xTIMEcomposer用户指南

版本B

出版日期：2013年3月1日

目录

第 A 部分 安装

内容

xTIMEcomposer运行系统要求

支持xTIMEcomposer工具的官方平台：

Windows XP SP3

- 32位操作系统安装带32位JAVA虚拟机

Windows 7 SP 1

- 32位操作系统安装带32位JAVA虚拟机
- 64位操作系统安装带32位JAVA虚拟机

Mac OS X 10.5 +

- 英特尔处理器

Linux CentOS 5.8

- 32位操作系统安装带32位JAVA虚拟机
- 64位操作系统安装带32位JAVA虚拟机

xTIMEcomposer工具还可用于其他多种版本的Linux系统中，如RedHat和Ubuntu。关于兼容版本的最新信息，请看：

- http://www.xmos.com/tools

你必须安装1.5或以上的JAVA虚拟机，可以从下面网址下载：

- http://java.sun.com/javase/downloads

Shiqiang Xiao 13-11-7 11:59 AM
已删除: 运行环境

Shiqiang Xiao 13-11-7 11:58 AM
已删除: 的32位系统

Shiqiang Xiao 13-11-7 11:59 AM
已删除: 带32位JAVA运行环境的32位系统

Shiqiang Xiao 13-11-7 11:59 AM
已删除: 带32位JAVA运行环境的64位系统

Shiqiang Xiao 13-11-7 11:59 AM
已删除: 带32位JAVA运行环境的32位系统
带64位JAVA运行环境的64位系统

Shiqiang Xiao 13-11-7 12:00 PM
已删除: 运行环境版本

2 安装说明

本章内容：

·工具安装

· USB驱动安装

xTIMEcomposer及相关驱动存放在独立平台上——可下载文件中。

2.1 工具安装说明

按下列步骤在个人电脑上安装工具

Windows系统：

1. 从下列地址下载Windows Installer

· http://www.xmos.com/tools

2. 双击运行Installer，按照屏幕提示在个人电脑上安装工具。

Mac系统：

1. 从下列地址下载Macintosh installer

· http://www.xmos.com/tools

2. 双击打开下载的Installer，将xTIMEcomposer图标拖到系统应用文件夹下。

Installer将文件拷贝至你硬盘中。

3. 卸载开发工具

Linux系统：

1. 从下列的址下载Linux archive

· http://www.xmos.com/tools

2. 将Archive解压到安装目录下，例如输入下列命令：

· tar -xzf archive.tgz -C /home/user

Shiqiang Xiao 13-11-7 11:58 AM
已删除: 工具安装

Shiqiang Xiao 13-11-7 11:57 AM
已删除: 卸载Installer

Shiqiang Xiao 13-11-7 12:01 PM
已删除: 从下列地址下载

2.2 Xtag2驱动安装

xTIMEcomposer通过Xtag2与开发板连接。一些开发板带有集成调试适配器，而另一些则需要外接适配器通过XSYS链接器与开发板链接，如图1所示。

Shiqiang Xiao 13-11-7 12:02 PM
已删除: USB

Shiqiang Xiao 13-11-7 12:03 PM
已删除: USB上方的

Shiqiang Xiao 13-11-7 12:03 PM
已删除: 相接

图1.使用xCORE开发板的适配

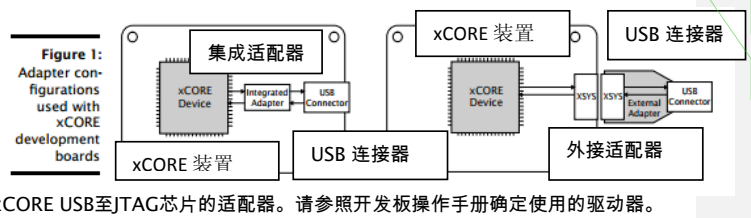


Figure 1: Adapter configurations used with xCORE development boards

xTIMEcomposer支持基于FTDI 或xCORE USB至JTAG芯片的适配器。请参照开发板操作手册确定使用的驱动器。

Windows系统：
使用工具安装程序安装JTAG驱动。安装后插入xCORE开发板以加载驱动。

Mac系统：

OS X本身就提供USB驱动支持。

Linux系统：

一些版本的Linux本身就提供USB驱动支持，而一些情况下则必须启用驱动（见XM-001514-PC）。

第B部分快速入门

Shiqiang Xiao 13-11-7 12:05 PM
已删除: 快速启动

内容

3 开启xTIMEcomposer之旅

本章内容

- 启动xTIMEcomposer Studio
- 启动命令行工具
- Quick Start Menu 快速启动菜单

3.1 启动xTIMEcomposer Studio

启动xTIMEcomposer Studio：

Windows系统：

选择开始——程序——XMOS——xTIMEcomposer_12——xtimecomposer

OS X系统：

打开一个新的Finder窗口，导航到应用文件夹，打开文件夹XMOS_xTIMEcomposer_12，双击xtimecomposer.app图标。

Linux系统：
打开终端窗口，变更至安装目录并输入下列命令：

- source SetEnv
- xtimecomposer

3.1.1 工具注册

首次使用xTIMEcomposer Studio时，系统将询问是否注册XMOS账户工具。注册将享受自动通知文档的更新和软件将直接在Studio里更新的权限，此外还提供了从工具中直接管理账户设定的功能选项。

稍后注册，具体请参考“帮助”——“注册信息”。

Shiqiang Xiao 13-11-7 12:08 PM
已删除: 提供的

Shiqiang Xiao 13-11-7 12:07 PM
已删除: 好处

Shiqiang Xiao 13-11-7 12:07 PM
已删除: 包括

Shiqiang Xiao 13-11-7 12:08 PM
已删除: 自动通知

Shiqiang Xiao 13-11-7 12:09 PM
已删除: 选择

3.2 启动命令行工具

xTIMEcomposer命令行工具使用一整套环境变量（见§ 9.8）来搜索标头文件、库及目标设备。在路径中添加xTIMEcomposer工具，并配置默认环境变量。

Windows系统：

选择“开始”——“程序”——XMOS——xTIMEcomposer_12——“命令提示符”

OS X系统：

打开终端窗口，变更至安装目录，或从Finder窗口打开并输入下列命令：

· SetEnv.command

Linux系统：

打开终端窗口，变更至安装目录并输入下列命令：source SetEnv

可以输入名称和命令行选项来运行任意一种工具。下文中概述了一些常用命令。

3.3 快速启动菜单

xTIMEcomposer Studio Developer栏中的快速启动菜单为所有用户包括XMOS的新开发人员以及经验丰富的用户，提供了便捷的启动点。

拥有xCORE开发板的开发人员可使用页面来生成板项目或找到套件规定的文档和教程。没有开发板的开发人员可以使用模拟器并根据工具教程来操作。

新开发人员可使用快速启动菜单来链接至关键文档和信息，以帮助其使用xTIMEcomposer Studio。而现有用户，可链接至说明工具链最新版与原版之间变更的文档。

为充分利用快速启动菜单，请选择你的专业水平或兴趣，然后根据屏幕提示操作。

在xTIMEcomposer Studio中选择“帮助”——“快速启动”，查看开发者专栏中的快速启动菜单。

4 常用命令

本章内容：

- XCC
- XRUN
- XGDB
- XSIM
- Quick Start Menu 快速启动菜单

本文概述了一些常用命令。

4.1 XCC

为开发板编译程序时，输入下列命令：

1．xcc -print-targets

XCC显示了一系列支持的开发板。

2．xcc <file> -target=<board> -o <binary>

XCC编译文件时为目标板生成二进制可执行文件。

4.2 XRUN

输入下列命令，在开发板上加载编译程序。

1．xrun -l

XRUN打印出一份与个人电脑链接的所有JTAG适配器以及JTAG链上所有设备的枚举列表，表格如下：

标识符	名称	适配器标识符	设备

2．xrun -id <n> -io <binary>

XRUN将二进制文件加载于与带规定标识符的适配器链接的硬件上。

假定从硬件到终端的为标准输出流，则-io选项使XRUN可以与适配器保持链接。

Shiqiang Xiao 13-11-7 12:58 PM
已删除：--

Shiqiang Xiao 13-11-7 12:58 PM
已删除：--

Shiqiang Xiao 13-11-7 12:58 PM
已删除：--

Shiqiang Xiao 13-11-7 12:58 PM
已删除：--

Shiqiang Xiao 13-11-7 12:58 PM
已删除：--

Shiqiang Xiao 13-11-7 12:58 PM
已删除：--

4.3 XGDB

输入下列命令，编译并调试程序：

1. xcc <file> -target=<board> -o <binary> -g
启用调试信息后，XCC会对文件进行编译。

2. xgdb bin.xe
GDB根据提示加载。

3. list-devices
GDB打印出一份与个人电脑链接的所有JTAG适配器以及每根JTAG链上所有设备的枚举列表，表格如下：

标识符	名称	适配器标识符	设备

4. connect -id <id>
GDB与目标硬件链接。

5. 加载
GDB加载二进制文件。

6. break main
GDB在主要功能中添加一个转效点。

7. continue
GDB运行程序直到达到主要部分。

4.4 XSIM

输入下列命令在模拟器上运行程序：

· xsim <binary>

根据GDB提示从调试器启动模拟器：

· connect -s
你可以按照开发板相同的方式在模拟器上加载程序。

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

第C部分

在XDE中开发

内容

5 在xTIMEcomposer中导入或导出项目

本章内容

- 导入项目
- 导出项目

xTIMEcomposer使与其他开发人员共享项目变得简单。

5.1 导入项目

按照下列步骤导入项目：

1. 选择“files”——“ Import”
2. 双击“General”选项，选择“Existing Projects into Workspace”，点击“Next”。
3. 在“Import”对话框中，点击“Browse”（位于Select archive file文本框旁边）
4. 选择归档导入文件，点击“Open”
5. 点击“Finish”

5.2 导出项目

按照下列步骤导出项目：

1. 选择“files”——“Export”
2. 双击“General”选项，选择“归档文件”，而后点击“Archive File”。
3. 在左上方面板中选择想要导出的项目，你可以在右上方面板中以反选的形式排除 文件。
4. 在“To archive file”文本框中输入存档文件名。
5. 点击“Finish”。

Shiqiang Xiao 13-11-7 12:51 PM
已删除: “文件”

Shiqiang Xiao 13-11-7 12:52 PM
已删除: 导入

Shiqiang Xiao 13-11-7 12:52 PM
已删除: 常规

Shiqiang Xiao 13-11-7 12:52 PM
已删除: 现有项目到工作区

Shiqiang Xiao 13-11-7 12:53 PM
已删除: 下一步

Shiqiang Xiao 13-11-7 12:53 PM
已删除: 输入

Shiqiang Xiao 13-11-7 12:53 PM
已删除: 浏览

Shiqiang Xiao 13-11-7 12:53 PM
已删除: 选择归档文件

Shiqiang Xiao 13-11-7 12:53 PM
已删除: 打开

Shiqiang Xiao 13-11-7 12:54 PM
已删除: 完成

Shiqiang Xiao 13-11-7 12:54 PM
已删除: 文件

Shiqiang Xiao 13-11-7 12:54 PM
已删除: 导出

Shiqiang Xiao 13-11-7 12:54 PM
已删除: 常规

Shiqiang Xiao 13-11-7 12:54 PM
已删除: 下一步

Shiqiang Xiao 13-11-7 12:57 PM
已删除: 归档文件

Shiqiang Xiao 13-11-7 12:57 PM
已删除: “完成”

6 使用xSOFTip开发应用程序

- xSOFTip浏览器视图
- 系统信息窗口
- 配置xSOFTip组件
- 从应用程序中生成项目

为便于建设系统，XMOS提供了含各种接口如USB、Ethernet和串行端口和DSP、协议功能的xSOFTip模块选择。xSOFTip模块使用xCORE资源执行规定功能。为了尽可能简单的选择和配置xSOFTip，可以使用xSOFTip浏览器从xSOFTip库中浏览可用模块，了解资源用途并根据自身规格配置模块。工具还可以评估是否选择了适合自身设计的设备。

Shiqiang Xiao 13-11-7 12:56 PM
已删除: 界面

6.1 xSOFTip浏览器视图

xTIMEcomposer Studio独立视图中含xSOFTip浏览器。

1．选择“Window”——“Open Perspective”——“XMOS xSOFTip Explorer”，从而打开xSOFTip浏览器视图。

Shiqiang Xiao 13-11-7 12:58 PM
已删除: 窗口

Shiqiang Xiao 13-11-7 12:58 PM
已删除: 打开视图

Shiqiang Xiao 13-11-7 12:58 PM
已删除: XMOS xSOFTip浏览器

Shiqiang Xiao 13-11-7 12:59 PM
已删除: xSOFTip浏览器

Shiqiang Xiao 13-11-7 12:59 PM
已删除: 系统配置

Shiqiang Xiao 13-11-7 12:59 PM
已删除: 系统信息

Shiqiang Xiao 13-11-7 12:59 PM
已删除: 开发者专栏

Shiqiang Xiao 13-11-7 1:00 PM
已删除: 一般用途

Shiqiang Xiao 13-11-7 1:01 PM
已删除: 但由

Shiqiang Xiao 13-11-7 1:01 PM
已删除: 负责

Shiqiang Xiao 13-11-7 1:00 PM
已删除: 早期开发

Shiqiang Xiao 13-11-7 1:00 PM
已删除:

xSOFTip浏览器视图有4个窗口：

- xSOFTip Browser——列出xSOFTip库中所有可用组件。向“系统配置”窗口添加组件时，“系统信息”窗口将随着应用资源信息不断更新。
- “System Configuration”：显示应用程序中的xSOFTip组件
- “System Information”：你所选xSOFTip以及最适合你应用程序xCORE设备所用的资源。
- “Developer Column”：关于xSOFTip组件的在线信息

每个组件都有一个范围，显示xSOFTip组件的状态：

- “General Use”：xSOFTip包含一整套自XMOS解压的产品。

  可使用整套资源信息。我公司尝试各种方法来确保该模块功能正确，来保证用户使用本模块产品的最终质量。
- Early Development：xSOFTip适用于产品开发并能充分发挥功能。然而，必须特别注意验证使用本软件模块的产品。资源信息可用。

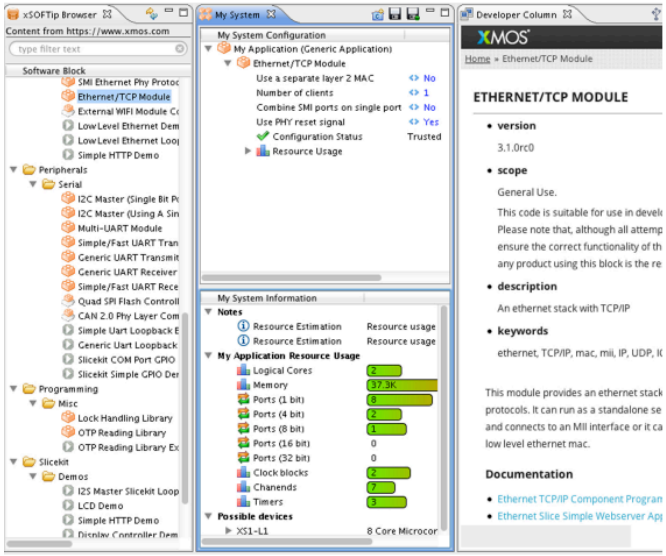


图2. xSOFTip浏览器视图

- 实验阶段：xSOFTip现处于实验/原型阶段。代码存在但不完整，资源信息可用。
- 路线图：xSOFTip位于XMOS开发路线图上。本xSOFTip存在估算资源信息，但代码不可用。
- 开源社区：由开源社区研发xSOFTip，资源信息不可用。

在“浏览器”窗口中选择组件时，“开发者专栏”中就显示了关于组件的信息，包括组件做什么、组件特征及哪种xKIT开发套装适用于本xSOFTip的描述。

将组件添加至“系统配置”窗口后，点击组件向左箭头，从而在“开发者专栏”显示个性化配置选项的附加信息。

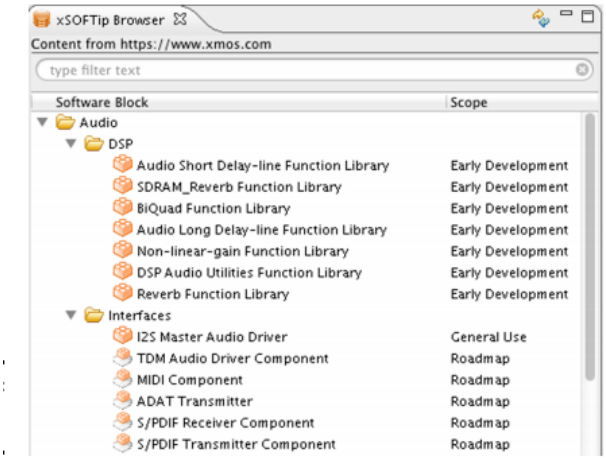


图3：xSOFTip范围

6.2 系统信息窗口

“系统信息”窗口显示在“系统配置”窗口选择所需的所有资源。

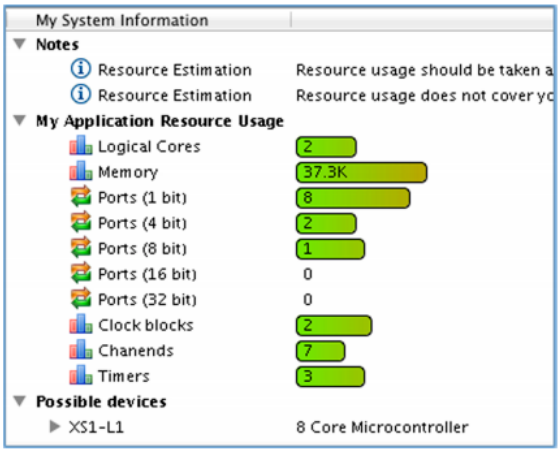


图4 xSOFTip浏览器系统信息窗口

逻辑核：32位的微控制器核。XMOS多核微控制器包括8、16和32个逻辑核的设备。

- 端口：XMOS多核心微控制器输入/输出引脚与端口链接，使软件能够以极低的延迟从引脚处发送、接收信息。端口具有不同的宽度：1bit端口链接至1个输入/输出引脚，4bit端口链接至4个输入/输出引脚。
- 时钟模块：时钟模块用于高精度要求的IO时序控制。
- 通道端点：通道端点为xConnect系统的一部分，允许核心通过低延迟xConnect通道向其他通道发送信息。
- 定时器：定时器用于控制时间的发生时间，定时器以100MHz频率运行，精确到10ns。

6.2.1 识别合适的xCORE设备

“系统信息”窗口底部显示了一系列可能的设备，显示出适合本应用程序的xCORE多核心微控制器。

6.3 xSOFTip组件配置

一些组件具有可配置选项，只要组件加入系统配置窗口后，这些选项就可以变更。

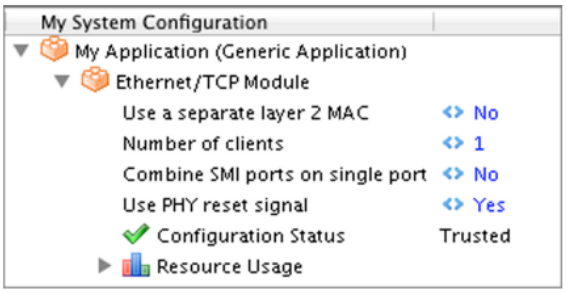


图5. xSOFTip浏览器可配置组件

改变配置时，也应更新“系统信息”窗口中的资源用途。

6.4 从应用程序中生成项目

可以在“系统配置”窗口从组件中自动生成项目。

1. 点击“系统配置”窗口顶部的“生成项目”按钮。

Shiqiang Xiao 13-11-7 1:03 PM
已删除: 心

Shiqiang Xiao 13-11-7 1:03 PM
已删除: 心

Shiqiang Xiao 13-11-7 1:03 PM
已删除: 心

Shiqiang Xiao 13-11-7 1:03 PM
已删除: 核

Shiqiang Xiao 13-11-7 1:03 PM
已删除: 核

Shiqiang Xiao 13-11-7 1:03 PM
已删除: 核

Shiqiang Xiao 13-11-7 1:04 PM
已删除: 比特

Shiqiang Xiao 13-11-7 1:04 PM
已删除: 比特

Shiqiang Xiao 13-11-7 1:04 PM
已删除: 使用时钟模块严格控制输入/输出引脚时间

Shiqiang Xiao 13-11-7 1:05 PM
已删除: 信道

Shiqiang Xiao 13-11-7 1:05 PM
已删除: 信道

Shiqiang Xiao 13-11-7 1:05 PM
已删除: 信道

Shiqiang Xiao 13-11-7 1:05 PM
已删除: 信道

Shiqiang Xiao 13-11-7 1:05 PM
已删除: 计时器

Shiqiang Xiao 13-11-7 1:06 PM
已删除: 软件使用计时器来控制事件发生时间

Shiqiang Xiao 13-11-7 1:06 PM
已删除: 计时器

2．在“生成项目”窗口输入项目名称。

3．从“目标硬件”列表选择开发板。

4．点击“完成”

xTIMEcomposer Studio为你所选择的xSOFTip生成项目。

xTIMEcomposer Studio生成项目的时候会改变至“编辑”视图。
xSOFTip以C代码的形式传送，因此你可以轻易更改它来满足要求并添加现有的C函数。

