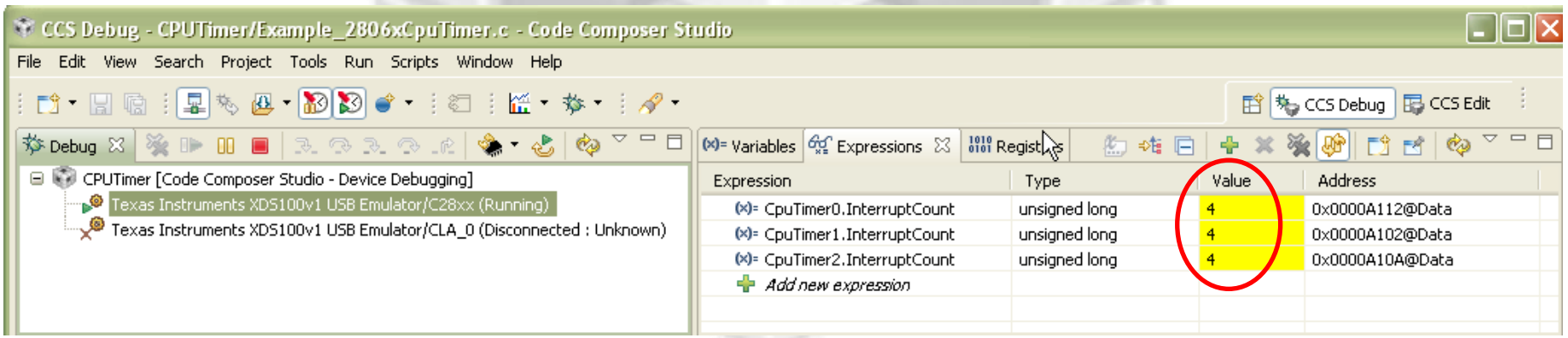
启用实时模式

- 停止目标，然后启用实时模式来监视正在改变的观测变量



实时模式运行

- 运行程序
- 观看表达式视图中，每秒CPU定时器中断而体现的变量的增量
- 实时模式允许CCS在目标运行中访问到目标存储器

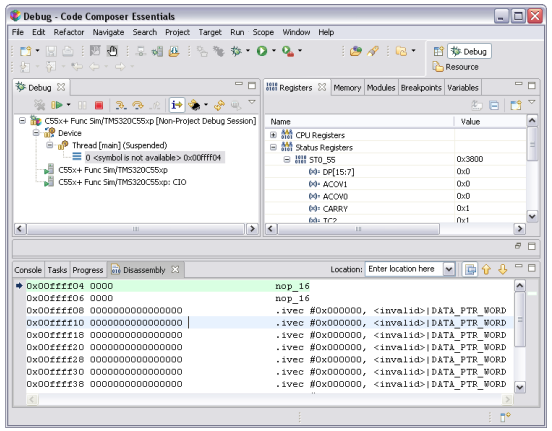


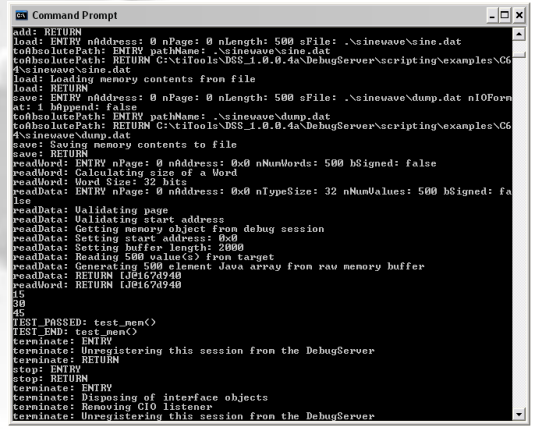
调试服务器脚本
DEBUG SERVER SCRIPTING

调试服务器脚本：简介

- 主要目标
  - 熟悉调试服务器脚本（DSS）
  - 运行DSS例子“loadti”

