

武汉理工大学

硕士学位论文

ARM CoreSight跟踪调试技术的研究与应用

姓名：王阳赞

申请学位级别：硕士

专业：计算机应用技术

指导教师：张能立

20090601

中文摘要

随着嵌入式系统的发展、嵌入式应用的不断增长以及嵌入式系统复杂性不断提高,嵌入式软件的规模和复杂性也不断提高。在目前的嵌入式系统开发中间,软件开发占 80%以上的工作量,嵌入式软件的质量和开发周期对产品的最终质量和上市时间起到决定性的影响。因此,为了保持产品竞争力,支持用户对嵌入式设备进行快速、高效的软件开发,嵌入式的开发人员迫切需要更加强大的调试技术和手段来为开发复杂的嵌入式应用提供帮助;同时,强有力的嵌入式软件开发工具也是基本的必备条件。

ARM 公司于 2004 年推出了一种新的调试体系结构 CoreSight,以提高更为强大的调试能力。CoreSight 体系结构支持多核系统的调试,能对全系统进行高带宽的实时跟踪,包括对系统总线的跟踪与监视。同时,ARM 推出了支持 CoreSight 调试技术的 RVDS 开发系统,该系统基于开放源码 Eclipse IDE 开发,可帮助设计工程师简化多核处理器或多处理器架构下软件开发的复杂度。

在研究和分析 ARM CoreSight 调试体系结构的基础上,本文结合 ARM 公司 RVDS 集成开发环境中调试模块组成部分 Event Viewer 系统的开发,实现了对通过原始数据源采集到的 CoreSight 跟踪数据的完整实时解析,并最终在显示模块中将其包含的信息以可视化的形式直观地展现给用户,以供后续的程序性能分析和嵌入式软件系统调试。

论文首先研究了与本课题相关的一些技术,包括 CoreSight 调试体系结构、嵌入式常见调试技术、Eclipse 平台体系架构及其插件扩展点技术。

其次,本文在研究嵌入式集成开发环境国内外现状及其发展趋势的基础上,结合 Event Viewer 系统的整体需求,介绍了系统的总体设计及其功能模块划分,并给出了系统的第三方扩展设计。

然后,本文着重讨论了系统解析模块的设计与实现。在分析 CoreSight 跟踪数据解析流程的基础上,对系统中解析模块进行了详细设计,并完成了基于 ITM 数据流的解析实现。结合系统的功能需求和解析模块的设计,本文利用 Eclipse 插件扩展点机制,划分解析模块提供对外扩展,实现了系统向第三方产品提供商提供扩展接口的功能,第三方可以在此基础上提供自己的解析处理。

最后,利用 Eclipse View 扩展点和 SWT/JFace 技术,实现了对跟踪数据的前台展示,包括 Text、Event、Analog 三种类型;本文着重讨论了 Analog 展示部分的详细设计和实现,将解析后得到的 Analog 数据信息以实时曲线图的形式展现给客户,提供对 Analog 数据变化趋势的直观描述。

本课题的研究成果可用于 ARM Core 的最终用户以及其他第三方产品提供商,对分析目标板程序的性能,调试嵌入式软件系统等都提供了较大的帮助。

关键字: CoreSight, Event Viewer, 解析, 跟踪数据, 插件扩展点

Abstract

With the development of embedded systems, increase of embedded applications and embedded systems complexity, the scale and complexity of embedded software system also grow increasingly. Software development account for more than 80% workload in the current embedded systems development, the quality and development cycle of embedded software have a decisive impact on the final quality and time-to-market of embedded product. Therefore, in order to maintain the competitiveness of the product, to support the user to execute embedded device software development faster and more efficient, embedded developers demand more powerful debugging techniques and tools for help in the development of complex embedded applications.

In 2004, ARM had launched a new debug architecture named CoreSight to enhance the capacity of debugging. CoreSight architecture support the debugging of multi-core system, and provide high-bandwidth and real-time tracking for the whole system, including the tracking and monitoring for system bus. At the same time, ARM introduced a powerful embedded Integrated Development Environment named RealView Development Suite which is based on open source Eclipse IDE to develop and support CoreSight debugging technique, it can help design engineers to simplify the complexity of software development in multi-core processors or multi-processor architecture.

This thesis, according to the study and analysis of ARM CoreSight debug architecture and the development of Event Viewer system which is the component part of the debugging module in RVDS integrated development environment, implements complete and real-time parsing for CoreSight trace datas which collected through the raw data source. Ultimately, these data information will be intuitionistic displayed to user by visual form in the display module, and it can provide much help for the follow performance analysis and debugging of embedded software systems.