你可以通过“窗口”——“打开视图”菜单随时切换视图。

第D部分编译

内容

7 使用xTIMEcomposer Studio构建项目

在项目浏览器中选择需要编译的项目，点击“Build”按钮旁边的箭头，选择“Debu”或“Release”。

xTIMEcomposer使用项目中的Makefiles来确定编译器中使用的配置设定。

如果你的程序中没有错误，则xTIMEcomposer将编译的二进制文件添加至同一个工程的bin文件夹。

如果你的程序有错误，控制台会生成错误的报告，双击标红的信息，在编辑器中对错误的信息进行定位。

Shiqiang Xiao 13-11-7 1:48 PM
已删除: 构建

Shiqiang Xiao 13-11-7 1:48 PM
已删除: 调试

Shiqiang Xiao 13-11-7 1:48 PM
已删除: 释放

Shiqiang Xiao 13-11-7 1:49 PM
已删除: 项目

Shiqiang Xiao 13-11-7 1:49 PM
已删除: 二进制

Shiqiang Xiao 13-11-7 1:49 PM
已删除: 报告发生的

Shiqiang Xiao 13-11-7 1:50 PM
已删除: 其

8 XCC程序指令

xTimeComposer支持下列指令

#pragma unsafe arrays
（仅适用XC）本指令禁用运行时的安全检查，从而禁止在下一行动范围内援引无效的数组元素，用于当前函数语句。在函数以外，本程序可用于定义下一个函数。

Shiqiang Xiao 13-11-7 1:50 PM
已删除: 程序

#pragma loop unroll (n)
（仅适用XC）本程序控制下一行动的次数，而无论当前函数是否展开。n指出了显示的迭代数，只在优先级01或更高级别才可以展开。省略系数n将使编译器完全展开循环。函数之外本程序可以忽略，编译器在不能展开时将会发出警告。

#pragma stackfunction n
本程序为当前编译单元中的下一个函数语句分配n个字符（整数）的堆栈空间。

#pragma stackcalls n
（仅适用XC）本程序为下一个语句中的任意函数分配n个字符（整数）的堆栈空间。如果下一语句不包含函数调用，则下一语句将出现在另一个函数中。

#pragma ordered
（仅适用XC）本程序控制下一选择语句的编译。本选择语句这样编译：选择开始时如有多个事件就绪，则优先选择发生较早的事件。

#pragma select handler
（仅适用XC）本程序表示下一函数语句为选择处理程序，该选择处理程序可用于选择项中，如下面例子所示。

```
#pragma select handler
void f(chanend c, int &token, int &data);

...
select {
  case f(c, token, data):
    ...
    break;
}
...
```

作用域在于启用资源中的事件作为函数的第一个参数。事件发生时，选择处理器会在示例之前有效。

该选择处理器第一个参数必须为传输型，而不得为返回型参数。

如果资源为关联状态，例如为一种境况，则等候事件之前进行的选择不会改变该状态。

#pragma fallthrough
（仅适用XC）本程序表示下面多分支结构将会在无中断语句或返回语句的情况下落入下一个多分支结构。由于落入过程，本程序将会阻止编译器发生报警或故障。

#pragma xta label "name"
本程序提供标签可用于说明时间限制。

#pragma xta endpoint "name"
（仅适用XC）本程序规定了终点，常出现在输入或输出语句之前。

#pragma xta call "name"
（仅适用XC）本程序定义了（函数）调用点标签。使用本程序定义函数调用示例。例如，如果一个函数包含一个回路，则可根据函数调用点将该回路迭代值设定为一个不同的数值。

#pragma xta command "command"
（仅适用XC）本程序允许在源代码中嵌入XTA命令。每当二进制文件载入XTA时，所有命令都会运行。命令按照它们在文件中的顺序执行，但不同源文件中的命令顺序并未界定。

#pragma xta loop (integer)
（仅适用XC）本程序将给定环路XTA迭代应用于含程序的回路中。

Shiqiang Xiao 13-11-7 1:53 PM
已删除: 效用

Shiqiang Xiao 13-11-7 1:53 PM
已删除: 发生效用

Shiqiang Xiao 13-11-7 1:54 PM
已删除: 回传型

9 XCC命令行选项

本章内容：

- 总体选项
- 警告选项
- 调试选项
- 优化选项
- 预处理器选项
- 链接器和映射器选项
- 目录选项
- 影响XCC的环境变量
- 由<platform.h>提供的板支撑

XCC为xCORE C, C++和XC编译器的前面部分。典型应用包括在数据块上进行编码及数据的预处理、编译、组合、链接和映射。一些选项允许在中间阶段终止进程，而另一些选项则会直接进入一个处理阶段。大多数选项都具有否定形式（如-fno-选项）。选项与参数之间允许存在空间。

9.1 总体选项

编译的四种可能阶段分别为预处理、正确编译、组合和链接/映射。前三个阶段适用于单个源文件，以产生目标文件。链接和映射将目标文件和XN文件组合在一个可执行XE文件中，里面包含了每个Tile的数据和代码。

Shiqiang Xiao 13-11-7 1:58 PM
已删除: 其中每个数据块都包含代码和数据段。

source-file 源文件后缀决定其默认处理方式。

-xlanguage 规定了下列输入文件的语言。本选项适用于下一个-x选项之前的所有输入文件。语言支持值为：

xc
c
c++
assembler
assembler-with-cpp
xta
xn
xta
无（关闭语言规范）

拓展名	文件类型	由XCC进行预处理
.xc	XC源代码	Y
.c	C源代码	Y
.cpp	CPP源代码（为了兼容，可识别扩展名为cc, cp, c++, C和cxx的文件）	
.S	汇编码	Y
.xta	xCORE计时分析仪下标	N
.xn	xCORE网络描述	N
.xi	XC源代码	N
.i	C源代码	N
.ii	C++源代码	N
.s	汇编码	N
其他	发送至链接器的目标文件	N

-std=标准
规定了输入C或C++文件所用的不同语言。标准支持值为：

c89
ISO C89

gnu89
扩展名为GNU的ISO C89

c99
ISO C99

gnu99
扩展名为GNU的ISO C99（C程序默认）

c++98
ISO C++ (1998)

gnu++98
扩展名为GNU的ISO C99（C程序默认）

-fsubword-select
在XC中，允许在目的变量尺寸小于32位的通道输入中选择。

该程序为基于XS1-L设备的目的变量默认值，而非基于XS1-G设备的目的变量默认值。进一步详情，请看§36.3。

-target=platform
规定了目的平台，平台配置必须包含在文件platform.xn中，从而按照XCC_DEVICE_PATH环境变量（见§9.8）中规定的路径查询配置。

Shiqiang Xiao 13-11-7 2:01 PM
已删除: 信道

-fxscope 启用XScope追踪支持。目的变量的XN文件必须包括XScope链接。
编译和链接时传递该选项。

-funroll-loops
使用较少的迭代次数来展开环路，该选项在-O2及以上版本中启用。

-finline-functions
将简单函数集成在调用器上，该选项在-O2及以上版本以及-Os上适用。

-pass-exit-codes
返回编译各阶段产生的数值最大的错误代码（默认情况下返回1，如果编译器所有阶段都没有成功，则返回0）

-c	编译或汇编源文件，为每个源文件产生一个目的文件，但不会链接或映射。默认情况下，目标文件文件名将源文件的后缀改为.o（例如a.c产生a.o的目标文件）。
-S	正确编译后停止，此时每一个未汇编输入文件将产生一个汇编代码文件。默认情况下，汇编文件文件名将源文件的后缀改为.s。 不需要编译的输入文件忽略不计。
-E	只需对源文件进行预处理，将经预处理过的源文件输出为标准输出文档。 不需要预处理的文件忽略不计。
-ofile	在文件中加入输出模块。 如果未指定-o，则为可执行文件添加后缀a.xe，作为后缀名为.o源文件的目标文件处，汇编代码文件的源文件后缀名为.s，所有预处理C/C++/XC源文件都为标准输出文档。
-v	（以标准误差）打印编译各阶段中执行的命令。同时打印XCC，预处理器和正确编译器的版本号。
	与-v相同，不执行的命令和引用的命令参数除外。
-###	打印支持命令行的选项描述。如果指定了-v选项，则同样将--help传递至XCC调用的子进程中。
--help	显示版本号和版权。
--version	

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Zhang Wilson 13-11-26 4:41 PM
已删除: 128128

9.2 警告选项

许多警告都可以通过以-W开头的选项控制。下列选项都具有以-Wno-开头的否定形式以关闭警告：

-fsyntax-only
当出现语法问题时候确认代码错误，然后退出。

Shiqiang Xiao 13-11-7 2:05 PM
已删除: 出现语法错误时勾选该选项

-w 关闭所有警告信息。

-Wbidirectional-buffered-port
关于输入或输出缓冲端口不合格时发出警告。本警告默认启用。

-Wchar-subscripts
数组下标为Char类型时报警。

-Wcomment 评论开始序列/*出现在/*评论中，或者反斜杠换行符位于//评论中时报警，该项为默认项。

-Wimplicit-int
函数语句中未规定类型时报警；在C中，函数语句中无传回类型时报警。

-Wmain 主函数类型不带外部链接传回整数时报警；在XC中主函数没有零参数时报警。在C中主函数没有适当类型的零或二参数时报警。

-Wmissing-braces
联合初始器未加括号时报警。

-Wparentheses
上下文中赋予真值时如果省略括号，或者内部操作员混乱时会报警。

-Wreturn-type
函数为返回型且缺省为整数时，或返回类型不为空值但返回语句未返回任何值时报警。

-Wswitch-default
切换语句不含默认情形时报警。

-Wswitch-fallthrough
（仅适用XC）含一条以上语句的切换语句中的示例可落入下一示例。

-Wtiming 不符合时间限制时报警，这是默认项。

Zhang Wilson 13-11-26 4:41 PM
已删除: 128128

-Wtiming-syntax
计时脚本中有无效的句法。这是默认的。

-Wunused-function
如果声明静态功能但尚未定义或未使用非在线静态功能，则会出现此警告。

-Wunused-parameter
将功能参数用于非声明用途时会产生此警告。

-Wunused-variable
如果本地变量或非常数静态变量未用于声明用途， 则会产生此警告。

-Wunused 同-Wunused-function、-Wunused-variable 及-Wno-unused-parameter。

-Wall 打开所有上述-W选项。
-Wall未暗示以下-W... 选项。
-Wextra
-W 打印针对以下方面的外警告消息：

- 返回数值或不返回数值的函数 （仅C、C++）。
- 逗号表达的表达声明或左边侧无不良影响。此警告可以通过将未使用的表达无效以抑制此警告（仅C、C++）。
- 无符号的数值与零的比较结果，使用符号<或<=。
- 存储类说明符（如static）不是声明中最重要的（仅C、C++）。
- 出现比较，如x<=y<=z（仅XC）。
- 函数的返回类型有一个冗余限定词，如const。
- 如果还规定了-Wall 或-Wunused，有关未使用的参数的警告。
- 将有符号数值转变成无符号数值时，有符号和无符号数值的比较可能产生不正确的结果。（如果还规定了-Wno-sign-compare，无警告。）
- 集合的 初始化程序并不初始化所有元素。
- 使用指定的初始化程序时，覆盖初始化框时不会产生不良影响（仅C、C++）。
- K&R- style函数中声明的函数参数，无类型说明符（仅C、C++）。
- if或else描述中的空主体（仅C、C++）。

Zhang Wilson 13-11-26 4:41 PM
已删除: 128128

- 指示犬与整数零比较，使用<、<=、>或>=等表示（仅C、C++）。
- 计数器和非计数器均出现在有条件的表达中。（仅C++）。
- 非静态参考或非静态const 计数器和非 计数器均出现在条件表达式中（仅C++）。
  - 模棱两可的虚拟基底 （仅C++）。
  - 给有声明寄存器的阵列加下标（仅C++）。
  - 采集声明寄存器的变量的地址（仅C++）。
  - 在得到的种类副本构造器中未初始化基本种类（仅C++）。

-Wconversion
如果负整数常数表达含蓄的转化成无符号类型，则会产生此警告。

-Wdiv-by-zero
如果compile-time整数被零除，则会默认产生此警告。

Shiqiang Xiao 13-11-7 2:07 PM
已删除: 这是默认的。

-Wfloat-equal
如果在相等性比较中使用浮点数值，则会产生此警告。

-Wlarger-than-*len*
如果对象大于定义的 *len*字节，则会产生此警告。

-Wpadded 如果结构中含有填充字节，则会产生此警告。（可以重新排列结构中的框以减少填充字节，从而使结构更小。）

-Wreinterpret-alignment
当重新解释导致更大的对齐时，会产生此警告。

-Wshadow 如果本地变量遮蔽另一个本地变量、参数和全局变量或如果内置函数被遮蔽，则会产生此警告。

-Wsign-compare
将有符号的数值转变成无符号的时，如果有符号和无符号数值比较会产生不正确的结果，则会产生此警告。

-Wsystem-headers
打印在系统数据块文件中发现的结构的警告消息。这不是默认的。参见§9.7。

-Wundef 如果在#if 指令中使用未定义的宏，则会产生此警告。

-Werror 将所有警告都当做错误处理。

Zhang Wilson 13-11-26 4:41 PM
已删除: 128128

-Werror=*option*
将一条警告消息转变成错误。选项应为前缀为-W的编译程序的警告选项之一。
默认设置为flag -Werror=timing-syntax 。将此警告转变成错误意味着计时警告（-Wtiming）也错误，反之亦然。

9.3 调试选项

-g 生成调试信息。

-fxta-info 生成与XTA一起使用的计时信息。这是默认的。

-fresource-checks
生成可执行代码，以便在资源分配故障时诱捕，从而可以尽早检测出资源错误。

-save-temps 保存中间文件。这些文件放在当前目录中，并根据源文件命名。

-fverbose-asm
产生外的汇编信息，作为中间汇编程序文件中的备注。

-dumpmachine
打印目标机器并退出。

-dumpversion
打印编译程序版本并退出。

-print-multi-lib
打印从multilib目录名称到激活其的编译程序开关的映射。目录名称与开关之间使用“;”，且每个开关的开头为“@”，而不是“-”，多个开关之间无空格。
-print-targets
打印编译程序支持的目标平台。目标名称与-target选项的目标名称对应。

9.4 优化选项

打开优化，使编译程序尝试改善性能he/或编码大小，然而这样会牺牲汇编时间及调试程序的能力。

-O0 切勿优化，这是默认的。

Zhang Wilson 13-11-26 4:41 PM
已删除: 128128

-O
-O1 优化。尝试缩短执行时间和代码大小，而不执行任何需要大量汇编时间的优化。

-O2 进一步的优化。这些优化均不涉及space-speed 权衡。

Shiqiang Xiao 13-11-7 2:09 PM
已删除: 更多

-O3 更进一步的优化。这些优化可能涉及space- speed权衡；高性能偏好小代码大小。

Shiqiang Xiao 13-11-7 2:09 PM
已删除: 再多一些

-Os 将代码优化至最小。

Shiqiang Xiao 13-11-7 2:10 PM
已删除: 可以优化最小的代码大小

-fschedule 尝试记录指示以提高性能。这不是任何优化水平默认的。

9.5 预处理器选项

以下选项控制预处理器。

-E 仅预处理器，然后退出。

-D*name* 预定义名称为定义为1的宏。

-D*name*=*definition*
语汇基元化和预处理定义内容，就像在#define指令中出现的一样。

-U*name* 删除任何之前的*name*定义。
处理-D和-U选项以将之显示在命令行上。

-MD 输出到文件，适用于说明源文件依赖性的合适规则。依赖性文件的默认名称根据是否规定了-o选项进行定义。如果规定了-o，则文件名为后缀为.d的-o的参数的基础名称。如果未规定-o，则文件名为后缀为.d的输入文件的基础名称。文件名可以被-MF覆盖。

-MMD 同-MD，除了忽略对系统数据块的依赖性。

-MF *file* 规定写入依赖性信息的文件。

-MP 针对每个源文件的依赖性发出虚假目标。每个虚假目标不依赖任何东西。这些虚拟规则变通错误，从而使可以在不更新Makefile文件到匹配的条件下删除数据块文件。

Zhang Wilson 13-11-26 4:41 PM
已删除: 128128

-MT *file* 规定了依赖性生成发出的规则的目标。

9.6 连接程序 和映射程序选项

以下选项控制连接程序/映射程序。

-llibrary 链接时检索库。连接程序依照规定的次序检索并处理库和对象文件。检索的实际库名称为lib*library* .a。
所检索的目录包括任何规定的带有-L的。
库为元素为对象文件的档案文件。连接程序扫描档案中的元素，检索参考但并未定义的符号。

-nostartfiles
切勿与系统启动文件链接。

-nodefaultlibs
切勿与系统库链接。

-nostdlib 切勿与系统启动文件或系统库链接。

-s 删除所有符号表并从可执行转移信息。

-default-clkblk *clk*
使用*clk*作为默认时钟模块。时钟模块可以使用其在<xs1.h>中的名称或其资源编号规定。
启动代码打开默认时钟模块，将之配置为在不分割的条件下关闭基准时钟，并使之处于运行状态。XC中声明的端口最初连接到默认时钟模块。如果未规定此选项，默认时钟模块设置为XS1_CLKBLK_REF。

-Wm,*option* 将*option*作为一个选项传到连接程序/映射程序。如果*option* 中含有逗号，则在逗号处将之分割成多个选项。
想要查看全套高级映射程序选项，输入 xmap --help。

-X mapper *option*
将option作为一个选项传到连接程序/映射程序。带有参数 use –X mapper的选项需要通过两次。

-report 打印源使用的总结报告。

Zhang Wilson 13-11-26 4:41 PM
已删除: 128128

9.7 目录选项

以下选项规定了检索数据块文件和库的目录。

-I*dir* 添加*dir*到将要用于检索数据块文件的目录清单中。

-isystem*dir* 在所有目录规定为-I后，在*dir* 中检索数据块文件。
将之标记为系统目录。
编译程序抑制系统目录中的数据块文件的警告。

-iquote*dir* 仅在*dir*中检索带有#include "file" （不带有#include <file>）的数据块文件，然后检索-I规定的所有目录和系统目录。

-L*dir* 添加*dir*到将由-I检索的目录清单中。

9.8 影响XCC的环境变量

以下环境变量会影响XCC的操作。OS特殊路径分隔符（Windows为“;”，Mac和Linux为“：”）分离多路径。

XCC_INCLUDE_PATH
如果规定使用-I，待检索的目录的清单，但是在命令行出现带有-I的任何路径后。

XCC_XC_INCLUDE_PATH XCC_C_INCLUDE_PATH XCC_CPLUS_INCLUDE_PATH XCC_汇编程序_INCLUDE_PATH
这些环境变量中每一个仅在处理提名语言的文件时适用。如果使用-isystem 规定，变量规定了将要检索的目录清单，但是在命令行出现带有-isystem 的任何路径后。

XCC_LIBRARY_PATH
如果规定使用-L，待检索的目录的清单，但是在命令行出现带有-L的任何路径后。

XCC_DEVICE_PATH
待检索用于设备配置文件的目录清单。

XCC_EXEC_PREFIX
如果设置由编译程序执行的子程序前缀为此环境变量。当前缀与子程序的名称组合时，不得添加分离的目录。执行汇编程序或映射程序时前缀不适用。

Zhang Wilson 13-11-26 4:41 PM
已删除: 128128

XCC_DEFAULT_TARGET
默认目标平台，就像使用-target=规定那样定位。如果未规定带有-target=的目标且未通过XN文件，则使用默认目标平台。

9.9 <platform.h>提供的板支持

在汇编程序期间，编译程序生成一个叫做platform.h的临时头文件，其中含有变量和宏定义（由目标XN文件定义），其中包括：

Shiqiang Xiao 13-11-7 2:13 PM
已删除: 数据块

- 典型tileref变量类型的声明（参见§42.2）。
- 端口名称的宏定义（参见§42.4.2）。

10 使用XMOS Makefiles

本章内容

XMOS开发环境创建的项目的结构由Makefiles控制。这些Makefiles使用程序xmake（Gnu Make[1]的端口）执行构建。构建可以从XDE或直接从称为xmake的命令行直接执行。

你无需理解Gnu Makefile语言就可以使用XMOS工具开发语言。常用的**XMOS Makefile** 可以为项目、应用和模块提供支持。你只需要规则需要来构建**Makefiles项目和Makefiles应用**中