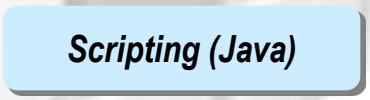
调试服务器脚本：概述





CCS：
基于Eclipse的开发环境

DSS：
批处理脚本环境，
没有GUI依赖



调试服务器脚本：介绍

- 什么是调试服务器脚本？
  - 调试服务器（调试器）的一套Java API
  - 可用于使调试器的任何功能自动化
- 为什么使用它？
  - 自动化测试和性能测试
  - 支持连接到调试器的所有CPU/内核
- 可以通过提供第三方工具脚本化，如：
  - Javascript （通过Rhino ）
  - Perl（通过“inline：Java”的模块）
  - Python（通过Jython）
  - TCL（通过Jacl/ Tclblend）

DSS vs. CCS 脚本

- DSS超越CCScripting的优点：
  - 更快（没有GUI的开销，在进程中运行）
  - 更强大和更详细的日志(logging)
  - 异常(exception)处理
    - 脚本可处理DSS APIs抛出的异常
  - 直接与调试服务器连接（没有调试器GUI）
    - 独立DSS的安装比全整的CCStudio安装所需的空间小很多
      - 轻量安装，适用于在实验室机器作自动化操作
  - 同时运行多个脚本实例
  - 可以从CCS- Eclipse得到互动控制台的支持。
  - 更直接的API支持（减少对GEL的依赖）
    - 引脚/端口连接的API
    - 数据加载/保存的API + 对二进制文件支持的计划
    - CCS 文件I / O API（计划中）
    - ATK的支持（计划中）
  - Linux上的支持
    - CCS的脚本是Win32/COM

DSS:更快

- 没有GUI的自动化（没有GUI的开销）
  - 没有窗口重新绘制/刷新/更新
- 对一些API开销更少
- 在进程内运行
  - CCScripting 在与CCS不同的进程内运行

DSS:基准(Benchmarks)

- 启动（启动调试器）
  - CCScripting：～5秒
  - DSS的：～1秒
- 程序加载API
  - CCScripting比DSS API在程序加载多出～4秒的开销
    - 用DSS加载小型的例子*.out文件需要～65毫秒
    - 同样的情况CCScripting需要～4秒
    - 在程序加载API被反复调用的情况下的叠加效果...
      - DSS：加载200次 =～13秒
      - CCScripting：加载200次 =～800秒
      - DSS节省了～12分钟！
  - 在几个个别的脚本上从CCScripting迁移到DSS后增加了1.5倍到3倍的速度

DSS： 日志(Logging)

- XML格式（易于解析）
  - 可以指定在XML文件中引用的XSTL文件
    - 配置日志在用Web浏览器打开时，信息显示的方式
    - XSTL教程：http://www.w3schools.com/xsl/xsl_intro.asp
- 更多信息
  - 序列号
  - 时序信息（总和，delta 值等）
  - 状态消息
- 日志的CIO输出

DSS: 日志: 生成XML日志

```
<?xml version="1.0" encoding="windows-1252" standalone="no"?>
<?xml-stylesheet type="text/xsl" href="SimpleTransform.xsl"?>
<log>
<record>
  <date>2007-05-02T15:30:15</date>
  <millis>1178134215917</millis>
  <sequence>0</sequence>
  <logger>com.ti</logger>
  <level>FINER</level>
  <class>com.ti.ccstudio.scripting.environment.ScriptingEnvironment</class>
  <method>traceSetConsoleLevel</method>
  <thread>10</thread>
  <message>RETURN</message>
</record>
<record>
  <date>2007-05-02T15:30:15</date>
  <millis>1178134215917</millis>
  <sequence>1</sequence>
  <logger>com.ti</logger>
  <level>FINER</level>
  <class>com.ti.ccstudio.scripting.environment.ScriptingEnvironment</class>
  <method>getServer</method>
  <thread>10</thread>
  <message>ENTRY sServerName: LegacySetupServer.1</message>
</record>

...
```