Firstly, this thesis studies some related techniques, including CoreSight debugging architecture, some common embedded debugging technology, Eclipse platform architecture and plug-in extension points.

Secondly, this thesis, according to the study of status quo and development trend in embedded IDE and the requirement of Event Viewer system, introduces the overall design of this system and the dividing of function modules; Ultimately, gives a systematic third-party expansion design.

Thirdly, this thesis focuses on the design and implementation of system parsing module. According to the analysis of the flow for parsing CoreSight trace data, this

this thesis introduces detailed design of the parsing module, and implements the parsing process based on ITM data stream. According to the requirement of system function and the design of parsing module, this thesis uses Eclipse plug-in extension points mechanism to divide parsing module for providing the third-party expansion interfaces, the third parties can provide their own parsing process by using these interfaces.

Finally, this thesis uses Eclipse View extension point and SWT/JFace technology to design and implement the display module which including text, event, analog three types. This thesis focuses on the detailed design and implement of analog display part, these data information collected after parsing will be displayed to user by real-time curve form in analog display part, to provide visual description of analog datas trend.

Results of research on this thesis can be used by the ARM Core user and the other third-party product providers, it can provide a greater help for the performance analysis of target board procedures and debugging of embedded software systems.

Key words: CoreSight, Event Viewer, parsing, trace data, plug-in extension points

缩略表

ARM:	Advanced RISC Machines
DAP:	Debug Access Port, 调试访问端口
DCC:	Debug Communications Channel, 调试通信通道
ECT:	Embedded Cross Trigger, 嵌入式交叉触发
ETB:	Embedded Trace Buffer, 嵌入式跟踪缓冲区
ETM:	Embedded Trace Macrocell, 嵌入式跟踪宏单元
HSP:	Hardware Source Packet, 硬件源包
HTM:	AHB Trace Macrocell, AHB 跟踪宏单元
ICE:	In-Circuit Emulator, 在线仿真器
IDE:	Integrated Development Environment, 集成开发环境
IDS:	Intelligent Data Source, 智能数据源
ITM:	Instrumentation Trace Macrocell, 仪器跟踪宏单元
JDT:	Java Development Tools, Java 开发工具
MCU:	Micro Controller Unit, 微控制单元
OCD:	On-Chip Debug, 片上调试
OCP:	Open-Closed Principle, 开闭原则
PDE:	Plug-in Development Environment, 插件开发环境
RDS:	Raw Data Source, 原始数据源
RVDS:	RealView Development Suite
RVI:	RealView ICE
RVT:	RealView Tracer
SG:	Stream Generation, 流生成器
SIP:	Software Instrumentation Packet, 软件仪器包
SoC:	System-on-Chip, 片上系统
SWO:	Serial Wire Output
TPIU:	Trace Port Interface Unit, 跟踪端口接口单元

独 创 性 声 明

本人声明，所呈交的论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得武汉理工大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

签 名： 王阳赞 日 期： 2009.6.11

关于论文使用授权书

本人完全了解武汉理工大学有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权武汉理工大学可以将本学位论文的全部内容编入有关数据库进行检索，可以采用影印、缩印或其他复制手段保存或汇编本学位论文。同时授权经武汉理工大学认可的国家有关机构或论文数据库使用或收录本学位论文，并向社会公众提供信息服务。

（保密的论文在解密后应遵守此规定）

签 名： 王阳赞 导师签名： 张永红 日 期： 2009.6.12

第 1 章 绪论

1.1 课题背景和意义

今天的嵌入式产品正在变得越来越复杂，以最常见的移动电子设备手机为例，消费者希望它能够变成一个集大成的平台，具有包括 TV、视频回放、GPS 导航定位、GPS 立体地图、拍照和摄像、音乐播放、通过互联网收发邮件和聊天等在内的所有功能。因此，未来的嵌入式产品很大一部分将采用多核处理器或多个处理器来开发，操作系统也将需要采用更复杂的 Windows CE、Symbian、Linux 和 Enea OSE，这也意味着嵌入式软件系统的开发将变得越来越复杂。可以看出，随着嵌入式系统的发展、嵌入式应用的不断增长以及嵌入式系统复杂性不断提高，嵌入式软件的规模和复杂性也不断提高。在目前的嵌入式系统开发中间，软件开发占 80% 以上的工作量，嵌入式软件的质量和开发周期对产品的最终质量和上市时间起到决定性的影响。因此，为了支持用户对嵌入式设备进行快速、高效的软件开发，嵌入式的开发人员迫切需要更加强大的调试技术和手段来为开发复杂的嵌入式应用提供帮助；同时，强有力的嵌入式软件开发工具也是基本的必备条件。