Shiqiang Xiao 13-11-7 2:14 PM
已删除: 应用

Shiqiang Xiao 13-11-7 2:14 PM
已删除: 项目

Shiqiang Xiao 13-11-7 2:15 PM
已删除: 规定

Shiqiang Xiao 13-11-7 2:15 PM
已删除: 在**项目**

Shiqiang Xiao 13-11-7 2:15 PM
已删除: 应用

Shiqiang Xiao 13-11-7 2:15 PM
已删除: 的应用

10.1 项目、应用和模块

应用由应用特有的源代码构成，或者由来自常用代码模块的源代码或二元库组成。在开发应用时，使用*工作空间*、*项目*、*应用和模块*说明工作区。

工作空间
工作空间为几个项目的容器。
项目
项目为可能含有几个应用和模块的目录及其他与特殊项目有关的文件。项目中可能含有特殊板的代码或参考设计，或为含有供其他项目使用的模块的软件组件。
应用
应用指的是含有源文件和Makefile（构建到单可执行（.xe）文件中）的应用。作为一项公约，应用目录的前缀为app_。这些应用出现在XDE项目浏览器中顶层水平。

[1] http://www.gnu.org/software/make/

IP模块

模块为含有源文件和/或二元库的目录。源自身并未构建到任何东西中，但是可以借助前缀为module_的模块目录，供application. 应用。

这些模块出现在XDE中项目浏览器的顶层水平。

10.1.1 结构示例

工作空间的结构示例如下所示。

```
sw_avb/
    app_avb_demo1/
    app_avb_demo2/
    module_avb1/
    module_avb2/
    doc/
sc_xtcp/
    module_xtcp/
    module_zeroconf/
sc_ethernet/
    module_ethernet/
```

此工作空间有三个项目：sw_avb、sc_xtcp和sc_ethernet。sw_avb项目含有两个应用，每个应用构建成单独的二元形式。这些应用可以使用项目中模块的源，且可以使用表格来自其自己的项目（module_avb1和module_avb2）和其他项目（module_xtcp, module_zeroconf和module_ethernet）的模块。

或者，可以使用以下方式构建工作空间：

```
app_avb_demo1/
app_avb_demo2/
module_avb1/
module_avb2/
doc/
module_xtcp/
module_zeroconf/
module_ethernet/
```

在本例中，所有应用和模块均在工作空间的顶层。

10.2 Makefile应用

每个应用目录均应含有一个叫做Makefile的文件，其里面含有常用的XMOS的Makefile文件。常用Makefile控制编译，且默认包含应用目录及其子目录中的所有源文件。应用Makefile支持以下变量分配。

Shiqiang Xiao 13-11-7 2:18 PM
已删除: 应用

Shiqiang Xiao 13-11-7 2:18 PM
已删除: XMOS

Shiqiang Xiao 13-11-7 2:18 PM
已删除: 控制构建

XCC_FLAGS*[_config]*
规定在构建期间到xcc的标记。此选项设置特殊构建配置*config*的标记。如果未给出后缀，它为默认构建配置设置标记。

XCC_C_FLAGS*[_config]*
如果设置了，所有.c文件的这些标记发送到xcc而不是XCC_FLAGS。此选项设置用于特殊构建配置*config*的标记。如果无后缀，它设置默认构建配置的标记。

XCC_ASM_FLAGS*[_config]*
如果设置了，所有.s或.S文件的这些标记转到xcc而不是 XCC_FLAGS 。此选项设置用于特殊构建配置*config*的标记。如果无后缀，它设置默认构建配置的标记。

XCC_MAP_FLAGS*[_config]*
如果设置了，在最终链接阶段，这些标记转到xcc而不是XCC_FLAGS。此选项设置用于特殊构建配置*config*的标记。如果无后缀，它设置默认构建配置的标记。

XCC_FLAGS_*filename*
为规定的文件名重写转到xcc的标记。此选项重写所有构建配置的标记。

VERBOSE 如果设置为1，则激活从make系统的冗长输出。

SOURCE_DIRS 规定了与汇编内容的应用目录有关的目录清单。默认包含所有目录。

INCLUDE_DIRS
规定待查找的目录，包括在构建期间创建的文件。默认包含所有目录。

LIB_DIRS 规定在构建期间链接到应用的库的目录。默认包含所有目录。

EXCLUDE_FILES
规定未汇编到应用中的源文件名称的空格分隔清单（不包括其路径）。

USED_MODULES
规定汇编到应用中的模块目录的空格分隔清单。给出的模块目录中不得有其全部路径，而不论其来自哪个项目，例如：

Shiqiang Xiao 13-11-7 2:20 PM
已删除: 模块

```
USED_MODULES = module_xtcp module_ethernet
```

MODULE_LIBRARIES
此选项规定了从规定了多个的模块得到的优选库清单。详情参见参见§11。

10.3 Makefile项目

Shiqiang Xiao 13-11-7 2:22 PM
已删除: 项目

不仅每个应用有自己的Makefile，项目也应有一个顶层Makefile。此Makefile控制项目内应用的构建，并赋予其一个变量以完成此任务：

BUILD_SUBDIRS

规定将要构建的应用目录的空格分隔清单。

10.4 module_build_info文件

每个模块目录应含有一个名称为module_build_info的文件。此文件通知如何在模块中构建文件的应用（如果应用的结构中有模块）。它可以选择含有以下分配变量中的几个。

DEPENDENT_MODULES
规定模块的依赖性。当应用包含一个模块时，它还包括其所有依赖性。

Shiqiang Xiao 13-11-7 2:22 PM
已删除: 模块

MODULE_XCC_FLAGS
规定在从当前模块汇编源文件时转到xcc的选项。此定义可以参考应用Makefile的XCC_FLAGS变量，例如：

```
MODULE_XCC_FLAGS = $(XCC_FLAGS) -O3
```

MODULE_XCC_XC_FLAGS
如果设置了，此模块中的所有.xc文件的这些标记转到xcc而不是 MODULE_XCC_FLAGS。

MODULE_XCC_C_FLAGS
如果设置了，此模块中的所有.c文件的这些标记转到xcc而不是 MODULE_XCC_FLAGS。

MODULE_XCC_ASM_FLAGS
如果设置了，此模块中的所有.s或.S文件的这些标记转到xcc而不是 MODULE_XCC_FLAGS。

OPTIONAL_HEADERS
规定特殊数据块文件为可选配置数据块。此数据块文件并不存在在模块中，而是由使用该模块的应用提供。如果此文件中有应用，则构建系统将特殊macro __ filename_h_exists__ 转到xcc。如果未提供文件，这样可以使模块提供默认配置数值。

11 使用XMOS Makefiles创建二元库

本章内容

在系统中默认的是将模块的源代码文件需要通过makefile的包含来用于XMOS的应用中。其实也可以通过将模块编译成二进制库使用，而无需源代码元件。

对一个模块进行编译并到二进制库中需要分成源以构建库和源/包括将要在库中分布的。例如，你可以设计以下架构。

```
module_my_library/
       Makefile
       module_build_info
       libsrc/
          my_library.xc
       src/
          support_fns.xc
       include/
          my_library.h
```

此结构的目的是将源文件my_library.xc汇编到库中，且库将分布在src中，且包括目录（而不是libsrc目录）。

11.1 module_build_info文件

当要构建二进制库时，需要在module_build_info文件中设置一些额外变量。必须设置其中一个库或库变量。

LIBRARY　　此变量规定了将要创建的库的名称，例如：

```
LIBRARY = my_library
```

LIBRARIES　　设置此变量而不是库变量以规定应构建的几个库（有不同的构建标记），例如：

Shiqiang Xiao 13-11-7 2:23 PM
已删除: 模块

Shiqiang Xiao 13-11-7 2:23 PM
已删除: 使用

Shiqiang Xiao 13-11-7 2:24 PM
已删除: The default module system used by XMOS应用makefiles默认使用的模块系统包括源代码水平的常用模块。

Shiqiang Xiao 13-11-7 2:25 PM
已删除: 但是，

Shiqiang Xiao 13-11-7 2:26 PM
已删除: 狗见

Shiqiang Xiao 13-11-7 2:26 PM
已删除: 到

Shiqiang Xiao 13-11-7 2:26 PM
已删除: 元

Shiqiang Xiao 13-11-7 2:26 PM
已删除: 以分布

Shiqiang Xiao 13-11-7 2:28 PM
已删除: 构建到二元的模块需要分成源以构建库和源

Shiqiang Xiao 13-11-7 2:29 PM
已删除: 规定

Shiqiang Xiao 13-11-7 2:29 PM
已删除: 结构

Shiqiang Xiao 13-11-7 2:29 PM
已删除: 沿线

Shiqiang Xiao 13-11-7 2:30 PM
已删除: 想

Shiqiang Xiao 13-11-7 2:30 PM
已删除: 元

```
LIBRARY = my_library my_library_debug
```

此清单中的第一个库为在应用包含在此模块中时将要链接的默认库。此应用可以通过添加其名称到其MODULE_LIBRARIES清单以规定其他库中的一个。

LIB_XCC_FLAGS_*libname*
此变量可以设置汇编库libname时转到xcc的标记。此选项额可用于将不同的汇编标记转到库的不同变量中。

EXPORT_SOURCE_DIRS
此变量应含有未汇编到库中的并作为源分布的目录的清单，例如：

```
EXPORT_SOURCE_DIRS = src include
```

11.2 Makefile模块的使用

Shiqiang Xiao 13-11-7 2:31 PM
已删除: 模块

构建到库中的模块可能有一个Makefile （与普通、仅源模块不同）。此Makefile的内容只需要为：

```
XMOS_MAKE_PATH ?= ../..
include $(XMOS_MAKE_PATH)/xcommon/module_xcommon/build/Makefile.library
```

Shiqiang Xiao 13-11-7 2:32 PM
已删除: 构建

此Makefile有两个目标。运行make all来建立库。调集目标make export将创建一份称为输出的目录中的模块的副本，该模块中报告不含有库的源文件。在上述示例中，输出模块如下：

```
export/
  module_my_library/
     module_build_info
     lib/
       xs1b/
         libmy_library.a
       src/
         support_fns.xc
       include/
         my_library.h
```

11.3 使用模块

应用可以依照使用源模块的方法使用库模块（包括USED_MODULE清单中的模块名称）。 可以使用会产生相同的最终结果的库源或输出模块。

第E部分

定时器

Shiqiang Xiao 13-11-7 2:35 PM
已删除: 计时

内容

0 使用xTIMEcomposer配置一个定时器程序

Shiqiang Xiao 13-11-7 2:35 PM
已删除: 给程序计时

12 使用xTIMEcomposer对程序进行时序仿真

本章内容

- 启动时序仿真分析仪
- 对代码进行部分仿真
- 规定时序仿真要求
- 添加程序执行信息
- 验证汇编期间的时序要求

xCORE计时分析仪使你能够确定在你的目标平台上执行代码所需要的时间。因为xCORE结构时钟的确定性，这些工具可以测试执行一部分代码的最短和最长时间。在与用户规定要求组合时，这些工具可以在汇编时间确定，是否所有代码的计时重要部分均由在其截止时间内执行的保证。

12.1 启动计时分析仪

依照以下步骤，在计时分析仪等控制下加载程序：

1. 在项目"Project Explorer"中选择一个项目。
2. 选择Run · Time Configurations。
3. 在左边的面板中，双击**XCore Application**。xTIMEcomposer创建了新的结构并在右边面板中显示默认设置。
4. xTIMEcomposer尝试识别目标项目可执行项目。需要选择其中一个时，单击项目文本框 的**Browse**，并在项目**Selection**对话框中选择项目。然后，单击**Search Project**和在对话框中选择**Program Selection**的可执行项目。

你必须在编译了你的程序并且在没有错误的情况下才可以使用这个选项

5. 在**Name**文本框中输入配置的名称。
6. 保存配置并启动计时分析仪，单击**Time**。xTIMEcomposer加载你的程序到计时分析仪中并在**Timing**透视图中打开。在此透视图中编辑器为只读的，目的是确保二进制文件和源代码之间的关系保持一致。

Shiqiang Xiao 13-11-7 2:35 PM
已删除: 给程序计时

Shiqiang Xiao 13-11-7 2:35 PM
已删除: 计时

Shiqiang Xiao 13-11-7 2:44 PM
已删除: 代码部分计时

Shiqiang Xiao 13-11-7 2:44 PM
已删除: 规定计时要求

Shiqiang Xiao 13-11-7 2:44 PM
已删除: 计时

Shiqiang Xiao 13-11-7 2:38 PM
已删除: 决定

Shiqiang Xiao 13-11-7 2:39 PM
已删除: 浏览器

Shiqiang Xiao 13-11-7 2:39 PM
已删除: 运行0 时间配置

Shiqiang Xiao 13-11-7 2:40 PM
已设置格式: 字体:非 加粗

Shiqiang Xiao 13-11-7 2:41 PM
已删除: 对话框中

Shiqiang Xiao 13-11-7 2:42 PM
已删除: 必须已经汇编了你的程序，而无任何可执行错误以供选择。

Shiqiang Xiao 13-11-7 2:45 PM
已删除: 二元

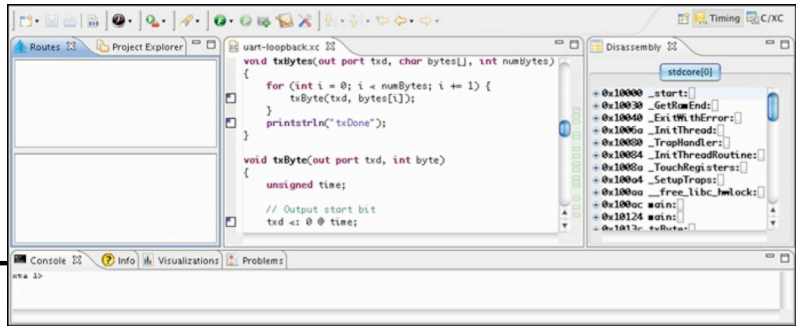


图 7：时序仿真透视图

Shiqiang Xiao 13-11-7 2:45 PM
已删除：<sp>计时透视图

xTIMEcomposer记住最后一次加载程序使用的配置。之后想要使用相同的设置加载XTA程序，则只需单击**XTA**按钮。想要使用不同的配置时，单击**XTA按钮**右边的箭头并从下拉清单中选择一个配置。

12.2 代码部分计时

*路径*由一套控制可以在程序中两点（或端点）之间流动的所有路径构成。每个路径有一个最佳案例时间，也即支路一直走执行时间最短的路线，对应最差案例时间。

依照下面的步骤规定路径并分析它：

1. 右击编辑器边缘的端点标记，并选择**Set from endpoint**。xTIMEcomposer的标记的右上四分之一显示一个绿色的点。

2. 右击一个端点标记并选择**Set to endpoint**。xTIMEcomposer的标记的右下四分之一显示一个红色的点你可以在端点上面规定一个起点。你还可以在端点处或其下面规定一个起点，定义；流式输出路径，然后返回到功能。这是称为multiple times或from within a loop的函数。

3. 单击主工具栏中的**Analyze Endpoints按钮**。xTIMEcomposer分析规定的路径中的所有路线，在**Routes**视图的下面使用树状展示，并在**Visualizations**视图的**Structure**标签使用图形展示。

或者，为了分析执行一个函数的时间，只需单击主工具栏中的**Analyze Function 按钮**，并从下拉菜单中选择一个函数。

xTIMEcomposer提供在汇编期间会保留次序的所有描述的端点标记。这些描述包括I/O 操作和函数调集。

12.2.1 可视化路径

Routes视图显示路径的结构展示。每次分析路径时，向顶部面板添加一个输入。单击路径以在底部面板查看它。使用以下节点展示它：

源水平函数。
依照顺序执行的节点的清单。
一套有条件执行的节点。
包含一系列节点的回路，其中最后一个节点可以从支路返回第一个节点。
含有指示的直线顺序的模块。
单一机器指示。

12.2.2 可视化视图

可视化视图提供了路径的图形展示。Structure标签使用从左到右的线条表示路径，如下面的例子所示。在代码分支时路径插入多路径，其所有路线的末端汇合。最佳案例计时路线使用绿色高亮表示，最差案例路线使用红色高亮表示，而所有其他路线均用灰色表示。

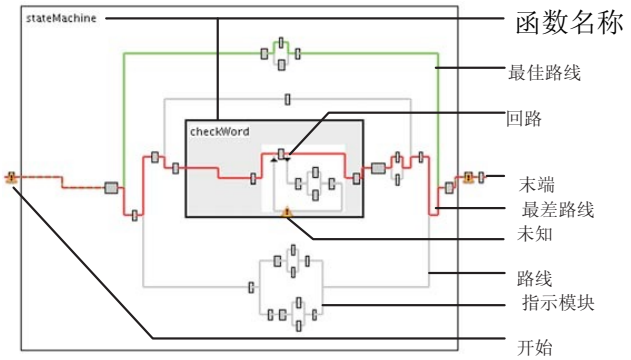


图 8：可视化视图

在**路径视图**和**结构 视图**中，你可以将鼠标放在节点上以显示其计时性质的总结。单击一个节点以高亮显示其在编辑器中的源代码，或双击进入节点起点的线条。在**结构**视图中，双击函数名称以扩展或收缩它。

12.3 规定计时要求

计时要求规定了路径中的路线让程序正确执行需要执行的时间。在**路径视图**的顶部面板中，每条路径的状态使用其名称左边的图标表示：


未规定计时要求。


规定并满足了计时要求。


规定并满足了计时要求，然而需要等待所有I/O指示准备就绪以执行。


规定然而并未满足计时要求。

右击路径并选择**Set timing requirements**一个对话框将打开，以规定计时要求。输入路线的最长执行时间（如*ns*、*cycles*或*MHz*）并单击**OK**。xTIMEcomposer更新路径的状态。

12.4 添加程序执行信息

在一些条件下，无额外信息时计时分析仪无法证明计时。常见的条件的示例包括：

- 路径中含有可以暂停未知时间长度的I/O指示。
- 路径中含有数据相关退出条件的回路。
- 路线未满足计时要求，但是该路线仅因为错误条件而执行，因此对计时并不重要。

在这些情况下，你可以提供有关你的程序执行的及时分析仪的额外信息。有了此额外信息，分析仪可以证明满足路线的计时要求。你呢可以提供的信息包括：

- **回路迭代的次数：**右击回路节点并选择**Set loop iterations**以显示一个对话框。输入最大回路计数并单击**OK**。
- **I/O指示的最长暂停时间：**右击指示节点并选择**Set instruction time**以显示一个对话框。输入一个数值，选择时间/速率单位（如纳秒或MHz）并单击**OK**。
- **排除路径中的一条路线：** 右击节点并选择**Exclude**。

12.4.1 细化最差案例分析

默认计时分析仪认为路径始终遵循执行时间最长的支路。如果你知道不是这样，例如在模拟期间检查或对你的程序进行正式的分析，你可以改善分析仪所使用的参数。你可以作出的改善包括：

- **规定函数调集的绝对执行时间：** 右击函数节点并选择**Set function time**以打开一个对话框并单击**OK**。
- **规定路线的绝对时间：**按下Ctrl 不放（Windows，Linux）或⌘（Mac）不放并单击两个指示节点，然后右击并选择**Set path time** 打开一个对话框。输入时间并单击**OK**。
- **规定节点执行的次数：**默认分析仪假定节点的执行次数为其范围中每个回路计数的乘积。为了将迭代次数转变成一个绝对的数值，右击节点并选择**Set loop scope打开一个对话框**。选择**Make scope absolute**并单击**OK**。
- **规定回路中某条件的执行次数：**默认分析仪假定有条件的节点始终遵循执行时间最长的路线。为了规定执行有条件的目标的次数，右击目标节点并选择**Set loop path**迭代以打开一个对话框。输入迭代次数并单击**OK**。

12.5 编译时验证计时要求

在规定了程序的计时要求后（包括有关其执行的任何改善），你可以生成一个在汇编时间检查这些要求的脚本。

依照这些步骤，创建检查**Routes视图**中规定的所有计时要求的脚本：

1. 单击**Generate Script按钮**。

2.在**Script location**文本框中输入脚本的文件名。文件的扩展名必须为.xta。

3. 在**Pragma name**框中修订其数值，以变更添加到源文件的编译指示的名称。

4. 单击**OK**以保存脚本并更新你的源代码。xTIMEcomposer添加脚本到你的项目并在编辑器中打开它。它还使用脚本需要的任何编译指示，更新你的源文件。