使用的XSTL文件

DSS日志：使用网页浏览器打开

Debug Server Log

Total Execution Time: 89517 ms

Sequence	Time (ms)	Delta (ms)	Level	Method	Message
0	0		FINER	traceSetConsoleLevel	RETURN
1	0	0	FINER	getServer	ENTRY sServerName: LegacySetupServer.1
2	0	0	FINER	getServer	Getting definition for: LegacySetupServer.1
3	10	10	FINER	getServer	Constructing server
4	10	0	FINER	getServer	Starting server
5	20	10	FINER	start	ENTRY
6	50	30	FINER	start	RETURN
7	50	0	FINER	getServer	RETURN com.ti.debug.engine.scripting.LegacySetupServer@253498
8	50	0	FINER	ccsConfigClear	ENTRY
9	50	0	FINER	ccsConfigClear	Creating system setup object
10	2865	2815	FINER	ccsConfigClear	Clearing configurations
11	4166	1301	FINER	ccsConfigClear	Saving system configuration
12	6260	2094	FINER	ccsConfigClear	RETURN
13	6260	0	FINER	ccsConfigImport	ENTRY sConfig: C:\DSS\DebugServer\drivers\import\DM6446_ltl_endian_sim.ccs

DSS：异常处理

- 无GUI（没有弹出对话框窗口等）
- 所有API在遇到错误时都会抛出一个Java异常
  - 可以在脚本中处理异常（例如Javascript的try - catch）