传统的 JTAG 调试方法能力非常有限，尤其是当目标设备实时运行的时候，开发人员通常只能使用串口打印输出的方法来做调试。当需要分析内存错误、多任务之间关系、异常或中断处理、睡眠模式等情况时，目前的调试方法基本无效，系统实时运行时基本是黑箱运行。在这种情况下，2004 年 ARM 推出了一种新的调试体系结构 CoreSight，以提高更为强大的调试能力。CoreSight 体系结构支持多核系统的调试，能对全系统进行高带宽的实时跟踪，包括对系统总线的跟踪与监视。ARM CoreSight 技术为完整的片上系统（SoC）设计提供最全面的调试、跟踪技术方案，通过最小端口可获得全面的系统可见度。ARM CoreSight 技术可快速地对不同的软件进行调试，通过对多核和 AMBA 总线的情况进行同时跟踪。此外，同时对多核进行暂停和调试，CoreSight 技术可对 AMBA 上的存储器和外设进行调试，无需暂停处理器工作，达到不易做到的实时开发。CoreSight 体系结构非常灵活，其中各个部件可以根据处理器厂商的需要而进行组合。

同时, ARM 推出了可帮助设计工程师简化多核处理器或多处理器架构下软件开发复杂度的 RVDS 开发系统。RealView Development Suite^[1] (RVDS) 是 ARM 公司继 SDT 和 ADS1.2 之后主推的新一代研发工具, 主要针对基于 ARM 处理器的 SoC、ASSP 和复杂多内核 ASIC 的系统开发者, 它基于开源源码 Eclipse IDE, 支持 CoreSight 调试技术, 它的编译器 RVCT 是目前业界所有针对 ARM 处理器的编译器中最好的, 它的调试工具支持带嵌入式 OS 的复杂单核和多核 SoC 的软件开发, 它支持 Windows XP 专业版、Windows Vista 商业版和企业版、以及 Red Hat 企业版 Linux V4/V5 运行平台。

Event Viewer 系统作为嵌入式 IDE 中调试模块的组成单元, 将被集成到 RVDS 中, 成为 ARM RVDS 集成开发环境的一部分。实现此系统的目的在于向 ARM Core 的最终用户提供采集、分析并展示 ARM CoreSight 跟踪数据的功能, 并向第三方产品提供商提供扩展接口的功能, 以方便第三方产品提供商产生自定义的跟踪数据。本文结合该系统的开发, 利用 Eclipse 插件扩展点机制, 划分解析模块提供对外扩展, 同时对 CoreSight 跟踪数据实现完整解析, 并最终通过显示模块在 Eclipse 中以可视化的形式重构目标板程序的运行状况。课题的研究成果可用于 ARM Core 的最终用户以及其他第三方产品提供商, 对分析目标板程序的性能, 调试嵌入式软件系统等都提供了较大的帮助。

1.2 国内外研究现状

随着嵌入式系统硬件技术与软件技术的不断发展, 以手机、PDA、信息家电为代表的各种嵌入式产品正在广泛而深入地改变着人们的生活。日益成熟的硬件技术以及更加复杂化的应用需求, 使得软件逐步取代硬件成为系统的主要组成部分。嵌入式软件系统的开发是否能跟上市场需求的变化, 成为制约嵌入式产品能否占据市场的关键因素。各种优秀的集成开发环境 IDE (Integration Develop Environment) 的出现, 使得嵌入式系统开发变得相对简单, 降低了嵌入式应用软件的开发难度, 有效缩短了嵌入式软件的开发周期, 从而对广大工程人员降低了门槛。目前有很多工程人员正在用各种开发平台进行嵌入式系统开发, 从 ARM 公司的 SDT 到 ADS, 再到 RVDS, 以及风河公司的 Tornado 和其他公司提供的各种专用开发工具, 这些 IDE 一般都是由文件管理器、编译器、汇编器、链接器、调试器等组件和工具组成。

1.2.1 国外研究现状

国外早在 20 世纪 80 年代就开始了嵌入式实时操作系统(以下简称 RTOS)的研究,所以早期的嵌入式 IDE 主要是国外的产品,是由第三方工具公司提供根据操作系统特性和硬件特性定制。国外比较流行的嵌入式 IDE 主要有:

1)ARM 公司的 SDT,英文全称为 ARM Software Development Kit,是 ARM 为方便用户在 ARM 芯片上进行应用软件开发而推出的一整套集成开发工具。

2) ARM 公司的 ADS,英文全称为 ARM Developer Suite,是 ARM 推出的新一代 ARM 集成开发工具,用来取代 ARM 以前推出的开发工具 ARM SDT。

3) ARM 公司的 RVDS^[2],英文全称为 RealView Development Suite,是 ARM 公司继 SDT 和 ADS1.2 之后主推的新一代研发工具。RVDS 集成的 RVCT 是业内公认的能够支持所有 ARM 处理器,并提供最佳的执行性能的编译器;RVD 是 ARM 系统调试方案的核心部分,支持含嵌入式操作系统的单核和多核处理器软件研发,能同时提供相关联的系统级模型构建功能和应用级软件研发功能,为不同用户提供最为合适的调试功效。

4) WindRiverSystems 公司(简称风河公司)的 Tornado^[3]开发环境,是嵌入式实时领域里面新一代的开发调试环境,是交叉开发环境运行在主机上的部分,是开发和调试 Vxworks 系统不可缺少的组成部分。Tornado 是集成了编辑器,编译器,调试器于一体的高度集成的窗口环境,给嵌入式系统开发人员提供了一个不受目标机资源限制的开发和调试环境。

5) WindRiverSystems 公司的 Workbench^[3]开发环境,继推出 Tornado 开发环境和 VxWorks 嵌入式操作系统产品后,风河公司新一代 Workbench 开发平台继承了其原有的 Tornado 集成开发平台的一贯优势,并且功能更加强大,由于新采用了先进的 Eclipse 软件框架结构,从而使整个系统更加开放和易于扩展。

6) Linux Work 公司的 VisualLinux 集成开发环境,需要与微软的 Visual Studio 一起安装,是针对 Linux 的嵌入式 IDE。

7)QNX 公司的 QNX IDE,也是根据 Eclipse 的架构设计的,针对 QNX RTOS 的嵌入式 IDE。

8) Microsoft 公司的 EVC 开发环境,针对 Windows CE 的嵌入式 IDE。

另外还有由自由软件协会 GNU(GNU's Not Unix)组织提供的免费研究开发成果,如针对特定处理器的本地编译器 GCC(GNU C Compiler)和交叉编译器

CGCC(Crossing GNU C Compiler)等。目前已有公司在 GNU 软件基础上推出商业化版本的嵌入式 IDE。

1.2.2 国内研究现状

目前由于嵌入式软件的广泛应用,无论国内外,众多 RTOS 开发商开始推出针对自己 RTOS 开发的嵌入式 IDE。虽然国内对嵌入式 RTOS 研究起步比较晚,但现在我国嵌入式的研究相比以前已不可同日而语,在国内已经出现了一些自主研发的嵌入式 IDE,目前比较成功的主要有:

1) 深圳市英蓓特信息技术有限公司的 Embest IDE^[4] (Embest Integrated Development Environment), 是英蓓特公司推出的一套应用于嵌入式软件开发的新一代集成开发环境。Embest IDE 是一个高度集成的图形界面操作环境,包含编辑器、编译器、汇编器、链接器、调试器等工具,其界面同 Microsoft Visual Studio 类似。Embest IDE 支持 ARM、Motorola 等多家公司不同系列的处理器。Embest IDE 运行的主机环境为 Windows95/98/NT/Me/2000, 支持的开发语言包括标准 C、Embedded C 和汇编语言。

2) 北京科银京成技术有限公司的 LambdaTool, 针对嵌入式 RTOSDeltaOS。

3) 武汉创维特公司的 ADT1000 开发环境, 支持 JTAG 仿真器。

4) 青岛海尔软件有限公司的海尔嵌入式 IDE, 目前支持 ARM 系列处理器。

1.3 研究的目标和内容

本课题来源于武汉理工大学英蓓特嵌入式研发中心开发的项目: RealView Event Viewer。此系统是 ARM RVDS 集成开发环境的一部分。本课题的研究目标是: 旨在按照预先定义好的格式, 实现对 ARM CoreSight 跟踪数据的解析处理, 获得完整的包格式数据, 并将其包含的信息在 Eclipse 中以可视化的形式直观地展现给用户, 以供后续的程序性能分析和嵌入式软件系统调试; 同时, 利用 Eclipse 插件扩展点机制向第三方产品提供商提供扩展接口的功能, 第三方产品提供商可以在此基础上提供自己的解析处理。