下次完成你的程序时，检查计时要求，并在发生任何故障时报告为汇编错误。双击计时错误以查看脚本中的故障要求。

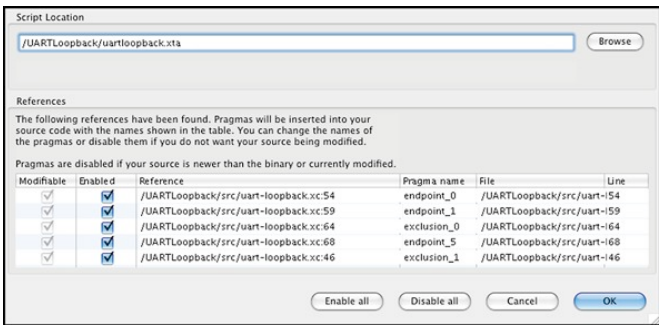

Script Location
/UARTLoopback/uartloopback.xta
Browse
References
The following references have been found. Pragmas will be inserted into your source code with the names shown in the table. You can change the names of the pragmas or disable them if you do not want your source being modified.
Pragmas are disabled if your source is newer than the binary or currently modified.
Modifiable
Enabled
Reference
Pragma name
File
Line
/UARTLoopback/src/uart-loopback.xc:54
endpoint_0
/UARTLoopback/src/uart-l54
/UARTLoopback/src/uart-loopback.xc:59
endpoint_1
/UARTLoopback/src/uart-l59
/UARTLoopback/src/uart-loopback.xc:64
exclusion_0
/UARTLoopback/src/uart-l64
/UARTLoopback/src/uart-loopback.xc:68
endpoint_5
/UARTLoopback/src/uart-l68
/UARTLoopback/src/uart-loopback.xc:46
exclusion_1
/UARTLoopback/src/uart-l46
Enable all
Disable all
Cancel
OK

图 9： 脚本选项对话框

第F部分

硬件上运行

内容

13 使用xTIMEcomposer运行程序

本章内容
- 创建一个运行配置
- 重新运行一个程序

xTIMEcomposer使用**Run Configurations**确定用于运行程序的设置。运行配置是项目和目标平台特有的。

13.1 创建一个运行配置

使用以下步骤创建一个运行配置：

1.在项目**Explorer**中选择一个项目

2. 选择**Run** ▶ **Run Configurations**。

3. 在左边的面板中双击**XCore Application**。
xTIMEcomposer创建一个新的配置并在右边的面板中显示默认设置，如图10所示。

4. 在**Name**中输入配置的名称。
5. xTIMEcomposer尝试为你识别目标项目和可执行项目。单击项目文本框右边的**Browse**并在项目**Selection**对话框中选择需要的项目。然后，单击**Search Project**，并在**Program Selection**对话框中选择可执行文件。

你必须之前已经编译了你的程序，且可选择的可执行程序无任何错误。
6. 如果你有一个连接到你的系统的开发板，则应检查**hardware**选项并从**Target**清单中选择你的调试适配器。
或者，检查**simulator**选项以在XMOS模拟器上运行你的程序。
7. 单击**Run**。

xTIMEcomposer加载你的可执行程序，并在**Console**显示你的程序生成的任何输出。

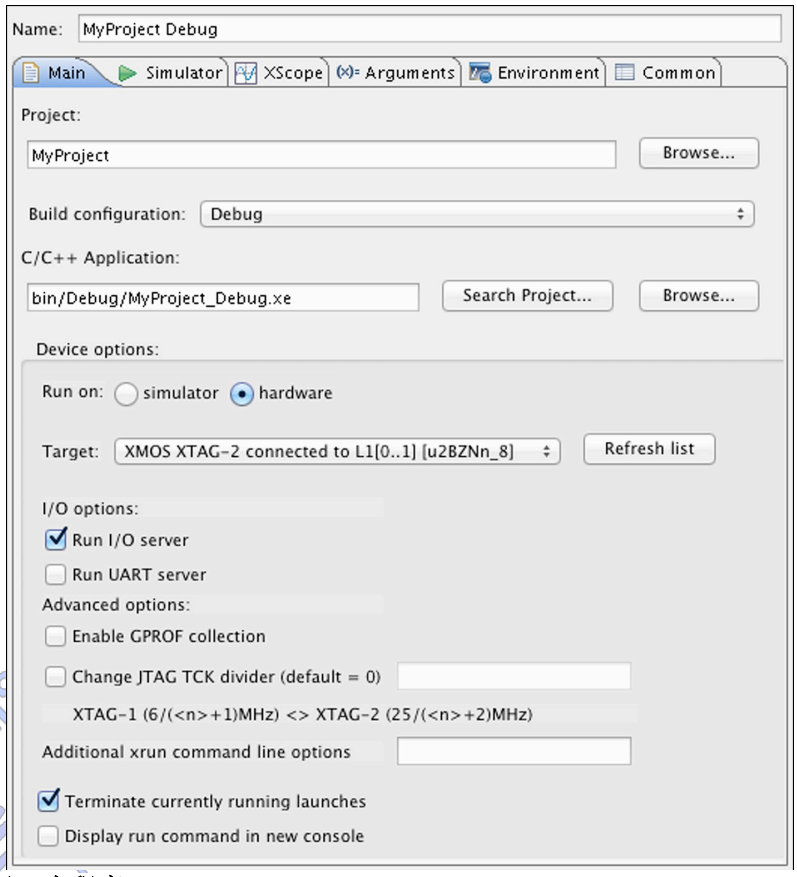


图 10：运行配置窗口

13.2 重新运行一个程序

xTIMEcomposer记住上次运行程序时使用的配置。想要再次使用相同的配置，只需单击**Run**按钮。想要使用不同的配置，单击**Run**按钮右边的箭头并从下拉清单中选择一种配置。

14XRUN命令行手册

本章内容

XRUN加载XMOS Executable （XE）文件到目标硬件并在上面运行。它需要安装XMOS或FTDI USB-to-JTAG 驱动器，具体视随目标硬件使用的适配器而定（参见§2）。

14.1 总体选项

以下选项被用于规定将要运行的可执行选项，且可以选择运行程序的xCORE展开图。

xe-file 规定加载和运行的XE文件。

--verbose 打印有关加载到目标设备的程序的信息。

--help 打印所支持的命令行选项的说明。

--version 显示版本号和版权。

14.2 目标选项

以下选项用于规定目标硬件平台。

--list-devices

打印连接到主机的所有JTAG适配器和每个JTAG链上的设备的编号清单，形式为：

ID	名称	适配器 ID	设备

适配器依照其序列号排序。

--list-board-info

-lb 显示有关所连接的目标板的信息。

--id *ID* 规定连接到目标硬件上的适配器。

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

--adapter-id *ADAPTER-SERIAL-NUMBER*
规定连接到目标硬件上的适配器的序列号。

--jtag-speed *n*
将JTAG时钟的驱动器设计为n。如有规定，默认数值为0，最大数值为70。
如果是基于FTDI的调试适配器，则JTAG时钟速度设置为6/（*n*+1）MHz。如果是基于XMOS的调试适配器，则JTAG时钟速度设置为25/（*n*+1）MHz。

--noreset 在加载程序前，切勿在JTAG扫描链上重置XMOS设备。这不是默认的。

14.3 调试选项

以下选项可用于激活调试能力。

--io 使XRUN在加载到程序后保持连接到JTAG适配器上，并激活主机的系统调集。当程序调集退出后XRUN终止。
默认在加载程序后，XRUN与TAG 适配器断开。

--uart 激活与XMOS USB-to-JTAG 适配器上的UART-to-USB转换器连接的UART服务器。转换器的操作速率为115200 bits/sec。
XMOS适配器上的USB-to-UART转换器在XSYS连接器上有两个插脚（在XMOS开发板上）连接到XMOS设备的端口上。端口在XN文件中的命名为PORT_UART_TX和PORT_UART_RX。
基于FTDI芯片的适配器不支持此选项。

--attach 连接到JTAG适配器（运行程序），使用主机激活系统调集。当程序执行调集退出时，XRUN终止。
必须使用此选项规定XE文件。

--dump-state
在JTAG扫描链中打印所有xCORE 展开图的核心、寄存器和堆栈内容。

14.4 XScope选项

以下选项可用于激活XScope能力。

--xscope 激活一个带有目标的XScope服务器。

--xscope-realtime
使用插座连接，激活带有目标的XScope服务器

--xscope-file *filename*
规定了用于XScope数据收集的文件名。

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

-xscope-port *ip：port*
规定了用于实时数据采集的IP地址和端口。

-xscope-limit *limit*
规定了XScope数据收集的记录极限。

Shiqiang Xiao 13-11-7 12:58 PM
已删除：--

Shiqiang Xiao 13-11-7 12:58 PM
已删除：--

第G部分

应用仪表和微调

内容

15 使用xTIMEcomposer和XScope实时追踪数据

本章内容
- XN文件配置
- 给程序装备仪表
- 启用追踪后的程序配置与运行
- 离线分析数据
- 实时分析数据
- 使用UART界面追踪

xTIMEcomposer和XScope库使你可以为程序装备实时收集应用数据的探针。此数据可以通过XTAG-2调试适配器发送到xTIMEcomposer以实时显示或写入文件以进行离线分析。

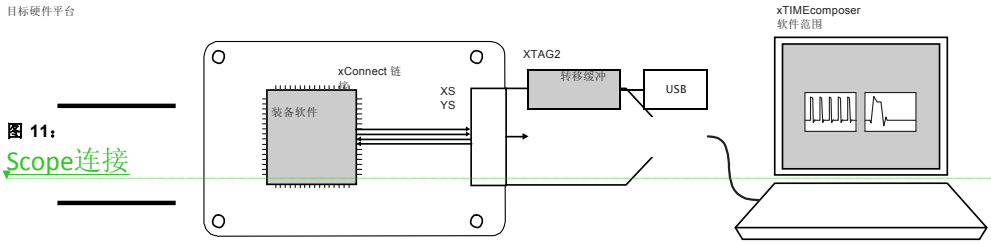


图 11： Scope连接

Shiqiang Xiao 13-11-8 2:57 PM
已删除： 范围连接性

如果你正在使用传统FTDI或XTAG-1调试适配器或你的目标硬件上的XSYS连接器未提供xConnect链接，则你可以配置探针，使之在你的适配器的UART界面输出追踪数据（参见XM-000957-PC）。需要说明的是，仅单展开图支持UART界面，并大大降低性能。

15.1 XN文件配置

为了使工具配置使用xscope进行高速数据收集需要的xConnect链接，板的XN文件必须配置与XTAG-2设备的连接。必须将以下信息添加到板的XN文件链接部分以设置目标设备使用的连接，用于与XTAG-2和xscope端倒通信。

Shiqiang Xiao 13-11-8 2:59 PM
已删除： 修订以暴露出

Shiqiang Xiao 13-11-8 2:59 PM
已删除： 以

Shiqiang Xiao 13-11-8 2:59 PM
已删除： 信道

```
<Link Encoding="2wire" Delays="4,4" Flags="XSCOPE">
  <LinkEndpoint NodeId="0" Link="XOLD"/>
  <LinkEndpoint RoutingId="0x8000" Chanend="1"/>
```

```
</Link>
```

需要说明的是，当链接设置为2线时，最短延迟设置为4且标志位规定了此链接将用于流水化调试。如果延迟数值设置的太高，将导致xscope数据包的输出频率更低。Routing Id也很重要，因为数值0x8000规定了这是专门用于xscope的特殊链接。

当用于多展开图系统时，必须规定连接到XSYS连接器程序包的节点识别符。工具使用其他展开图设置链接，但是它们需要知道那个设备有与XTAG-2的外部链接。

15.2 给程序装备仪表

图12中的示例程序使用XScope仪表函数追踪麦克风的输入水平。

```
#include <xscope.h>

port micL;
port micR;

void xscope_user_init(void) {
  xscope_register(2,
    XSCOPE_CONTINUOUS, "Microphone Left",  XSCOPE_UINT, "mV",
    XSCOPE_CONTINUOUS, "Microphone Right", XSCOPE_UINT, "mV"
  );
}

int main() {
  while (1) {
    int sample;
    micL :> sample;
    xscope_probe_data(0, sample);
    micR :> sample;
    xscope_probe_data(1, sample);
  }
}
```

图 12: 追踪麦克风的输入级别程序

构造器xscope_user_init寄存两个探针以追踪麦克风左声道和右声道输入。探针定义为**continuous**，也即xTIMEcomposer可以在两次连续测量结果之间插入数值。这些探针定义为采集典型unsigned int数值。

大体上来讲，程序每次调集探针函数xscope_probe_data时，就会从麦克风采集一次数据。此函数创建了一份追踪记录并将之发送到PC。

Shiqiang Xiao 13-11-8 3:00 PM
已删除: 标记

Shiqiang Xiao 13-11-8 3:01 PM
已删除: 使用的

Shiqiang Xiao 13-11-8 3:02 PM
已删除: 水平的

Shiqiang Xiao 13-11-8 3:02 PM
已删除: 寄存两个探针以追踪麦克风左边和右边的输入

图13总结了可以配置的各种类型的探针。仅**continous探针可以实时显示。**

探针类型	数据类型	Scope视图	示例
XSCOPE_CONTINUOUS	XSCOPE_UINT XSCOPE_INT XSCOPE_FLOAT	线图。可以插入。	电机控制器的电压水平
XSCOPE_DISCRETE	XSCOPE_INT	水平线	音频CODEC的缓冲水平
XSCOPE_STATEMACHINE	XSCOPE_UINT	状态机器	协议的进度
XSCOPE_STARTSTOP	XSCOPE_NONE XSCOPE_UINT XSCOPE_INT XSCOPE_FLOAT	启动/停止栏	记录功能输入和退出，带有可选标签数值

图13 支持的探针类型

15.3 启用追踪后的程序配置与运行

在装备你的程序的时候，你必须汇编并将之与XScope库链接，并离线或实时运行它。

依照下面的步骤链接XScope library和run XScope：

1. 打开你项目的Makefile文件 。
2. 找到你构建配置的XCC_MAP_FLAGS_*config*变量，例如XCC_MAP_FLAGS_Release。
3. 添加选项 –fxscope。
4. 为你的目标设备创建一个运行配置（参见§13.1）。
5. 单击**XScope**标签并选择**Offline Mode**以保存数据到文件以进行离线分析，或**Real-Time Mode**以输出数据到实时查看器。
   - 在离线模式下，xTIMEcomposer记录追踪数据直到程序终止并保存追踪数据到文件xscope.xmt。想要变更时，在**Output file**文本框中输入文件名。在**Limit records to**文本框中输入数字以限制追踪文件的大小。
   - 在实时模式下，xTIMEcomposer打开Scope视图并在执行程序时，显示追踪数据的动画视图。
6. 单击**Run**以保存并运行配置。

15.4 离线分析数据

双击项目*Explorer*中的追踪文件以在 **Scope视图**中打开，如图14所示。

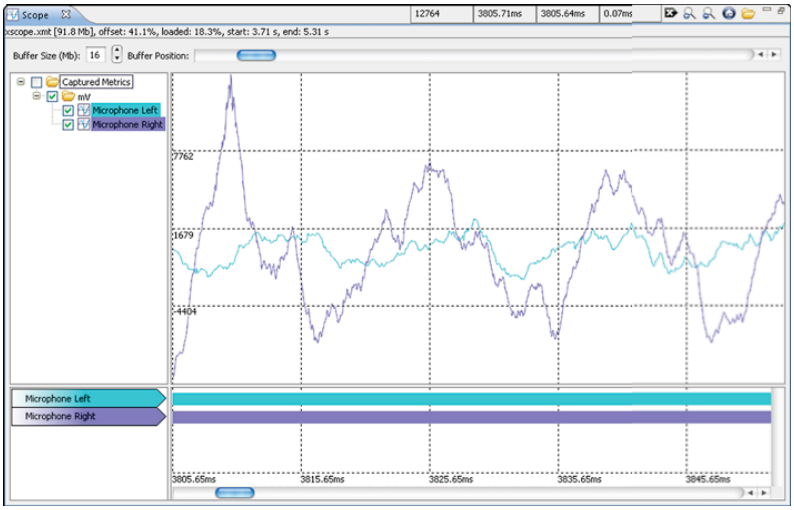


图 14： 离线Scope视图

Scope视图的顶部面板显示每个选中的探针的数值的图形：*x* –轴代表时间（依照底部面板的时限）和*y* –轴代表追踪到的数值。探针依照赋予其的单位分组，且多个单位相同的探针可以覆盖到同一个图上。

移动指针到范围数据上，显示窗口顶部四个数字框中前两个的当前数据（*y* –值）和时间（*x* –值）。单击视图以显示红线标记 – 相关的时间显示在第三个框中。第三个数字框显示标记时间与当前指针位置之间的差异。

如果指针变为手指标签，双击以确定负责生成追踪点的源代码中的描述。

此视图的底部面板显示每个探针的时限：探针时限上的垂直线表示探针创建记录的时间。

向左或右拖动**Buffer Position**滑块，穿过时限。升高**Buffer Size**框中的数值以在窗口中显示更多信息。

使用窗口顶部的**Scope视图**工具栏以执行额外任务：单击**Continuous points 按钮，**显示用于插入连续信号的数据点。

.

单击**Zoom Fit按钮**查看所有数据点。

单击**Open按钮**并浏览文件以加载并非你的项目的一部分的追踪文件。

15.5 实时分析数据

*Scope视图*可以实时显示从硬件流式输入的追踪数据。左边的面板显示信号信息和控制器，右边的面板显示信号的屏幕视图。

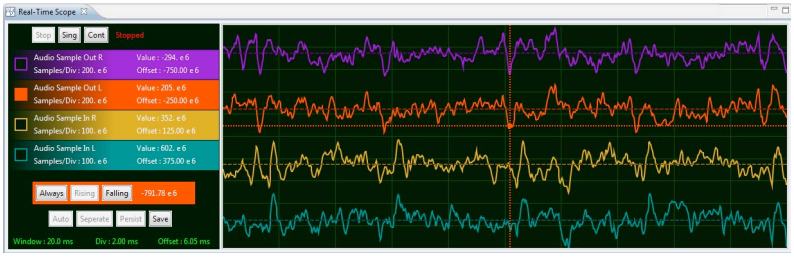


图 15：
实时Scope视图

左边的面板显示了应用寄存的*continous*探针的清单（参见§15.2）。每个命名的探针被赋予一种颜色，以在显示器上绘制事件并用于在屏幕面板中识别探针。

*Scope视图*基于传统示波器，并在触发器周围捕集数据和显示。捕集模式、显示模式、触发器和时间基准均在左边面板中控制。右边的面板中有10个水平和垂直部分，比例使用每个部分的单元数表示。

数值控制均可以使用鼠标修改：单击左边的按钮以升高数值，或单击右边的按钮以降低数值。如果你的平台太支持，则可以使用滚轮（Mac OS/X、Linux及部分而非全部Windows版本）。

15.5.1 捕集控制

有三种捕集模式：*continous*、*single capture*或*stopped*。默认的启动模式为系统连续捕集和显示。与捕集控制有关的标签显示XScope系统的当前状态。

图 16：
捕集控制器

停止显示

停止屏幕面板触发和捕集，设置此模式时无法继续更新屏幕。此模式可用于检查所捕集的数据。鼠标可用于变更新信号、时间基准比例和偏移（如下所述）以详细检查信号。停止后，你可以在时间基准上缩放并更加详细的查看信号：当时间基准较大，在时间基准上缩放将展示所有数据。

单次捕集

选择单次模式捕集一次屏幕数据就返回到停止状态。如果激活了触发器（参见图18），系统将等待满足此触发器条件，然后再更新屏幕并返回到停止状态。

连续捕集

选择自由运行模式以尽可能频繁的更新屏幕。如果触发器激活，则屏幕仅在满足触发器条件时更新。

15.5.2 信号控制

左边面板中显示的彩色标签上的每个注册探针可用的信号控制（参见图17）

图 17：
信号控制

激活/ 失活信号

双击名称切换新号的可见度。

信号样品/部分

使用鼠标按钮，更改此探针每个部分的样品数量，这影响信号的垂直比例。

信号屏幕偏移

使用鼠标按钮变更此探针的垂直偏移，这影响信号的垂直位置。

信号触发器

单击探针标签左边的触发器箱，可将信号用作触发器（参见图18）。只有一个信号可用于触发。

15.5.3 触发控制

触发器可用于限制系统，从而使仅在满足条件时捕集数据。默认所有触发器均失活，导致可以无条件的捕集数据。为了激活触发，必须单击探针标签左边的框以选择触发器。

当触发激活时，屏幕上出现十字线以显示触发器水平（相对于所选择的触发器的信号）和触发器在时间基准上的偏移。十字线的中心为触发器发生（现在或之前）的时间和数值；其左边为触发器之前的信号，而右边为触发器后面的信号。