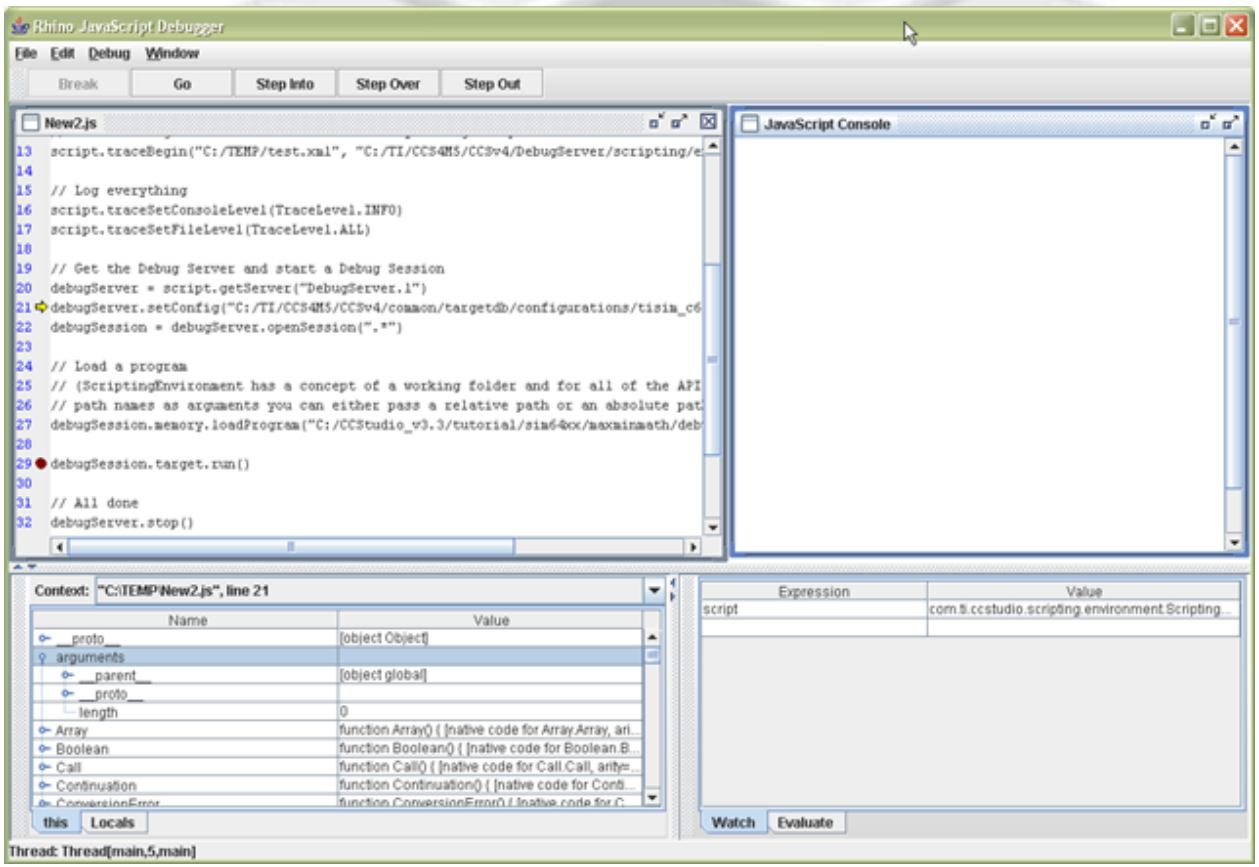
DSS：异常处理

- DSS API抛出Java异常
- 您可以脚本中处理异常，使其优雅地关闭/失败或继续

```
try {
    debugSession.memory.loadProgram(testProgFile);
} catch (ex){
    errCode = dssErrorHdl.getLastErrorID();
    dssScriptEnv.traceWrite("errCode: " + errCode + " - " +
          testProgFile + " does not exist!");
    quit();
}
```

DSS:调试脚本

- DSS附带Rhino调试器，它支持：
  - 单步，断点，查看脚本变量...



DSS：使用Rhino的调试器

- 启用犀牛(Rhino)
  - 用文本编辑器打开“dss.bat”（<安装目录>\ ccsv5\ ccs_base\ scripting \ bin）
  - 犀牛脚本引擎启动调动在65行，把更换！RHINO_SHELL！ 成 ！RHINO_DEBUGGER！

DSS的示例脚本（1）

```
// Import the DSS packages into our namespace to save on typing
importPackage(Packages.com.ti.debug.engine.scripting);
importPackage(Packages.com.ti.ccstudio.scripting.environment);
importPackage(Packages.java.lang);

// Modify these variables to match your environment. Use forward slashes
var ccs5InstallDir = "C:/Program Files/Texas Instruments";
var ccs5ScriptingLabDir = "C:/CCSWorkshop/labs/scripting";

// Create our scripting environment object - which is the main entry point
// into any script and the factory for creating other Scriptable Servers and Sessions
var script = ScriptingEnvironment.instance();

// Create a log file in the current directory to log script execution
script.traceBegin(ccs5ScriptingLabDir + "/test.xml", ccs5InstallDir +
    "/ccsv5/ccs_base/scripting/examples/DefaultStylesheet.xsl");

// Set trace levels for console and logs
script.traceSetConsoleLevel(TraceLevel.INFO);
script.traceSetFileLevel(TraceLevel.ALL);
```

DSS的示例脚本（2）

```
// Get the Debug Server and start a Debug Session
debugServer = script.getServer("DebugServer.1");
debugServer.setConfig(ccs5ScriptingLabDir + "/c6416_le_sim.ccxml");
debugSession = debugServer.openSession(".*");

// Load a program
debugSession.memory.loadProgram(ccs5ScriptingLabDir + "/mainapplication.out");

// Run the target
debugSession.target.run();

// All done
debugServer.stop();

script.traceWrite("TEST SUCCEEDED!");

// Stop logging and exit.
script.traceEnd();
java.lang.System.exit(0);
```

DSS的示例：loadti

- DSS的JavaScript示例，作为一个命令行程序，它可以在TI的目标板上加载/运行可执行文件 *. Out
- DSS的例子，但是能适用于任何程序和目标
- 高易用性和缺乏对GUI依赖，使其成为一个有吸引力的自动化工具，用于快速完整性测试(Sanity Tests)和批处理回归(Batch Regressions)测试
- 一些支持的基本功能包括：
  - C I/O标准输出可以被发送到控制台和脚本日志
  - 生成详细的运行日志
  - 执行基本的应用程序基准测试（通过使用性能分析的时钟）
  - 把参数传到主函数
  - 从主机到目标内存，载入/保存数据（反之亦然）
  - 复位目标
- 关于loadti上的更多细节，请参阅“README.TXT”
  - <安装目录>\ccsv5\ccs_base\scripting\examples\loadti

使用loadti: 运行‘BlinkingLED.out’