可以直接单击右边的窗格直接设置触发器水平和偏移。变更仅在范围未停止时生效，而不论是连续运行还是设置为单一触发。

图 18：
触发控制器

始终

失活触发器并无条件的捕集数据。

升高

触发信号的升高边缘。当选择将要用于触发的信号时，这是默认模式。

下降

在信号的下降边缘触发。

与激活触发器有关的数值标签显示信号的当前数值设置。这可以使用鼠标按钮变更。

15.5.4 时间基准控制

时间基准控制用于设置信号捕集窗口的时间范围，使你能够缩放和移动水平轴。

图 19：
时间基准控制器

时间窗口

时间窗口的当前尺寸。缩放时间中的所有信号并影响每个部分的时间。

每个部分的时间

每个部分的时间单元。缩放时间中的所有信号并影响时间窗口。

时间窗口偏移

触发器在时间窗口中的位置。向左和向后转移所有信号。需要说明的是，触发器不可见，且应在时间窗口的左边或右边。仅在有限的数值下，信号向右移动。

15.5.5 屏幕控制

有几个命令可用于操作所有信号。

图 20：
屏幕控制器

自动范围信号

自从布置所有当前信号以与屏幕匹配。短时测量信号，每个信号缩放并偏移以与屏幕匹配。显示的所有信号均交叉。

单独的信号

与Auto Range类似，但是所有信号均缩放以与屏幕的一小部分匹配。所有信号均偏移，因此可以单独查看它们。

持久显示器

失能。

Shiqiang Xiao 13-11-8 3:11 PM
已删除： 失活

保存数据

保存当前scope视图到用户定义位置的PNG文件中。

15.6 使用UART界面追踪

如果你正在使用传统的FTDI或XTAG-1调试适配器，或如果你的目标硬件上的XSYS连接器未提供xConnect链接，则你可以通过你的适配器提供的UART界面输出数据。

想要使用UART界面，你必须向XScope库提供一个使用连接到你的调试适配器上的UART-TX插脚初始化的1bit的UART TX端口。初始化的示例如下。

Shiqiang Xiao 13-11-8 3:11 PM
已删除: 比特的

```
#include <platform.h>
#include <xscope.h>

port uart_tx = PORT_UART_TX;

void xscope_user_init(void) {
  xscope_register(2,
    XSCOPE_CONTINUOUS, "Microphone Left",  XSCOPE_UINT, "mV",
    XSCOPE_CONTINUOUS, "Microphone Right", XSCOPE_UINT, "mV"
  );
  xscope_config_uart(uart_tx);
}
```

因为UART界面使用端口，而不是xConnect链接，因此只需一个扩展图就可以调集探针函数。

16XScope性能特点

本章内容

从xCORE设备转移到调试适配器的数据无损失，但是从调试适配器转移到你的主机PC的数据可能会出现损失，具体视你的PC的速度而定。

16.1 xCORE Tile和XTAG-2之间的转移速率

所建议的大部分目标硬件的过渡之间的xConnect链接速度为10ns（10MByte/sec）。这可以通过设置链接比特间间隙到5个周期实现（参见§42.5）。此速度下使用xConnect链接的探针函数的传统和最大调集速度见图21。

图 21： 带有5循环比特间隙的xConnect 链接XScope 性能数据

探针函数	等待时间（核心循环）最大调集次数/秒	
xscope_probe_data_pred	15 （始终）	666,000
xscope_probe	20 （没有连接）	999,000
xscope_probe_cpu	27 （没有连接）	666,000
xscope_probe_data	22 （没有连接）	666,000
xscope_probe_cpu_data	28 （没有连接）	555,000

如果连续调集两次，则第二次调集可能依照最大频率延迟。例如，如果调集xscope_probe_data_pred两次，第二次调集延迟约1.5μs。

可以通过加速链接并缩小比特间间隙以增加最大调集速度（参见§42.5）。小比特间间隙需要认真布置链接，因为它会提高链接频率。

UART界面执行速率为2MB/s。

16.2 XTAG-2和Host PC之间的转移速率

很多个人计算机从XTAG-2输入追踪数据仅限于以500,000项追踪记录/秒或更低的速度。如果你的个人计算机无法保持此速率，则可能丢失记录，降低追踪该数据的间隔尺寸。XDE Scope视图在时间线上标记数据损失。

Shiqiang Xiao 13-11-8 3:12 PM
已删除: 数据

Shiqiang Xiao 13-11-8 3:13 PM
已删除: 转移

Shiqiang Xiao 13-11-8 3:13 PM
已删除: 转移

Shiqiang Xiao 13-11-8 3:13 PM
已删除: 无争用

Shiqiang Xiao 13-11-8 3:13 PM
已删除: 无争用

Shiqiang Xiao 13-11-8 3:14 PM
已删除: 无争用

Shiqiang Xiao 13-11-8 3:14 PM
已删除: 无争用

Shiqiang Xiao 13-11-8 3:14 PM
已删除: 从XTAG-2输入追踪数据

17Xscope库API

本章内容
0 函数
0 列举

17.1 函数

void xscope_config_uart（port id）
配置XScope UART端口

必须在 xscope_register前调集。

此函数具有以下参数：

id UART port id。

void xscope_config_io（xscope_IORedirectionMode模式）
配置XScope I/O重新定向。

此函数具有以下参数：

mode I/O重新定向模式

void xscope_probe（无符号char id）
添加特殊事件的记录。

此函数具有以下参数：

id Probe id

void xscope_probe_cpu（无符号char id）
添加用于特殊事件的记录，带有cpu追踪功能。此函数具有以下参数：

id Probe id。

void xscope_probe_cpu_data（unsigned char id，unsigned int data）
使用cpu追踪和用户数据添加特殊事件的记录。此函数具有以下参数：

id Probe id。

Shiqiang Xiao 13-11-8 3:15 PM
已删除: 端口

Shiqiang Xiao 13-11-8 3:15 PM
已删除: 端口

Shiqiang Xiao 13-11-8 3:15 PM
已删除: 探针

Shiqiang Xiao 13-11-8 3:16 PM
已删除: 探针

Shiqiang Xiao 13-11-8 3:16 PM
已删除: 无符号

Shiqiang Xiao 13-11-8 3:16 PM
已删除: 无符号

Shiqiang Xiao 13-11-8 3:16 PM
已删除: 数据

Shiqiang Xiao 13-11-8 3:16 PM
已删除: 探针

data 用户数值。

void xscope_probe_data（unsigned char id，unsigned int data）

使用用户数据添加特殊事件的记录。此函数具有以下参数：

id 探针id。

data 用户数值。

void xscope_register（int num_probes,...）

使用主机系统寄存追踪探针。

第一个参数为将要注册的探针。其他参数每组四个。

示例：

```
xscope_register(1, XSCOPE_DISCRETE, "A probe", XSCOPE_UINT, "value"); ``
xscope_register(2, XSCOPE_CONTINUOUS, "Probe", XSCOPE_FLOAT, "Level",
                   XSCOPE_STATEMACHINE, "State machine", XSCOPE_NONE, "no
                     ↪ name");
```

此函数具有以下参数：

num_probes 将规定的探针数量。

void xscope_set_register_location（int location）

17.2 列举

xscope_IORedirectionMode

列举所有I/O重新定向模式。此类型具有以下数值：

XSCOPE_IO_NONE
I/O未重新定向。

XSCOPE_IO_BASIC
基础I/O重新定向。

XSCOPE_IO_TIMED
计时I/O重新定向。

xscope_UserDataType

列举所有用户数据类型。

此类具有以下数值：

Shiqiang Xiao 13-11-8 3:17 PM
已删除: 无符号

Shiqiang Xiao 13-11-8 3:17 PM
已删除: 无符号

Shiqiang Xiao 13-11-8 3:17 PM
已删除: 等

Shiqiang Xiao 13-11-8 3:17 PM
已删除: 位置

XSCOPE_NONE	无用户数据。
XSCOPE_UINT	无符号 int用户数据。
XSCOPE_INT	有符号的int 用户数据。
XSCOPE_FLOAT	浮动点用户数据。

xscope_EventType

列举所有类型的xscope事件。这种类型具有以下数值：

XSCOPE_STARTSTOP
启动/停止 - 事件得到代表执行模块的启动和停止数值。

XSCOPE_CONTINUOUS
连续 - 仅启动一个事件，单时间戳“ping”。

XSCOPE_DISCRETE
离散 – 自上一个事件起，事件生成一个离散模块。

XSCOPE_STATEMACHINE
状态机器 - 为每个新数值创建一个新事件状态。

第H部分

仿真

Shiqiang Xiao 13-11-8 3:18 PM
已删除: 模拟

内容

Shiqiang Xiao 13-11-8 3:19 PM
已删除: 模拟程序

18 使用xTIMEcomposer对程序进行仿真

Shiqiang Xiao 13-11-8 3:19 PM
已删除: 模拟程序

本章内容
- 模拟器配置
- 信号追踪
- 设置回路
- 配置模拟器插件

xCORE模拟器提供了使用一个或多个xCORE设备构建的接近准确循环的系统模型。你可以使用此模拟器查看处理器指示追踪、可视化系统状态及配置回路以模拟连接到XMOS端口和链接的组件的行为。

18.1 模拟器配置

使用下面的步骤配置模拟器：

1. 在项目**Explorer**中选择一个项目。
2. 选择**Run** ▶ **Run Configurations**。
3. 在左边的面板双击**XCore Application**。xTIMEcomposer创建一个新配置并在右边的面板显示默认设置。
4. 在右边的面板的**Name**中输入配置的名称。
5. xTIMEcomposer尝试尝试为你识别目标项目和可执行项目。单击项目文本框右边的**Browse**并在项目**Selection对话框**中选择需要的项目，然后单击**Search Project**并在**Program Selection对话框**中选择可执行的文件。你必须之前已经汇编了你的程序，且可选择的可执行文件无任何错误。
6. 选择**simulator**选项并单击**Simulator**标签以配置额外选项，如图22所示。

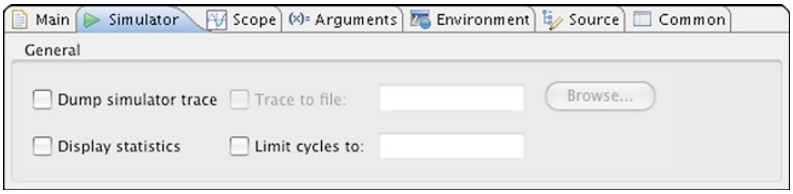


图 22：
模拟器配置选项

- 选择**Dump simulator trace**以在模拟期间输出处理器指示追踪。
  默认为在**Console**.中显示指示追踪。如果想要将追踪写入文件中，则应选择**Trace to file**并输入文件名。该文件名必须与你的项目中的所有其他文件都不同。
  指示追踪的格式如图26所示。
- 想要在程序终止时查看程序执行总结时，选择**Display statistics**。该总结包括每个逻辑核心的指示计数及通过开关发送的数据和进行的控制的次数。
- 为了限制模拟器所执行的循环的次数，则应在文本框**Limit cycles to**中输入一个数值。如果你想程序从始到终运行，则应留白。这对无线循环中的模拟程序有用。

7. 想要保存和运行配置，单击**Run**。

xTIMEcomposer加载你的可执行文件，并在**Console**中显示你的程序所生成的任何输出。

 xTIMEcomposer记住上次运行程序时使用的配置。想要再次使用相同的配置，只需单击**Run**按钮。想要使用不同的配置，单击**Run**按钮右边的箭头并从下拉清单中选择一种配置。

18.2 信号追踪

仿真器能够输出信号追踪数据到VCD文件，以便你使用xTIMEcomposer波形查看器可视化。

Shiqiang Xiao 13-11-8 3:30 PM
已删除: 模拟器

18.2.1 启用信号追踪

遵循下面的步骤激活模拟期间的信号追踪：

1. 创建一个模拟器**Run Configuration** （参见§18.1）。

2. 在**Signal Tracing**面板的**Simulator**标签中选择**Enable signal tracing**。
   - 想要追踪所有I/O插脚，在**System Trace选项**组中选择**Pins**。
- 想要追踪特殊核心的机器状态，在**Core Trace选项**组中单击**Add**以显示一组可配置下拉菜单和复选框。然后，选择你想追踪的核心和机器。你可以追踪过程循环、端口、核心、时钟模块、垫和处理器指示。

3.单击**Run**。

xTIMEcomposer加载你的程序到仿真器，并在终止时，添加所生成的VCD文件到你的项目。

Shiqiang Xiao 13-11-8 3:30 PM
已删除: 模拟器

18.2.2 查看追踪文件

在项目**Explorer**中双击VCD文件以将之在**Signals视图**中打开，如图23所示。

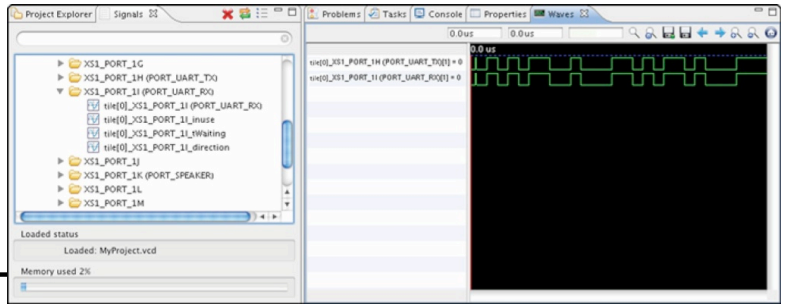


图 23：
信号和波形视图

在**Signals视图**中单击**plus sign** （Windows）或**disclosure triangle** （Mac）以扩展文件夹并显示其信号或子文件夹集合。双击信号或整个文件夹以在**Waves视图**中显示。


单击**Display**按钮以在层次和平面视图之间切换。

18.2.3 查看信号

在**Waves视图**中移动指针到信号上以在Waves视图顶部的右边数值控制中开始仿真。如果指针编程一个手指标签，则应可以双击以确定负责驱动信号的源代码中的输出描述。使用**Waves视图**工具栏进行以下操作：

Shiqiang Xiao 13-11-8 3:31 PM
已删除: 模拟


单击**Zoom Fit按钮**查看整个波形图。


单击**Next和Previous按钮**以在选中的信号之间过渡。负责过渡的输出描述在编辑器中高亮显示。


单击**Search Transition按钮**打开一个对话框以检索特殊过渡，输入一个数值并单击**Find**。


单击**Write Session File按钮**并输入该文件的文件名以保存配置。在加载VCD文件到**Waves视图**中时，保存你的设置以便使用。单击**Read Session File按钮**以加载最近保存的设置文件。

你可以通过以下方式控制信号在**Waves视图**中的显示方式：

- **使用ASCII显示信号数值：**右击**Waves视图**中的信号以弹出右键菜单并选择**Data Format ▶ ASCII**。
- **在信号之间添加分隔符：** 右击Waves视图上的信号以弹出菜单并从中选择 **Add Separator**。
- **给分隔符命名：**右击分隔符以弹出菜单并选择**Name Separator**。在 **Name Separator对话框**中输入分隔符的名称并单击**OK**。
- **移动分隔符：**单击并拖动分隔符到需要的位置。

18.3 设置回路

你可以在模拟中连接任何两个端口或插脚，以模拟插脚之间的连接。依照以下步骤配置回路：

1. 创建模拟器**Run Configuration** （参见§18.1）。
2. 单击**Simulator**标签以显示模拟器配置选项。
3. 单击**Plugins**面板中的**Loopback**标签并选择**Enable pin connections**。
4. 在**Pin Connections**面板中单击**Add**。显示一个空的回路配置。回路由两套你可以用于两个不同的端口的选项组成。

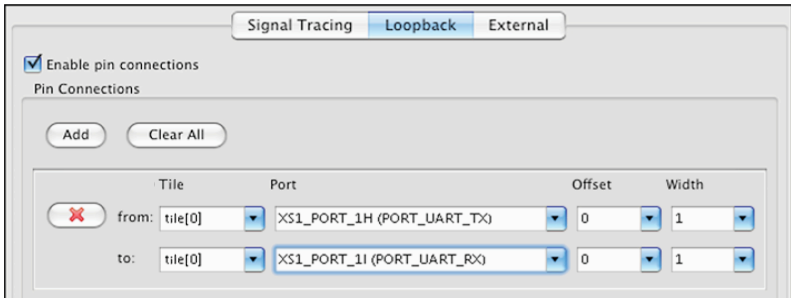


图 24：
设置回路连接

5. 在每个连接末端的下拉清单中选择**Tile和Port**的数值。如果你不规定展开图，则从项目的XN文件选择端口清单，并自动确定展开图。如果你规定了展开图，则从数据块文件中获取端口清单<xs1.h>。想要规定仅与端口连接的插脚的子集时，变更**Offset 和Width**的数值。

6. 单击**Run**。

18.4 配置仿真器插件

你可以连接模拟器到任何已经使用XMOS模拟器插入界面汇编到你的主机PC上的任何外部插入的模拟器。依照如下步骤配置外部插入：

1. 创建一个仿真器**Run Configuration** （参见§18.1）。

2. 单击**Simulator**标签以显示模拟器配置选项。

3. 在**Plugins**面板中单击**External标签。**

4. 单击**Add**以打开插入型配置对话框。

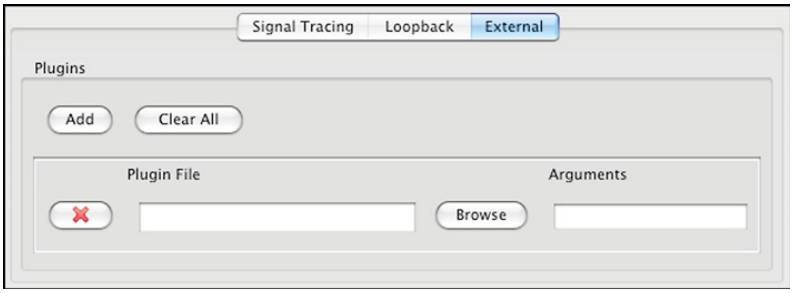


图 25：
设置外部插入

5. 选择插入DLL并规定可选命令行参数串。
6. 单击**Run**以保存你的设置，并使用规定的插入在模拟器上运行你的程序。

Shiqiang Xiao 13-11-8 3:35 PM
已删除：模拟器

Shiqiang Xiao 13-11-8 3:35 PM
已删除：创建一个模拟器

本章内容
- 总体选项
- 警告选项
- 追踪选项
- 回路插件选项

XSIM执行了XMOS Executable （XE）文件的基于周期的模拟。XE文件中含有对目标硬件的说明。

19.1 总体选项

xe-file 制定将要仿真的XE文件。

--max-cycles *n*
当达到*n*系统周期时退出。

--plugin *name args*
加载一个插入DLL。*Args*的格式由插入确定；如果*args*中含有任何空格，则必须将之附在引述中。

--stats 退出时打印以下内容：
- 每个逻辑核心的指示计数的明细表。
- 通过开关发送的数据和控制记号的数量。

--help 打印所支持的命令行选项的说明。.

--version 显示版本号和版权。

19.2 警告选项

--warn-resources
打印（有关标准误差）以下方面的警告信息：
- 计时输入或输出操作规定之前的时间。
- 在处理器输入前，重写缓冲端口的转移寄存器中的数据。

Shiqiang Xiao 13-11-8 3:36 PM
已删除: 规定

Shiqiang Xiao 13-11-8 3:36 PM
已删除: 模拟

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

-warn-stack

打开有关可能的堆栈破损的数据。
当XC任务尝试读或写另一个任务工作空间时，XSIM打印警告。如果使用#pragma stackfunction （参见§8）或#pragma stackcalls （参见§8）规定了任务的堆栈空间，则会出现此警告。

-no-warn-registers

当寄存器在写之前读时，不会发出此警告。

19.3 追踪选项

-trace

-t 打开所有展开图的指示追踪（参见图26）。

-trace-to *file*

打开所有开展图的指示追踪。追踪为*file*的输入。

-disable-rom-tracing

关闭从ROM执行的所有指示的追踪。

-enable-fnop-tracing

打开FNOP指示的追踪。

图 26： 追踪XS1处理器的输出

展开图	核心状态													地址		指示		存储	周期
从XN得到的名称	I0	I1	I2	S0 S1 （T0 ） .. S0 S1 （T*n* ）			.	M	S	K	N	PC	（ sym + 偏移地址 ） ：	名称	操作数	地址	@val
	-	*	-	-	-	*n* status		-	-	-	-				val	L[adr]	
	D	P	d	a A i l p m s w	b i e	pairs		m	s	k	n				*r*n （val） res[id]	S[adr]	

I0 : -	无调试中断	S1 : -	中断并失活事件
I0 : D	导致调试中断的指示	S1 : b	中断并激活事件
I1 : *	预期指示	S1 : **i**	激活中断且事件失活
I1 : P	暂停指示	S1 : e	中断失活且事件激活
I2 : -	未处于调试模式	M: -	MSYNC未设置
I2 : d	调试模式中的展开图	M: m	MSYNC设置
S0 : -	未使用的核心	S: -	SSYNC未设置
S0 : a	核心激活	S: s	SSYNC设置
S0 : A	核心激活（正在追踪的指示属于此核心）	K: -	INK未设置
S0 : **i**	使用ININT比特设置激活核心	K: k	INK设置
S0 : **I**	使用ININT比特设置激活核心（属于此核心）	N: -	INENB未设置
S0 : p	因为指示提取暂停的核心	N: n	INENB设置
S0 : m	使用MSYNC比特设置暂停的核心	**r***n* （val）	寄存器***n***的数值
S0 : s	使用SSYN比特设置暂停的核心	**res[id]**	资源识别符
S0 : w	使用WAITING比特设置暂停的核心	L/S[adr]	从某地址加载/贮存到某地址

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-8 3:38 PM
已删除: Mem

Shiqiang Xiao 13-11-8 3:37 PM
已删除: 状态对

--vcd-tracing *args*
激活信号追踪。追踪数据的输出为标准VCD文件格式。
如果*args*中含有任何空格，则必须附在标记中。其格式为：
global-options$_{opt}$ *h*-tile name *h*trace-options*i* *i*
全局选项为：

-pads 打开垫追踪。

-o *file* 在文件中输出。

追踪选项是与XN核心减速名称有关的展开图特有的，例如展开图[0]。
追踪选项为：

-ports 打开端口追踪。

-ports-detailed
打开更详细的端口追踪。

-cycles 打开时钟循环追踪。

-clock-blocks
打开时钟模块追踪。

-cores 打开逻辑核心追踪。

-instructions
打开指示追踪。

从不同的节点、展开图或逻辑核心输出追踪到不同的文件时，可以给此选项规定多个时间。
例如，以下命令配置模拟器以追踪展开图[0]的端口到文件trance.vcd。

```
xsim a.xe --vcd-tracing "-o trace.vcd -start-disabled -tile tile[0]
-ports"
```

VCD插入的追踪额可以使用_traceStart（）和_traceStop（）系统调用激活和失活。-start-disabled参数从开始失活vcd追踪，使用户只能激活/失活需要追踪的代码部分。例如：

```
#include <xs1.h>
#include <syscall.h>

port p1 = XS1_PORT_1A;

int main() {
   p1 <: 1;
   p1 <: 0;

   _traceStart();
```

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-8 3:39 PM
已删除: 将输出放入*file*中

Shiqiang Xiao 13-11-8 3:39 PM
已删除: 追踪

```
    p1 <: 1;
    p1 <: 0;
    _traceStop();

    p1 <: 1;
    p1 <: 0;

    return 0;
}
```

19.4 回路插件选项

XMOS Loopback plugin配置将要连接到一起的任何两个端口。插入的参数格式为：

-pin *package pin*

在包数据单上使用名称规定插脚。*Package*的数值必须与用于汇编程序的XN文件的包节点的ID属性匹配（参见§42.3）。

-port *name n offset*

规定与提名端口对应的*n*插脚。

*Name*的数值必须与用于汇编程序的XN文件的端口节点的Name属性匹配（参见§42.4.2）。

设置*offset*为非零数值以规定可用插脚的子集。

-port *tile p n offset*

规定连接到展开图上端口*p*的*n*个插脚。

展开图的数值必须与用于汇编程序的XN文件的展开图节点的参考属性匹配（参见§42.4.1）。

*P*可以为<xs1.h>定义的任何端口识别符。设置*offset*为非零数值，规定可用插脚的子集。

plugin选项成对规定，连接的两端各一个。例如，以下命令配置模拟器，以使插脚连接到展开图[0]的端口XS1_PORT_1A、到程序中端口UART_TX定义的插脚。

```
▶ xsim uart.xe --plugin LoopbackPort.dll '-port tile[0] XS1_PORT_1A 1 0 -port UART_TX
  1 0'
```

第I部分调试

内容

Shiqiang Xiao 13-11-8 3:41 PM
已删除: 通过打印输出进行实时调试

20 使用xTIMEcomposer调试程序

本章内容：

- 启动调试器
- 控制程序执行
- 检查暂停的项目
- 设置断点
- 查看反汇编代码

程序在硬件或仿真器上执行时，xCORE 调试器使你看到程序内部发生的变化，从而帮助你找到错误行为的原因。

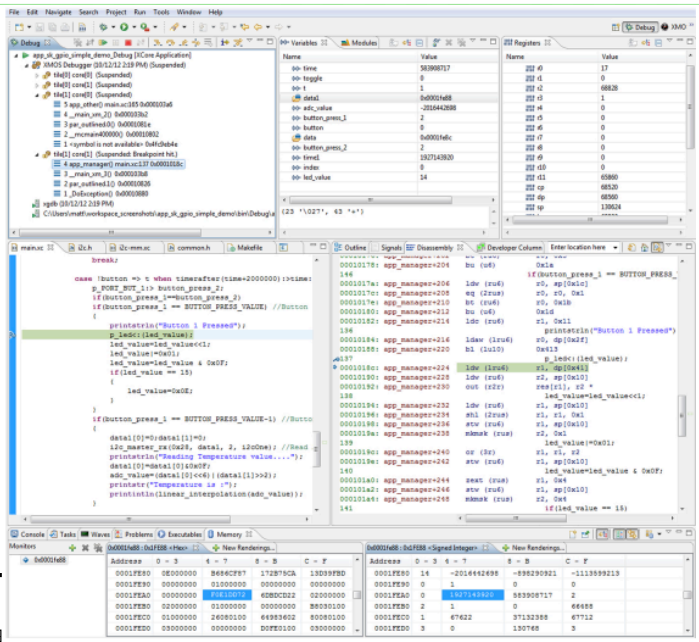


图 27.调试视图

Shiqiang Xiao 13-11-8 3:42 PM
已删除: 设定

Shiqiang Xiao 13-11-8 3:42 PM
已删除: 模拟

为使程序完全可见，你必须在启用调试的前提下对其进行编译（见§9.3）。这将为你的编译器添加可执行符号，从而使调试器与源代码直接相关。注意，启用优化后再进行编译会使调试变得更加困难。

20.1 启动调试器

在调试器控制下根据以下步骤加载程序：

1. 在“Project Explorer”中选择一个项目。
2. 点击“Run”——“ Debug Configurations”。
3. 在左侧面板中双击“XCore Application”。xTIMEcomposer 新建新的配置并于右侧面板中显示默认设置。
4. 在“Name “文本框中输入配置名称。
5. xTIMEcomposer 试图确定可执行的目标项目。自行选择时，点击“Project”文本框右侧的“Browse”按钮，在“Project Selection”对话框中选择你的项目，而后点击“Search Project”并在“Program Selection”对话框中选择可执行文件。

你必须无差错编译可执行程序，以备选择。

6. 将开发板与系统连接后，在“Device options “选项面板勾选“hardware”选项，然后从适配器里列表中选择“Adapter”。或者，勾选模拟器选项从而在模拟器上运行程序。
7. 点击“Debug”以保存配置并启动调试器。如果询问你是否打开“Debug “视图，则勾选“Remember my decision”并点击“YES”。

xTIMEcomposer 在调试器中加载程序，并于“Debug”视图中打开程序。

xTIMEcomposer 记住最近一次加载程序所用的配置。点击“调试”按钮，使用相同设置调试程序。点击“调试”按钮右边的箭头，从下拉列表中选择配置，使用不同配置。

20.2 控制程序执行

启动之后，调试器将会运行程序直到发生例外情况，或点击“Suspend”按钮暂停。

点击“复位”按钮继续执行暂停的程序，或使用步骤控制，逐步进入“Debug”视图中选择的核心。

Step into：于“调试”视图所选择的核心中执行单线源代码。如果某一代码的下一行是函数调用，则调试器将于调用函数的第一语句处暂停，从其他代码重新开始。

Shiqiang Xiao 13-11-8 3:43 PM
已删除: 项目浏览器

Shiqiang Xiao 13-11-8 3:43 PM
已删除: 运行

Shiqiang Xiao 13-11-8 3:43 PM
已删除: 调试配置

Shiqiang Xiao 13-11-8 3:43 PM
已删除: XCore 应用

Shiqiang Xiao 13-11-8 3:43 PM
已删除: 名称

Shiqiang Xiao 13-11-8 3:44 PM
已删除: 项目

Shiqiang Xiao 13-11-8 3:43 PM
已删除: 浏览

Shiqiang Xiao 13-11-8 3:44 PM
已删除: 项目选择

Shiqiang Xiao 13-11-8 3:44 PM
已删除: 搜索项目

Shiqiang Xiao 13-11-8 3:44 PM
已删除: 项目选择

Shiqiang Xiao 13-11-8 3:45 PM
已删除: 设备

Shiqiang Xiao 13-11-8 3:45 PM
已删除: 硬件

Shiqiang Xiao 13-11-8 3:45 PM
已删除: 调试适配器

Shiqiang Xiao 13-11-8 3:45 PM
已删除: 调试

Shiqiang Xiao 13-11-8 3:46 PM
已删除: 调试

Shiqiang Xiao 13-11-8 3:46 PM
已删除: 记住我的选择

Shiqiang Xiao 13-11-8 3:46 PM
已删除: “是”

Shiqiang Xiao 13-11-8 3:46 PM
已删除: “调试”

Shiqiang Xiao 13-11-8 3:46 PM
已删除: “暂停”

Shiqiang Xiao 13-11-8 3:46 PM
已删除: 调试

Shiqiang Xiao 13-11-8 3:46 PM
已删除: 进入

单步：在调试视图中，在所选的内核上执行单行源代码。
单步返回：在调试视图中，在所选内核上单步进行，直到当前函数返回。如果代码的下一行是一个函数调用，则调试程序执行整个函数。所有其它内核继续执行。
单步调试：将调试程序的语境切换到信道输出语句的对应输入内核。这对于跟踪数据路径非常有用，因为数据在内核之间流动。所有内核都不继续执行。
当调试已优化的代码时，单步运算不能保证进入源代码的下一行，因为编译器可能已经对指令重新排序，以改进性能。

20.3 检查暂停的程序

一旦程序暂停，你可以查询每一个内核的状态，并且可以检查寄存器和内存中保存的值。

检查内核的调用栈：调试视图显示了软件任务的列表，可以对每一个列表进行扩展，以显示其调用栈，如图 28 所示。

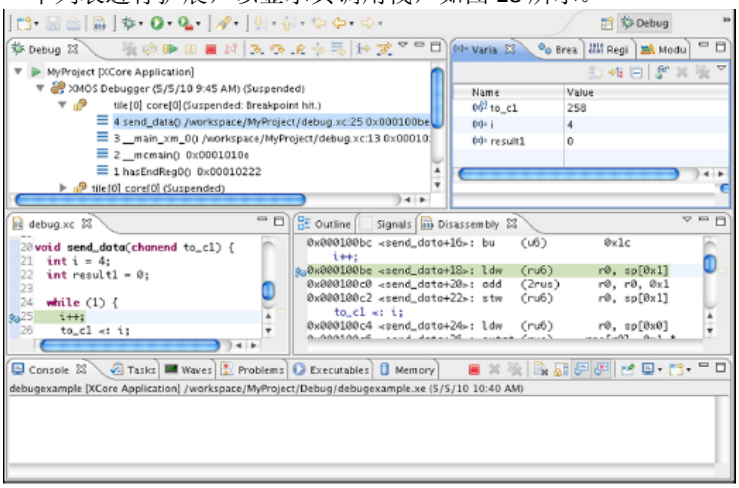


图 28：调试视图

在上例中，在 tile[0]在文件 debug.xc 的 25 行中函数 send_data 的断点暂停。

Shiqiang Xiao 13-11-8 3:50 PM
已删除：tile

检查变量：变量视图显示了变量和它们的值。在调试视图中，点击内核调用栈中的任何函数，以查看其变量，如图 20.3 所示。

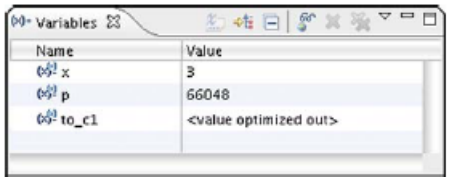


图 29：
变量视图

为了查看全局变量，右键单击变量视图，从弹出菜单中选择“增加全局变量”打开一个对话框，然后选择增加到视图中的全局变量。

不经优化编译程序保证每一个变量在其作用域的时间段内保存在内存中，从而总是能够显示它的值。如果启用了优化，则可能无法对变量进行检查，从而产生了消息<优化后的值输出>。

你可以对变量进行以下操作：

- 以十六进制格式显示变量的值：右键单击一个变量，以显示一个菜单，然后选择 格式▶十六进制。你也可以选择二进制，十进制或普通格式。普通格式由变量的类型确定。
- 改变变量的值：点击值使其高亮显示，输入一个新的值，然后按下回车。表项以黄色高亮显示，从而表明其值已经改变。这允许你按照“what-if”脚本测试发生了什么。
- 防止调试程序读出变量：右键单击变量，并从语境菜单中选择禁用。这在变量类型具有“易失性”时很有用。为了一次将设置应用到多个变量，在按下 Ctrl（Windows，Linux）或⌘（Mac）的同时点击多个变量，然后右键单击和从语境菜单中选择一个选项。
- 检查内存：内存视图提供了内存监视器列表，每一个类表代表一段内存。为了打开内存视图，选择 窗口▶显示视图▶内存。在调试视图中，点击任意内核查看内存的内容，如图 20.3 中所示。

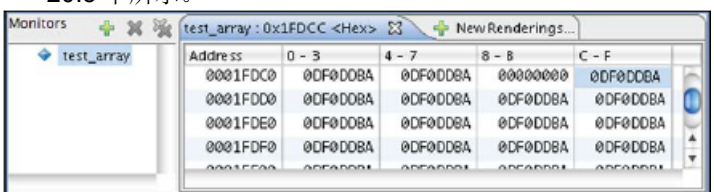


图 30：
内存视图

为了指定待查看的内存位置，点击增加按钮打开内存监视器对话框，输入一个内存位置，然后点击确定。你可以输入一个绝对值或者一个 C/XC 表达式。如果要查看数组内容，只需要输入数组名称。
为了以不同格式显示内存内容（例如 Hex 或 ASCII），点击“新的释义”标签，选择一种格式，然后点击“增加释义”。
xTIMEcomposer 在内存视图右侧的面板中增加了新标签，每一个标签显示了内存中值的不同解释。

20.4 设置断点

断点时程序中的一个标记，它命令调试程序中断执行，从而使你能够调查程序的状态。你可以对任何可执行的代码行增加一个断点，从而使执行在该代码行执行之前暂停。

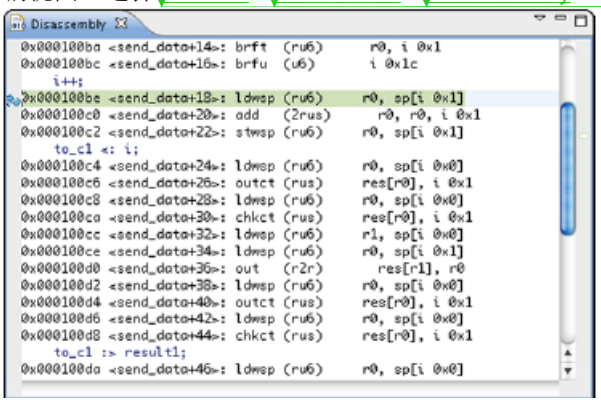
为了增加一个断点，在你希望暂停执行的代码行旁边的代码编辑器左边中的标记栏双击，以暂停执行。一个蓝点将会出现，代表存在断点。请注意，断点应用到执行函数的每一个内核。

断点也显示在断点视图中。为了打开断点视图，选择 窗口▶显示▶视图▶断点。在一个断点上双击确定对应代码行在源代码编辑器中的位置。
以下是你可以对断点进行的其它操作：
设置一个条件断点：右键单击断点标记，以显示一个语境菜单，然后选择“断点性质”显示性质对话框。在左侧面板中双击“通用”选项，并在右侧面板的“条件”文本框中输入 C/XC 条件表达式。表达式可以含有断点作用域内的任何变量。
设置一个条件断点：右键单击断点标记，以显示一个语境菜单，然后选择“断点性质”显示性质对话框。在左侧面板中双击“通用”选项，并在右侧面板的“条件”文本框中输入 C/XC 条件表达式。表达式可以含有断点作用域内的任何变量。
对全局变量设置一个监视点：监视点是一个特殊的断点，它在表达式的值改变时暂停执行（没有指定何时发生）。在断点视图的任意位置右键单击，然后从语境菜单中选择“增加监视点 C/XC”。在对话框中输入一个 C/XC 表达式，例如，[MAX]。选择“写入”以在表达式被写入时中断，选择“读出”以在表达式被读出时中断。
显示一个断点：在断点视图中，取消断点旁边的选择框。勾选该选择框，以重新启用断点。
移除一个断点：在代码编辑器中双击一个断点标记将其移除。或者，在断点视图中右键单击一个断点，然后从语境菜单中选择移除；为了移除所有断点，选择全部移除。

20.5 查看反汇编代码

反汇编视图显示了在目标平台上执行的汇编指令。为了打开反汇编视图，选择Window▶Show View▶Disassembly。

图 31：反汇编视图



当焦点位于反汇编视图时，xTIMEcomposer 自动启用指令步进模式。或者点击“Instruction Stepping Mode”按钮启用。一旦启用，点击“Step”按钮按每一条汇编指令运行程序。

Shiqiang Xiao 13-11-8 3:52 PM
已删除: 窗口

Shiqiang Xiao 13-11-8 3:52 PM
已删除: 显示视图

Shiqiang Xiao 13-11-8 3:53 PM
已删除: 反汇编

Shiqiang Xiao 13-11-8 4:00 PM
已删除: 指令步进模式

Shiqiang Xiao 13-11-8 4:02 PM
已删除: “步进”

21 通过 printf 实时调试

在本章中

- 将 stdout 和 stderr 重定向到 XTAG-2
- 在启用 XTAG-2 输出的情况下运行程序
- 使用 UART 接口输出

xCORE 调试程序允许你暂停程序的执行，以便分析其内部状态。但是，如果你的程序含有时序-关键行为，例如由于其实现一个实时通信协议，暂停程序的行为可能引起其它系统组件出现故障，从而妨碍了进一步调试。

调试的一个备选方法是对你的程序增加追踪语句，以便在实时环境下观察其内部行为（有时候被称为 printf 调试）。通过打印中间计算的结果，你可以快速的隔离你的程序中出错的地方。

在传统的调试环境中，使用诸如 JTAC 标准输出数据造成妨碍内核执行的中断，从而使你的程序运行速度显著降低。xTIMEcomposer 允许你将标准流 stdout 和 stderr 重定向到 XTAG-2 调试适配器，数据缓存在调试适配器中，直到可以将其输出到主机为止。

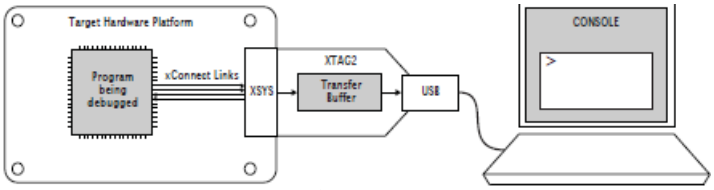


图 32：XTAG-2 具有 I/O 重定向的调试配置

在此配置中，对输出例行程序的调用（例如 printf）在数据被输出到 xConnect 链接时完成，从而将对程序时序特性的影响降到最低。这允许将调试语句增加到多个时序-关键代码段，并在执行期间在控制台查看。在程序崩溃的情况下，XTAG-2 缓冲区中所有剩下的内容被发送到 PC，从而确保重要信息不会丢失。

如果你使用了 legacy FTDI 或 XTAG-1 调试适配器，或者如果你的目标硬件上的 XSYS 连接器没有提供 xConnect 链接，则你可以通过你的适配器的 UART 接口（详见 21.3）输出数据。请注意，UART 接口严重影响性能。

Shiqiang Xiao 13-11-8 4:29 PM
已删除: 请

21.1 将 stdout 和 stderr 重定向到 XTAG-2

以下程序将标准输出重定向到 XTAG-2。

```
#include <stdio.h>
#include <xscope.h>

port receive;
port transmit;

int process(int);

void xscope_user_init(void) {
  xscope_register(0);
  xscope_config_io(XSCOPE_IO_BASIC);
}

int main() {

  while (1) {
    int dataIn, dataOut;

    receive :> dataIn;
    dataOut = process(dataIn);

    /* Debug Information */
    if (dataOut < 0)
      printf("%d %d", dataIn, dataOut);

    transmit <: dataOut;
  }
}
```

在构造符 xscope_user_init 中，对 xscope_register 的调用初始化了 XTAG-2 接口，然后对 xscope_config_io 的调用将流 stdout 和 stderr 重定向到此接口。

主程序从一个端口输出数据，在该端口执行计算，并将结果输出到另一个端口。它使用了标准输出函数 printf 记录计算结果小于零的实例。

你可以同时在任何内核上使用 C 标准 I/O 函数。这一使用引起单通道末端被分配到输出数据的每一个 tile 上。

你可以通过调用 xscope_config_io（选项 XSCOPE_IO_TIMED）对输出数据加时间戳。这使得输出时间戳随数据显示在控制台中。请注意，这也减少了任何时间可以缓冲的数据量。

21.2 在 XTAG-2 输出启用的情况下运行程序

为了将标注输出重定向到 XTAG-2 并将其显示在控制台中，你必须通过 Xscope 仪表库编译和运行你的程序。为了编译和运行你的程序，请遵守以下步骤：

1. 打开你的项目所用的 Makefile。
2. 定位你的编译配置所用的 XCC_MAP_FLAGS_config 变量，例如，XCC_MAP_FLAGS_Release。
3. 增加选项-fxscope。
4. 如果使用 XDE 开发，对你的目标设备创建一个运行配置（请见 13.1）。在 Xscope 标签中，选择“Offine mode”模式。点击”RUN”保存和运行此配置。