- 关闭所有正在运行的CCS
- 打开一个DOS命令窗口
- 浏览到“BlinkingLED.out”程序的位置
  - Location: <WORKSPACE>\BlinkingLED\F2806x_FLASH\BlinkingLED.out
  - > set PATH=%PATH%;<CCS INSTALL DIR>\ccsv5\ccs_base\scripting\examples\loadti
  - 您需要把<CCS_INSTALL_DIR>换成您的安装位置
  - > loadti –h (for console help)
  - > loadti –c ..\TMS320F28069.ccxml BlinkingLED.out
- 加载将需要一些时间，因为需要擦除/编程闪存
  当你可以看到LD2闪烁时，程序已在运行
- 按CTRL- C中止

DSS: 资源

- DSS MediaWiki的文档（推荐阅读）：
  - http://processors.wiki.ti.com/index.php/Debug_Server_Scripting
- DSS API的文档：
  - <INSTALL DIR>\ccsv5\ccs_base\scripting\docs\GettingStarted.htm
- 命令行项目管理工具的MediaWiki文档：
  - http://processors.wiki.ti.com/index.php/Projects_-_Command_Line_Build/Create

脚本控制台 (Scripting Console)

- CCS的命令行操作
- 视图 - >脚本控制台 View->Scripting Console
- 按TAB查看命令列表
  - 按TAB使用部分输入的命令自动完成功能
- 命令相关的文档
  - JS：> help <COMMAND>
- JavaScript和shell可以访问到所有的DSS API
- 从控制台运行DSS的脚本
- 创建自己的自定义命令
  - 在*. js文件中创建一个JavaScript函数
  - 加载自定义的JavaScript文件
    - loadJSFile <full路径>/ myCustomConsoleCmd.js
    - 可选的第二个参数(布尔)，将决定是否自动加载脚本
  - 该函数现在可以从脚本控制台内用其名称调用

脚本控制台

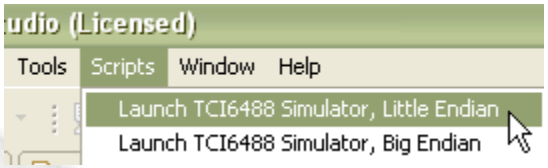
- 视图 - >脚本控制台 View->Scripting Console
- 按TAB查看命令列表
  - 按TAB使用部分输入的命令自动完成功能
- 命令相关的文档
  - JS：> help <COMMAND>

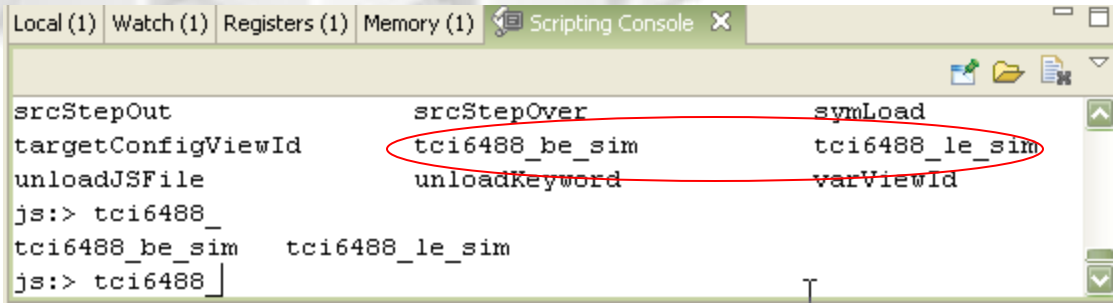
脚本控制台

- 脚本控制台和GEL都可用于自动化
- GEL仅限于一个激活的调试会话并且（多数情况）只适用于一个调试上下文
- 脚本控制台随时都可以使用（虽然某些命令在没有激活的调试会话的情况下将无法正常工作）
- 脚本控制台和GEL都可以为自定义“脚本”添加菜单
  - GEL：hotmenu<function>
  - 脚本控制台：hotmenu.addJSFunction