   XDE 加载你的程序，同时在控制台中显示从 XTAG-2 收到的数据。
5. 如果你使用命令行工具开发，跳过选项—xscope 到 XRUN，例如：

   ► xrun --xscope myprog.xe

   XRUN 加载你的程序，并保持到 XTAG-2 适配器的连接，同时在终端显示从其收到的数据。当程序执行对 exit 的调用时，XRUN 终止。

Shiqiang Xiao 13-11-8 4:29 PM
已删除：“离线”

Shiqiang Xiao 13-11-8 4:30 PM
已删除：运行

Shiqiang Xiao 13-11-7 12:58 PM
已删除：--

21.3 使用 UART 接口输出

如果你使用了 legacy FTDI 或 XTAG-1 调试适配器，或者如果你的目标硬件上的 XSYS 连接器没有提供 xConnect 链接，则你可以通过你的适配器提供的 UART 接口输出数据。

为了使用 UART 接口，你必须提供具有 1 位 UART TX 端口的 Xscope 库，通过将该端口引脚与你的调试适配器上的 UART-TX 引脚连接，对该端口初始化。下面给出了一个初始化的例子。

```
#include <platform.h>
#include <xscope.h>

port uart_tx = PORT_UART_TX;

void xscope_user_init(void) {
  xscope_register(0);
  xscope_config_uart(uart_tx);
  xscope_config_io(XSCOPE_IO_BASIC);
}
```

为了在 XDE 中运行你的程序，对你的目标设备创建一个运行配置（请见 13.1），并选择选项“运行 UART 服务器”。

为了使用命令行工具运行你的程序，跳过选项--uart 到 XRUN，例如：

► xrun --uart --xscope myprog.xe

因为 UART 接口使用了一个端口取代 xConnect 链接，你只能在单个 tile 上使用 C 标准 I/O 函数。

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

J 部分
闪存编程

目录

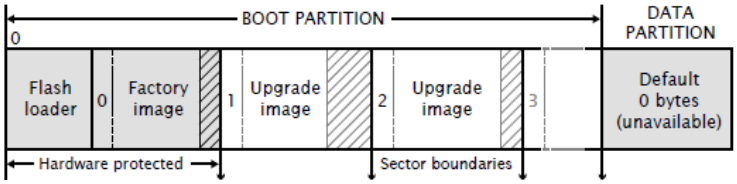
22 具有闪存的设计和制造系统

在本章中

- 将 boot 设置在 flash memory
- 生成制造 flash 镜像
- 现场升级
- 配置 flash 加载器

xTIMEcomposer 可用于找到使用 SPI 闪存进行引导和长期存储的 xCORE 设备。图 33 中给出了 xCORE 闪存格式。

图 33：闪存格式图



从逻辑上将闪存分为引导和数据分区。引导分区由闪存加载器和其后的“出厂镜像”和零个或多个可选的“升级镜像”组成。每一个镜像以一个描述符开始，描述符含有一个唯一的版本号，含有程序所用的每一个 tile 的代码/数据分段列表和 CRC。在默认情况下，闪存加载器引导具有有效 CRC 的最高版本镜像。

22.1 从闪存引导程序

为了将一个程序加载到你的开发板上的 SPI 闪存装置上，启动命令行工具（请见 3.2），并输入以下命令：

1. xflash –l
   XFLASH 按以下格式打印连接到你的计算机和每一个 JTAG 链上上设备的所有 JTAG 适配器：

ID	名称	适配器 ID	设备
--	----	----------	-----

2. xflash --id ID program.xe
   XFLASH 从你的已编译程序以 xCORE 闪存格式（含有第一个阶段加载器和由二进制和数据段构成的出厂镜像）生成镜像。然后它将此镜像写入使用 xCORE 设备的闪存。

   用于编译你的程序的 XN 文件必须定义 SPI 闪存设备，并指定与其相连的 Xcore 设备的四个端口（请见 XM-000929-PC）。

Shiqiang Xiao 13-11-8 4:33 PM
已删除: 从闪存引导一个程序

Shiqiang Xiao 13-11-8 4:33 PM
已删除: 所用的

Shiqiang Xiao 13-11-8 4:33 PM
已删除: 执行

Shiqiang Xiao 13-11-8 4:37 PM
已设置格式: 缩进: 左: 0 cm

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

22.2 生产一个用于制造的闪存镜像

在制造环境中，通常将相同的程序编入多个闪存设备。

为了以 Xcore 闪存格式生成镜像文件（随后可以将这种格式编程到闪存设备中），启动命令行工具（见 3.2），然后输入以下命令：

► xflash program.xe -o image-file

XFLASH 生成由第一阶段加载器和你的程序（作为出厂镜像）构成的镜像，它将该镜像写入指定的文件。

22.3 执行一个现场升级

xTIMEcomposer 和 libflash 库允许你在产品的生命周期管理多个固件升级。你可以使用 XFLASH 创建一个升级镜像（从你的程序），使用 libflash 将此镜像写入引导分区。通过使用 libflash，升级能够避免部分完整写入，例如，由于电源故障：如果引导期间升级镜像的 CRC 失败，则将加载之前的镜像。

22.3.1 写一个自我更新的程序

图 34 中的程序例子使用了 libflash 库自我更新。

对 fl_connect 的调用打开了 xCORE 和 SPI 设备之间的连接，对 fl_getPageSize 的调用确定了 SPI 设备的页面大小。所有读取和写入操作发生在页面层次。

通过调用 fl_getFactoryImage 和 getNext- BootImage 定位第一个更新页面。在定位之后，fl_startImageReplace 准备此镜像，以便由规定（最大）大小的新镜像代替。必须调用 fl_startImageReplace，直到其返回零，返回零表示准备完成。

函数 fl_writeImagePage 将数据的下一页写入 SPI 设备。对此函数的调用在数据被输出到设备之后返回，但是可能在设备已经将数据写入其闪存之前返回。这增加了处理器抓取数据下一页的时间量。函数 fl_endWriteImage 等待 SPI 设备将数据的最后一页写入其闪存。为了简化写入操作，XFLASH 对升级镜像增加了填充内容，以确保其大小是页面大小的倍数。

图 34：使用 libflash 自我升级的 C 程序

```
#include <platform.h>
#include <flash.h>

#define MAX_PSIZE 256

/* initializers defined in XN file
 * and available via platform.h */

fl_SPIPorts SPI = { PORT_SPI_MISO,
                    PORT_SPI_SS,
                    PORT_SPI_CLK,
                    PORT_SPI_MOSI,
                    XS1_CLKBLK_1 };

int upgrade(chanend c, int usize) {

  /* obtain an upgrade image and write
   * it to flash memory
   * error checking omitted */

  fl_BootImageInfo b;
  int page[MAX_PSIZE];
  int psize;

  fl_connect(SPI);

  psize = fl_getPageSize();
  fl_getFactoryImage(b);
  fl_getNextBootImage(b);

  while(fl_startImageReplace(b, usize))
    ;
  for (int i=0; i page[j];)
    fl_writeImagePage(page);

  fl_endWriteImage();

  fl_disconnect();

      return 0;
}

int main() {
  /* main application - calls upgrade
   * to perform an in-field upgrade */
}
```

调用 fl_disconnect 关闭了 xCORE 和 SPI 设备之间的连接。

22.3.2 编译和部署升级器

为了编译和部署你的程序的第一个版本，启动命令行工具（请见3.2），然后输入以下命令：

1. xcc file.xc -target=boardname -lflash -o first-release.xe
   XCC 编译你的程序，并将其连接到 libflash。或者对你的 Makefile 增加选项-lflash。
2. xflash first-release.xe -o manufacture-image
   XFLASH 以 Xcore 闪存格式生成一个镜像，镜像含有第一阶段加载器和你的程序的第一个版本，以作为出厂镜像。

为了编译和部署你的程序的升级版本，输入以下命令：

1. xcc file.xc -target=boardname -lflash -o latest-release.xe
   XCC 编译你的程序，并将其连接到 libflash。
2. xflash --upgrade version latest-release.xe -o upgrade-image
   XFLASH 生成具有指定版本号（必须大于零）的升级镜像。你的程序必须获得此镜像，以自我更新。

如果更新操作成功，则在重置设备时加载器引导更新镜像，否则其将引导出厂镜像。

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

22.4 优化闪存加载器

xTIMEcomposer 允许你配置选择从闪存加载哪一个镜像的机制。图 35 中的程序例子基于数据分区开始时的值确定加载哪一个镜像。

xCORE 加载器首先调用函数 init，然后反复声明引导分区中的每一个镜像。对于每一个镜像，其以镜像版本调用 checkCandidateImageVersion，然后如果函数返回值不等于零，则其 CRC 验证通过，其以镜像版本号和地址调用 recordCandidateImage。最后，加载器调用 reportSelectedImage，以获得所选镜像的地址。

为了生成一个定制加载器，你需要定义函数 init，checkCandidateImageVersion，recordCandidateImage 和 reportSelectedImage。

加载器提供了 readFlashDataPage 函数。

Shiqiang Xiao 13-11-8 4:39 PM
已删除: 函数

图 35：定制闪存加载器的 C 函数

```
extern void *readFlashDataPage(unsigned addr);

int   dpVersion;
void *imgAdr;

int init(void) {
  void *ptr = readFlashDataPage(0);
  dpVersion = *(int *)ptr;
}

int checkCandidateImageVersion(int v) {
  return v == dpVersion;
}

void recordCandidateImage(int v, unsigned adr) {
  imgAdr = adr;
  return 1;
}

unsigned reportSelectedImage(void) {
  return imgAdr;
}
```

22.4.1 编译加载器

为了创建含有定制闪存加载器和出厂镜像的闪存镜像，启动命令行工具（请见 3.2），然后输入以下命令：

1. xcc -c file.xc -o loader.o
   XCC 编译你用于镜像选择的函数，生成一个二进制对象。
2. xflash bin.xe --loader loader.o
   XFLASH 将含有定制加载器和出厂镜像的闪存镜像写入规定的文件。

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

22.4.2 增加额外的镜像

以下命令编译了含有定制加载器、出厂镜像和两个额外镜像的闪存镜像：

► xflash factory.xe --loader loader.o --upgrade 1 usb.xe 0x20000 --upgrade 2 avb.xe

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

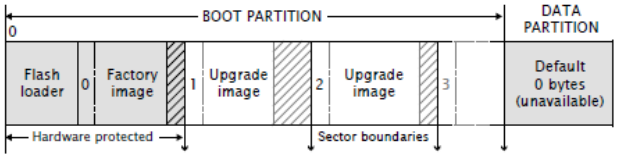
到—upgrade 的参数包括版本号、可执行文件和可选大小（单位为字节）。XFLASH 将每一个升级镜像写入下一个扇区边界。大小参数用于对镜像加入填充，从而能够在将来通过更大的镜像对其现场更新。

23 libflash API

在本章中

Libflash 库提供了对使用下图所示的 XCORE 格式的 SPI 闪存设备读写数据的函数。

图 36：闪存格式图



所有函数的原型都在头文件<flash.h>中。除非另有规定，函数运行成功返回 0，运行失败返回 0 以外的任何数。

23.1 一般操作

Shiqiang Xiao 13-11-8 4:40 PM
已删除：。。

在试图使用 SPI 设备之前，程序必须明确的打开到 SPI 设备的连接，并且必须在完成对设备的访问之后断开。

函数 fl_connect 和 fl_connectToDevice 需要一个类型 fl_SPIPorts 的参数，该参数定义了用于连接到该设备的四个端口和时钟块。

```
typedef struct {
  in buffered port:8 spiMISO;
  out port spiSS;
  out port spiCLK;
  out buffered port:8 spiMOSI;
  clock spiClkblk;
} fl_SPIPorts;
```

int fl_connect(fl_SPIPorts *SPI)

fl_connect 打开到指定 SPI 设备的连接。

int fl_connectToDevice(fl_SPIPorts *SPI, fl_DeviceSpec spec[], unsigned n)

fl_connectToDevice 打开到 SPI 设备的连接。它通过 n 个 SPI 设备规约的数组枚举，尝试使用每一个规约连接，直到它成功为止。

int fl_getFlashType(void)

fl_getFlashType 对闪存设备返回 enum 值。下面给出了 libflash 已知的设备枚举。

```
typedef enum {
  UNKNOWN = 0,
  ALTERA_EPCS1,
  ATMEL_AT25DF041A,
  ATMEL_AT25FS010,
  ST_M25PE10,
  ST_M25PE20,
  WINBOND_W25X40
} fl_FlashId;
```

如果函数调用 fl_connectToDevice(p, spec, n)用于连接到闪存设备，fl_getFlashType 返回参数值 spec[i].flashId，其中 i 是所连接设备的索引。

unsigned fl_getFlashSize(void)

fl_getFlashSize 返回 SPI 设备的容量（单位字节）

int fl_disconnect(void)

fl_disconnect 关闭到 SPI 设备的连接。

23.2 引导分区函数

默认情况下，引导分区的大小用于设置闪存设备的大小。通过一个枚举器接口，提供了对引导镜像的访问。

int fl_getFactoryImage(fl_BootImageInfo *bootImageInfo)

fl_getFactoryImage 提供了关于出厂引导镜像的信息。

int fl_getNextBootImage(fl_BootImageInfo *bootImageInfo)

fl_getNextBootImage 提供了关于下一个引导镜像的信息。在定位之后，就可以对镜像升级。另外还提供了读取升级镜像的函数。

unsigned fl_getImageVersion(fl_BootImageInfo *bootImageInfo)

fl_getImageVersion 返回指定镜像的版本号。

int fl_startImageReplace(fl_BootImageInfo *, unsigned maxsize)

fl_startImageReplace 准备 SPI 设备用于替换镜像。在此调用之后，无法分配旧的镜像。

试图写入数据分区或另一个升级镜像的空间是无效的。非零返回值说明准备还没有完成，并且应当再次调用此函数。此行为允许由程序屏蔽的扇区擦除的延迟。

int fl_startImageAdd(fl_BootImageInfo*, unsigned maxsize, unsigned padding)

fl_startImageAdd 准备 SPI 设备用于在指定镜像之后增加一个镜像。在之前的镜像之后，新镜像的开始至少是填充字节。

试图写入数据分区或另一个升级镜像的空间是无效的。非零返回值说明准备还没有完成，并且应当再次调用此函数。此行为允许由程序屏蔽的扇区擦除的延迟。

int fl_startImageAddAt(unsigned offset, unsigned maxsize)

fl_startImageAddAt 准备 SPI 设备用于从出厂地址之后第一个扇区的基址开始，以指定的地址偏移量增加一个镜像。

试图写入数据分区或另一个升级镜像的空间是无效的。非零返回值说明准备还没有完成，并且应当再次调用此函数。

int fl_writeImagePage(const unsigned char page[])

fl_writeImagePage 等待，直到 SPI 设备能够接受一个请求为止，然后将数据的下一页输出到设备。试图超过传递到 fl_startImageReplace, fl_startImageAdd 或 fl_startImageAddAt 的最大大小写入是无效的。

int fl_writeImageEnd(void)

fl_writeImageEnd 等待，直到 SPI 设备将数据的最后一页写入其内存。

int fl_startImageRead(fl_BootImageInfo *b)

fl_startImageRead 准备 SPI 设备用于读取指定的升级镜像的内容。

int fl_readImagePage(unsigned char page[])

fl_readImagePage 从 SPI 设备输入数据的下一页，并将其写入数组页。

int fl_deleteImage(fl_BootImageInfo* b)

fl_deleteImage 通过指定的镜像擦除升级镜像。

23.3 数据分区函数

假设所有闪存设备具有统一的页面大小，但是假设没有统一的扇区大小。读写操作发生在页面层次，并且擦除操作发生在扇区层次。这表示为了写入扇区的一部分，至少有一个扇区的缓存大小用于储存其它数据。

在以下函数中，写入数据分区和从数据分区擦除并不是故障安全的。如果操作被中断，例如由于电源故障，页面或分区中的数据是不确定的。

unsigned fl_getDataPartitionSize(void)

fl_getDataPartitionSize 返回数据分区的大小（单位为字节）

int fl_readData(unsigned offset, unsigned size, unsigned char dst[])

fl_readData 从一个偏移量将字节数读入数据分区，并将它们写入数组 dst。

unsigned fl_getWriteScratchSize(unsigned offset, unsigned size)

fl_getWriteScratchSize 返回 fl_writeData 对给定参数所需的缓存大小

```
int fl_writeData(unsigned offset,
                 unsigned size,
                 const unsigned char src[],
                 unsigned char buffer[])
```

fl_writeData 将数组 src 写入数据分区中的指定偏移量。它使用数组缓冲区保存必须重写的页面数据。

23.3.1 页面级别的函数

unsigned fl_getPageSize(void)

fl_getPageSize 返回页面大小（单位为字节）

unsigned fl_getNumDataPages(void)

fl_getNumDataPages 返回数据分区中页面数量

unsigned fl_writeDataPage(unsigned n, const unsigned char data[])

fl_writeDataPage 将数组数据写入数据分区中的第 n 个页面。数据数组必须至少与页面大小一样大；如果更大，则最高的元素被忽略。

unsigned fl_readDataPage(unsigned n, unsigned char data[])

fl_readDataPage 读出数据分区中的第 n 个页面，并将其写入数组数据。数据的大小必须至少与页面大小一样大。

23.3.2 扇区级别函数

unsigned fl_getNumDataSectors(void)

fl_getNumDataSectors 返回数据分区中的扇区数量。

unsigned fl_getDataSectorSize(unsigned n)

fl_getDataSectorSize 返回数据分区中第 n 个扇区的大小（单位为字节）

unsigned fl_eraseDataSector(unsigned n)

fl_eraseDataSector 擦除数据分区中的第 n 个扇区

unsigned fl_eraseAllDataSectors(void)

fl_eraseAllDataSectors 擦除数据分区中的所有扇区。

24 libflash 原生支持的设备列表

Libflash 支持市场上的大量闪存设备。使用 SPI 规约文件描述每一种闪存设备。图 37 中的表列出了通过 xTIMEcomposer 包括的 SPI 规约文件的闪存设备列表。

图 37 libflash 原生支持的闪存设备列表

制造商	部件编号	通过默认值在 libflash 中启用
Altera	EPCS1	Y
AMIC	A25L016	N
	A25L40P	N
	A25L40PT	N
	A25L40PUM	N
	A25L80P	N
Atmel	AT25DF021	N
	AT25DF041A	Y
	AT25F512	N
	AT25FS010	Y
EMT	F25L004A	N
Macronix	MX25L1005C	N
NUMONYX	M25P10	N
	M25P16	N
	M25P40	N
	M45P10E	N
SST	SST25VF010	N
	SST25VF016	N
	SST25VF040	N
ST Microelectronics	M25PE10	Y
	M25PE20	Y
Winbond	W25X10	N
	W25X20	N
	W25X40	Y

25 增加对新闪存设备的支持

在本章中

为了支持新的闪存设备，必须写入描述设备特征的配置文件，例如页面大小，页面数量和读取、写入和擦除数据命令。这些信息可以在闪存设备的数据单中找到。市场上的许多设备都可以使用这些配置参数描述；这些还没有得到支持。

下文给出了 Numonyx M25P10-A2 的配置文件。该设备被描述为 C 结构的初始化程序，C 结构的值在以下章节中描述。

```
10,                     /* 1.  libflash device ID */
256,                    /* 2.  Page size */
512,                    /* 3.  Number of pages */
3,                      /* 4.  Address size */
4,                      /* 5.  Clock divider */
0x9f,                   /* 6.  RDID cmd */
0,                      /* 7.  RDID dummy bytes */
3,                      /* 8.  RDID data size in bytes */
0x202011,               /* 9.  RDID manufacturer ID */
0xD8,                   /* 10. SE cmd */
0,                      /* 11. SE full sector erase */
0x06,                   /* 12. WREN cmd */
0x04,                   /* 13. WRDI cmd */
PROT_TYPE_SR,           /* 14. Protection type */
{{0x0c,0x0},{0,0}},     /* 15. SR protect and unprotect cmds */
0x02,                   /* 16. PP cmd */
0x0b,                   /* 17. READ cmd */
1,                      /* 18. READ dummy bytes*/
SECTOR_LAYOUT_REGULAR,  /* 19. Sector layout */
{32768,{0,{0}}},        /* 20. Sector sizes */
0x05,                   /* 21. RDSR cmd*/
0x01,                   /* 22. WRSR cmd */
0x01,                   /* 23. WIP bit mask */
```

25.1 Libflash 设备 ID

```
10, /* 1. libflash device ID */
```

Libflash 在调用函数 fl_getFlashType 时需要该值，以使应用程序能够识别所连接的闪存设备。

25.2 页面大小和页面数量

```
10,                     /* 1.  libflash device ID */
256,                    /* 2.  Page size */
```

这些值规定了每一个页面的大小（单位为字节）和所有可用扇区的全部页面数量。在 M25P10-A 数据单中，可以从第 6 页的以下段落找到这些内容：

内存被组织为 4 个扇区，每一个含有 128 个页面。每一个页面的宽度为 256 字节。因此，可以将整个页面看作由 512 个页面或者 131072 个字节构成。

Shiqiang Xiao 13-11-8 4:42 PM
已删除：

25.3 地址大小

```
3,                          /* 4.  Address size */
```

这些值规定了用于表示地址的字节数。图 38 复制了提供此信息的 M25P10-A 数据单的一部分。在该表中，所有指令要求地址占用三个字节。

图 38：M25P10-A 数据单 17 页上的表 4

指令	描述	单字节指令代码		地址字节	空字节	数据字节
WREN	启用写入	0000 0110	06 h	0	0	0
WRDI	禁用写入	0000 0100	04 h	0	0	0
RDID	读出	1001 1111	9fh	0	0	1 到 3
RDSR	读出状态寄存器	0000 0101	05 h	0	0	1 到 ∞
WRSR	写入状态寄存器	0000 0001	01 h	0	0	1
READ	读出数据字节	0000 0011	03 h	3	0	1 到 ∞
FAST_READ	以更高的速度读出数据字节	0000 1011	08 h	3	1	1 到 ∞
PP	页面程序	0000 0010	02 h	3	0	1 到 256
SE	扇区擦除	1101 1000	D8 h	3	0	0
BE	批量擦除	1100 0111	C7 h	0	0	0
DP	深度低耗电	1011 1001	B9 h	0	0	0
RES	从深度低耗电释放，并读出电子签名	1010 1011	ab h	0	3	1 到 ∞
	从深度低耗电释放			0	0	0

25.4 时钟频率

```
4,                          /* 5.  Clock divider */
```

该值用于确定与 SPI 设备交互的时钟频率。对于 n 的值，所用的 SPI 时钟频率为 100/2*n MHz。libflash 支持最大 12.5MHz 的频率。

图 39 复制了提供此信息的 M25P10-A 数据单的一部分。AC 特性表给出了配置文件中所用的所有指令，正如在本文中所讨论的，其最高可以以 25MHz 运行。此频率高于 libflash 可以支持的频率，所以给出的值为 4，以产生 12.5MHz 的时钟。

总的来说，如果 SPI 设备对 libflash 所用的不同命令支持不同的时钟频率，则必须规定最低的值。

图 39：M25P10-A 数据单 40 页上的表 18（仅有前四个表项）

符号	备选	参数	最小	类型	最大	单位
		以下指令的时钟频率：FAST_READ, PP, SE, BE,DP, RES, WREN, WRDI, RDSR, WRSR				
		READ 指令的时钟频率				
		时钟高时间				
		时钟低时间				

25.5 读取设备 ID

```
0x9f,                   /* 6.  RDID cmd */
0,                      /* 7.  RDID dummy bytes */
3,                      /* 8.  RDID data size in bytes */
0x202011,               /* 9.  RDID manufacturer ID */
```

大部分的闪存设备具有硬件标识符，可用于标识该设备。当应用程序支持一个或多个闪存设备时标识符由 libflash 使用，以确定所连接的设备类型。读取设备 ID 的顺序通常是发出一个 RDID（读取 ID）命令，等待零个或多个空字节，然后读出一个或多个数据字节。

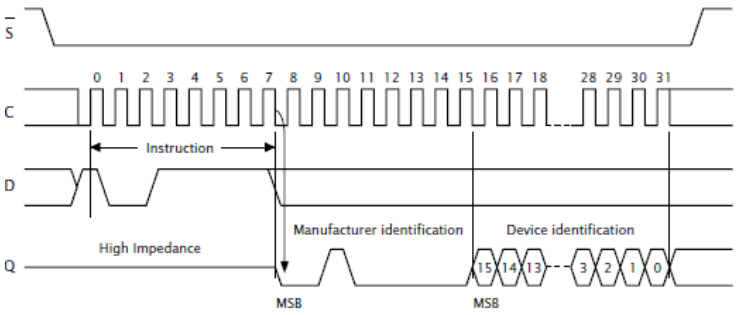
图 38 复制了提供此信息的 M25P10-A 数据单的一部分。指令 RDID 的列表示命令值为 0x9f，也即没有空字节，并且有一到三个空字节。如图 40 和 41 所示，读出的数据量依赖于是否只需要制造商 ID（第一字节），或者是否需要制造商 ID 和设备 ID（第二字节）。为了唯一的标识该设备，需要所有三个字节，所以制造商 ID 被规定为三字节的值 0x202011。

图 40：M25P10-A 数据单 19 页上的表 5

制造商标识	设备标识	
	内存类型	内存容量
20h	20h	11h

一般来说，如果有 RDID 命令的选择，则应当首选 JEDEC 兼容的命令。否则，应当使用返回最长 ID 的命令。

图 41：M25P10-A 数据单 19 页上的图 9



25.6 扇区擦除

```
0xD8,                   /* 10. SE cmd */
0,                      /* 11. SE full sector erase */
```

大部分闪存设备提供了擦除所有扇区或部分扇区的命令。

图 38 复制了提供了此信息的 M25P10-A 数据单的一部分。指令 SE 列显示命令值为 0xd8。在 M25P10-A 数据单上，被擦除的数据量可以从 28 页的第一段上找到：
扇区擦除（SE）指令对所选扇区内的所有位设置为 11’ (FFh)。

在此例中，SE 命令擦除了所有扇区，所以 SE 数据值被设置为 0。如果被擦除的位数小于整个扇区，则应当将该值设置为被擦除的字节数。

25.7 写入启用/禁用

```
0x06,                   /* 12. WREN cmd */
0x04,                   /* 13. WRDI cmd */
```

大部分的闪存设备提供指令启用和禁用对内存的写入。图 38 复制了提供了此信息的 M25P10-A 数据单的一部分。指令 WREN 行显示该命令值为 0x06，并且指令 WRDI 行显示该命令值为 0x04。

25.8 内存保护

```
PROT_TYPE_SR,           /* 14. Protection type */
{{0x0c,0x0},{0,0}},     /* 15. SR protect and unprotect cmds */
```

当写入被禁用时，一些闪存设备提供了额外的扇区保护。对于支持此能力的设备，libflash 尝试保护闪存镜像，防止被应用程序意外影响。保护类型的支持值为：

PROT_TYPE_NONE
设备没有提供保护
PROT_TYPE_SR
设备通过写入状态寄存器提供保护
PROT_TYPE_SECS
设备提供命令保护单独扇区

保护细节被规定为以下命令格式的一部分：
{{a,b},{c,d}}
设备没有提供保护，所有值都应当设置为 0。如果设备提供 SR 保护，应当将 a 和 b 设置为写入 SR 的值，以启用和禁用设备保护，并将 c 和 d 设置为 0。否则，应当将 c 和 d 设置为写入到命令的值，以启用和禁用设备保护，并将 a 和 b 设置为 0。

图 42 和图 43 复制了提供了此信息的 M25P10-A 数据单的一部分。第一个表显示状态寄存器应当被设置为 1，以保护所有扇区，并且全部为 0 禁用保护。第二个表显示这些是 SR 的 2 和 3 位。

图 42 M25P10-A 数据单 13 页上的表 2

状态寄存器		内存内容	
BP1	BP0	保护区	无保护区
0	0	无	所有扇区（四个扇区：0，1，2 和 3）
0	1	前四分之一（扇区 3）	后四分之三（三个扇区：0 到 2）
1	0	前一半（两个扇区：2 和 3）	后一半（扇区 0 和 1）
1	1	所有扇区（四个扇区：0，1，2 和 3）	无

图 43 M25P10-A 数据单 20 页上的表 6

状态寄存器写保护

b7							b0
SRWD	0	0	0	BP1	BP0	WEL	WIP

块保护位

启用写入开关位

写入进度位

25.9 编程命令

```
0x02,                    /* 16. PP cmd */
```

图 44：F25L004A 数据单第 12 页的表 7

每次编程时，将设备编程为页面或数个子节。如果有页面编程则应使用，因为它总结了通过SPI接口传输的数据的数量。

图38为提供此信息的M25P10-A数据单的一部分。表中提供了一个页面程序命令，且其数值为0x02。

如果不支持页面编程，则此数值为三个单独的数值的级联。比特0..7必须设置为0。比特8..15应含有程序命令。比特16..23应含有每个命令的子节数。Libflash库要求第一个程序命令接受三字节地址，但是后续的程序命令使用自动地址增量（AAI）。

无 PP 命令的设备实例为 ESMT F25L004A[3]。图 44 为提供此信息的数据单 F25L004A 的一部分。在计时图中，该 AAI 命令的数值为 0xad，后面为三字节地址和两子节数据。

符号	参数	最小值	单位
TPU-READ	VDD Min to Read Operation	10	µs
TPU-WRITE	VDD Min to Write Operation	10	µs

规范文件中的相应条目为：

```
0x00|(0xad<<8)|(2<<16), /* No PP, have AAI for 2 bytes */
```

[3] http://www.xmos.com/references/f25l004

25.10 读数据

```
0x0b,                   /* 17. READ cmd */
1,                      /* 18. READ dummy bytes*/
```

从设备读数据的次序一般为发布READ命令、等待清零或更多虚拟子节，然后读一个或更多个数据子节。

图 38为提供此信息的数据单M25P10-A的一部分。有两个可用于读数据的命令：READ和FAST_READ。指示FAST_READ的行显示命令数值为0x0b，后面为一个虚拟子节。

25.11矢量信息

```
SECTOR_LAYOUT_REGULAR,  /* 19. Sector layout */
{32768,{0,{0}}},        /* 20. Sector sizes */
```

第一个数值规定了所有矢量的大小是否相同。所支持的数值为：

SECTOR_LAYOUT_REGULAR

矢量的大小相同

SECTOR_LAYOUT_IRREGULAR

矢量的大小不同

在M25P10-A数据单中可以从第15页的以下段落中找到这些：

内存组织为：

- 131,072子节（每个8比特）
- 4个矢量（256千比特，每个32768子节）
- 512页（每个256子节）

矢量大小规定为结构的一部分：{a, {b, {c}}}。如果是常规矢量大小，则大小规定在a中。b和c的数值应为0。

如果是非常规的矢量大小，则矢量的大小规定在 b 中。每个矢量的页面编号的对数（底数为 2）规定在 c 中。数值 a 应为 0。具有非常规矢量的设备的实例为 AMIC A25L80P[4]。图 45 为提供矢量信息的数据单的一部分。

[4] http://www.xmos.com/references/a25l80p

图 45：A25L80P 数据单第 7 页的表 2

扇区	山区大小（Kb）	地址域	
15	64	F0000h	FFFFFh
14	64	E0000h	EFFFFh
13	64	D0000h	DFFFFh
12	64	C0000h	CFFFFh
11	64	B0000h	BFFFFh
10	64	A0000h	AFFFFh
9	64	90000h	9FFFFh
8	64	80000h	8FFFFh
7	64	70000h	7FFFFh
6	64	60000h	6FFFFh
5	64	50000h	5FFFFh
4	64	40000h	4FFFFh
3	64	30000h	3FFFFh
2	64	20000h	2FFFFh
1	64	10000h	1FFFFh
0-4	32	08000h	0FFFFh
0-3	16	04000h	07FFFh
0-2	8	02000h	03FFFh
0-1	4	01000h	01FFFh
0-0	4	00000h	00FFFh

规范文件中相应的条目为：

```
SECTOR_LAYOUT_IRREGULAR,
    {0,{20,{4,4,5,6,7,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8}}},
```

25.12 状态寄存器比特

```
0x05,                   /* 21. RDSR cmd*/
0x01,                   /* 22. WRSR cmd */
0x01,                   /* 23. WIP bit mask */
```

大部分闪存设备会提供读和写状态寄存器的指示，包括write-in-progress比特掩码。

图 38 为记录 RDSR 和 WRSR 命令的数据单 M25910-A 的一部分。图 43 中的图示出了 WIP 比特在 SR 的比特位置 0，导致比特掩码为 0x01。

25.13 添加支持到xTimeComposer

可以与libflash或xflash一起使用配置文件。下面的实例程序使用libflash连接到M25P10-A设备上，配置参数规定在m25p10a中。

```
#include "platform.h"
#include "flash.h"
#include "flashlib.h"
#include "stdio.h"
#include "stdlib.h"

fl_PortHolderStruct SPI = {PORT_SPI_MISO,
                           PORT_SPI_SS,
                           PORT_SPI_CLK,
                           PORT_SPI_MOSI,
                           XS1_CLKBLK_1};

fl_DeviceSpec myFlashDevices[] = {
 {
   #include "m25p10a"
 }
};

int flash_access() {
  if (fl_connectToDevice(SPI, myFlashDevices,
      sizeof(myFlashDevices)/sizeof(fl_DeviceSpec)) != 0) {
    printf("No supported flash devices found.\n"); exit(1);
  } else {
  printf("Found custom flash device m25p10a.\n"); exit(0);
  }
  return 0;
}

int main() {
  // multicore main is required for xscope
  par {
    on stdcore[0] : flash_access();
  }
}
```

必须在XN文件中规定自定义闪存设备，具体如下：

```
<ExternalDevices>
  <Device NodeId="0" Tile="0" Name="bootFlash"
          Class="SPIFlash" Type="M25P10A">
    <Attribute Name="PORT_SPI_MISO" Value="PORT_SPI_MISO" />
    <Attribute Name="PORT_SPI_SS" Value="PORT_SPI_SS" />
    <Attribute Name="PORT_SPI_CLK" Value="PORT_SPI_CLK" />
    <Attribute Name="PORT_SPI_MOSI" Value="PORT_SPI_MOSI" />
  </Device>
</ExternalDevices>
```

为了汇编链接到 lib 闪存库的图像文件，启动命令行工具（参见§3.2）并输入以下命令：

```
xcc main.xc -o prog.xe -target=target_with_custom_flash.xn -lflash
```

输入以下命令，生成可以在之后编程到上述闪存设备中的 xCORE 闪存格式的图像文件：

```
xflash prog.xe -o imgfile --spi-spec m25p10a
```

XFLASH 生成用于自定义闪存设备的图像，并将之写入规定的图像文件中。

25.14 选择闪存设备

在选择与 xCORE 设备一起使用的闪存设备时，建议遵循以下指南：

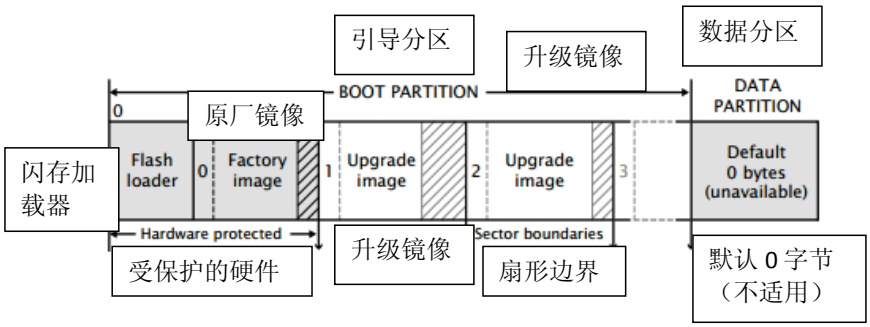
- 如果需要访问数据部分，则应选择一个具有细粒度消除粒度的设备，因为这样会将工厂和升级图像之间的间隙最小化，还可以将在写数据时libflash需要缓冲的数量最小化。
- 如果可能，选择具有矢量保护的设备以确保引导装载程序和工厂图像有保护，防止在部署后意外受损。
- 选择一个适用于应用的闪存速度等级。即便在低下启动时间也是最短的。

26 XFLASH 命令行手册

在本章中
- 总体选项
- 目标选项
- 安全选项
- 编程选项

如下图所示，XFLASH 可用于创建 XCORE 闪存格式的二进制数。该命令还可以将此类文件用于启动 XMOS 系统的闪存设备。

图 46：Flash 格式图



26.1 总体选项

下列选项用于指定程序的镜像和数据所构成的二进制数字及其布局。在插入填充时必须确保镜像对齐扇区边界。

xe-file [size]

--factory xe-file [size]

指定的 Xe 文件为原厂的镜像。如果相应尺寸被指定，填充的插入使这一镜像的开始和下一个镜像之间的空间至少为指定的容量。尺寸默认单位是“字节”，可以后缀 K 来指定单元的字节。

可指定最多一个原厂镜像。

--upgrade id xe-file [size]

指定的文件版本 ID 为 Xe 各版本号升级镜像必须是一个大于 0 的独一无二的号码。如果容量被指定，填充的是插入使这一镜像的开始和下一个镜像之间的空间至少为指定的容量。尺寸默认单位是“字节”，容量可以后缀 K 来指定单元的字节。

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

多个升级镜像按照命令行中指定的顺序插入启动分区。
如未指定任何原厂镜像，单一的升级镜像可以被指定和写入选项文件。

--boot-partition-size n
指定的启动分区的容量为 n 个字节。如果未指定，使用默认容量的闪存设备的总容量。n 必须大于或等于所需的存储引导程序、原厂镜像和任何升级镜像的最小尺寸。

--data file 指定要写入的数据分区的文件的内容。

--loader file
指定自定义闪存装载函数文件（见§22.4）。文件可以是一个对象（.O）或档案（.a）。
默认情况下，该 xcore Flash 用最高版本加载镜像，验证其 CRC。

--verbose 印刷品附加信息的程序加载到目标系统。

--help 打印一个描述支持指令的选项。

--version 显示版本号和版权。

26.2 目标选项

下列选项用于指定哪个闪存装置的二进制为可编程的。闪存装置使用的类型决定了 SPI 分频器值，扇区的容量和内存容量。

--list-devices
-l 打印所有连接到 PC 的 JTAG 适配器的枚举列表和每个 JTAG 链中的设备，形式如下：

ID	名称	适配器 ID	设备

适配器按其序列号订购。

--id ID 指定连接到目标硬件的适配器。
XFlash 连接到目标平台并确定连接到它的闪存装置的类型。

--adapter-id ADAPTER-SERIAL-NUMBER
指定连接到目标硬件的适配器序列号。
XFlash 连接到目标硬件并确定连接到它的闪存装置的类型。

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

--jtag-speed n
设置用于 JTAG 时钟的分频器。相应的 JTAG 时钟速度为 6/(n+1)兆赫。分压器的默认值为 JTAG 时钟是 0，代表 6 兆赫。

--spi-spec file
指定的可支持的闪存设备见文件（见§25）。

--spi-div n 设置用于 SPI 时钟的分频器，生产 100 / 2n 兆赫的 SPI 时钟速度。默认情况下，如果未指定目标，分频器的值设置为 3（16.7 兆赫）。

--noinq 不运行驱动程序的检查设备，核对检查镜像对齐扇区边界。如果 –noinq 被省略，期望 XFlash 能够通过 JTAG 连接到设备。

--disable-boot-link-warn
禁用警告发出时，节点之间的联系不允许从链接到工作的启动，例如仅低于已被指定为连接节点的链接 3，则引导 ROM 链接 4-7（如果链接 3 为主瓦）。

26.3 安全选项

下列选项应与 AES 模块一同使用（见第 27.1 节）。

--key keyfile
用密匙文件加密引导分区中的镜像。.

--disable-otp
使闪存装载在程序启动时禁止访问 OTP 存储器。这是默认的选项——加密。

--enable-otp
使闪存装载在程序启动时能够访问 OTP 存储器。这是默认的选项——加密

26.4 编程选项

默认情况下，XFlash 程序生成二进制文件到目标的闪存设备。

-o file 位置输出文件，禁用程序。
如果目标平台是从一个以上的闪存装置启动，为每一个装置创建多个输出文件。每个输出文件的名称是 file_node，在相应节点的 ID 属性的值（见§42.4）。

下列选项在目标闪存设备上执行通用的读取、写入和擦除的操作。目标的文件必须被指定，它提供了用于对硬件平台的 SPI 设备通信端口。

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

--target-file xn-file [node]
指定的 xn-file 文件作为目标平台。
如 xn-file 指定多个闪存装置，则节点的值必须被指定。该值必须对应连接到目标闪存设备节点的 ID 属性（见§42.4）。

--target platform [node]
指定一个目标平台。该平台的配置必须在文件 platform.xn 被指定，被搜索的路径由 xcc_device_path 环境变量指定（见§9.8）。
如果 xn-file 指定多个闪存装置，则节点的值必须被指定。该值必须对应连接到目标闪存设备节点的 ID 属性（见§42.4）。

--erase-all 清除闪存装置的所有存储。

--read-all 读取闪存设备的所有内存内容并写入到主机。必须同-o 共同使用。

--write-all file
以字节为单位的文件写入到闪存设备。

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --

Shiqiang Xiao 13-11-7 12:58 PM
已删除: --