脚本控制台

```
// Add entries to the 'Scripts' menu
hotmenu.addJSFunction("Launch TCI6488 Simulator, Little Endian", "tci6488_le_sim()");
hotmenu.addJSFunction("Launch TCI6488 Simulator, Big Endian", "tci6488_be_sim()");

// Path to folder with target setup ccxml files
var setupConfigFileFolder = "C:/Documents and Settings/login/user/CCSTargetConfigurations";

// configure for a TCI6488 Symmetric Simulator, Little Endian
function tci6488_le_sim()
{
   ds.setConfig(setupConfigFileFolder + "/tci6488_le_sim.ccxml");

   debugSessionCPU1 = ds.openSession("*", "C64+_0");
   debugSessionCPU2 = ds.openSession("*", "C64+_1");
   debugSessionCPU3 = ds.openSession("*", "C64+_2");
}

...
```





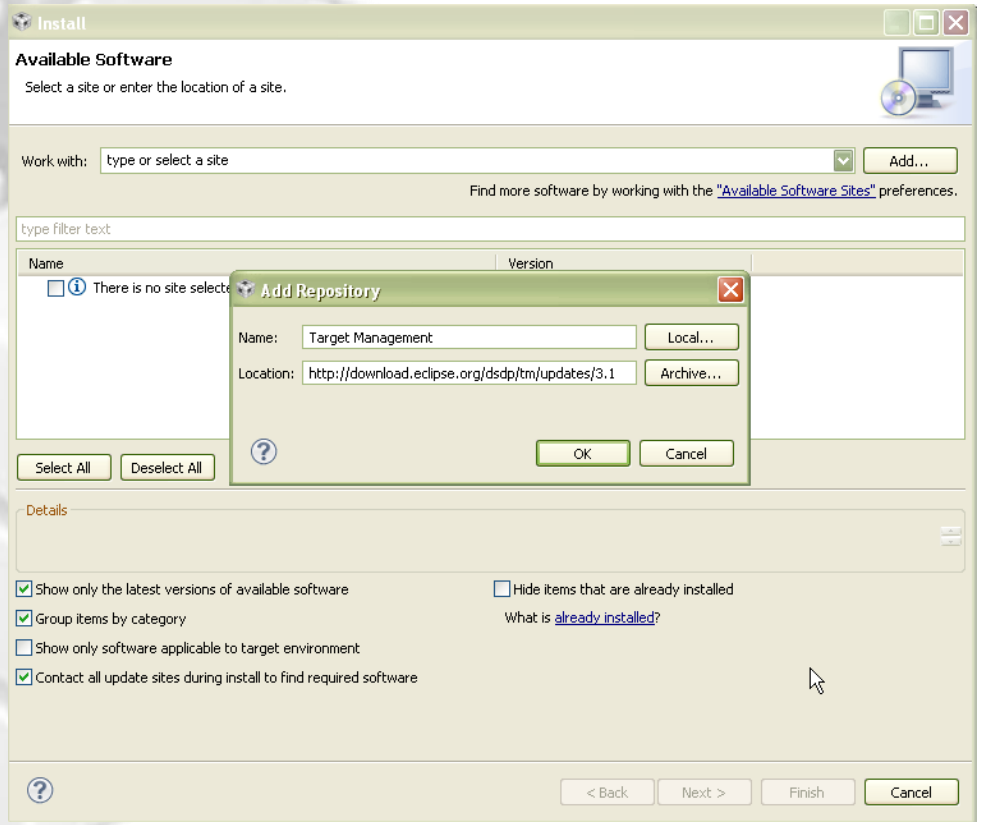
ECLIPSE插件

Eclipse插件 - 基本

- CCSv5基于Eclipse，所以用户拥有巨大的第三方插件的选择
  - http://marketplace.eclipse.org/
- CCSv5基于Eclipse3.7
  - 为了最佳的兼容性，请使用这个版本对应的插件

Eclipse插件 - 安装

- 使用Eclipse更新管理器(Update Manager)
  - “帮助 - >安装新软件”(Help -> Install New Software)来进行更新安装（指定的远程站点（URL）或本地站点（目录）



- 拖放
  - 许多插件是压缩归档，可以直接下载并复制到: \ ccsv5\eclipse \dropins文件夹

controlSTICK

- 想了解更多28069 controlSTICK吗？
  - http://processors.wiki.ti.com/index.php/C2000_32-bit_Real-Time_MCU_Training

Questions?