本课题研究的主要内容如下: 关注嵌入式 IDE 国内外研究现状及其发展趋势; 研究 ARM CoreSight 调试体系架构及嵌入式常用调试技术; 研究和分析 Eclipse 开源平台的特点和插件开发技术; 分析 ITM 包数据格式, 研究 ARM

CoreSight 跟踪数据解析模块的详细设计和基于 ITM 数据流的解析实现；研究 Eclipse 插件扩展机制在实现系统第三方扩展接口中的应用；研究 ARM CoreSight 跟踪数据前台展示的基本体系架构和功能模块划分，以及 Analog 展示部分的详细设计和实现。

1.4 论文结构

第一章，绪论，介绍了课题背景和意义，国内外研究现状以及本文研究的目标和主要内容。

第二章，相关技术研究，包括 CoreSight 调试体系结构、嵌入式常见调试技术、Eclipse 平台相关以及插件扩展点技术的研究。

第三章，Event Viewer 系统简介，分析了系统的整体需求，简单介绍了系统的总体架构，并给出了系统的第三方扩展设计。

第四章，CoreSight 跟踪数据的解析，提出了系统中 CoreSight 跟踪数据的解析流程，在此基础上给出了解析模块的详细设计，着重讨论了基于 ITM 数据流的解析实现，最后详细介绍了本系统中第三方扩展的实现。

第五章，CoreSight 跟踪数据的前台展示，提出了前台展示的基本体系架构，在此基础上给出了功能模块的划分及其简要说明，着重讨论了 Analog 展示部分的设计与实现，将 CoreSight 跟踪数据的信息以实时曲线图的形式展现给客户，提供对 Analog 数据变化趋势的直观描述。

第六章，测试与分析，主要介绍了解析器单元测试和解析模块集成测试的流程，并对测试结果进行了分析。

第七章，总结，总结了论文的主要内容，讨论了论文的不足及未来的工作。

- DAP(Debug Access Port): 可以实时访问 AMBA 总线上的系统内存、外设寄存器, 以及所有调试配置寄存器, 而无需挂起系统;
- ECT (Embedded Cross Trigger): 包含 CTI (Cross Trigger Interface) 和 CTM (Cross Trigger Matrix), 为 ETM (Embedded Trace Macrocell) 提供一个接口, 用于将一个处理器的调试事件传递到另一个处理器。
- 源部件: 用于产生向 ATB (AMBA Trace Bus) 发送的跟踪数据, 典型的有:
 - HTM (AHB Trace Macrocell): 用于获取 AHB 总线跟踪信息, 包括总线的层次、存储结构、时序、数据流和控制流等;
 - ETM (Embedded Trace Macrocell): 用于获取处理器核的跟踪信息;
 - ITM (Instrumentation Trace Macrocell): 是一个由软件驱动跟踪源, 其输出的跟踪信息可以由软件设置, 包括 Printf 类型的调试信息、操作系统以及应用程序的事件信息等。
- 连接部件: 用于实现跟踪数据的连接、触发和传输, 典型的有:
 - ATB 1:1 bridge: 具有两个 ATB 接口, 用于传递跟踪源发出的控制信号;
 - Replicator.: 可以让来自同一跟踪源的数据同时写到两个不同的汇集点去;
 - Trace Funnel: 用于将多个跟踪数据流组合起来, 在 ATB 总线上传输。
- 汇集点: 是芯片上跟踪数据的终点, 典型的有:
 - TPIU (Trace Port Interface Unit): 将片内各种跟踪源获取的信息按照 TPIU 帧的格式进行组装, 然后通过 Trace Port 传送到片外;
 - ETB (Embedded Trace Buffer): 一个 32 位的 RAM, 作为片内跟踪信息缓冲区;
 - SWO (Serial Wire Output): 类似 TPIU, 但仅输出 ITM 单元的跟踪信息, 只需要一个引脚来实现。

对于带有 CoreSight 调试体系结构的处理器, 工程师就可以实现实时调试, 当应用程序在处理器上全速运行时, 可以透明地观察并记录处理器中的各种事件, 包括内存单元读写、中断异常的发生与处理、操作系统任务之间的触发关系与运行过程等等。这一新的调试体系结构将嵌入式系统调试从黑盒变为了白盒, 使得工程师有能力应付更复杂的系统的设计与调试。

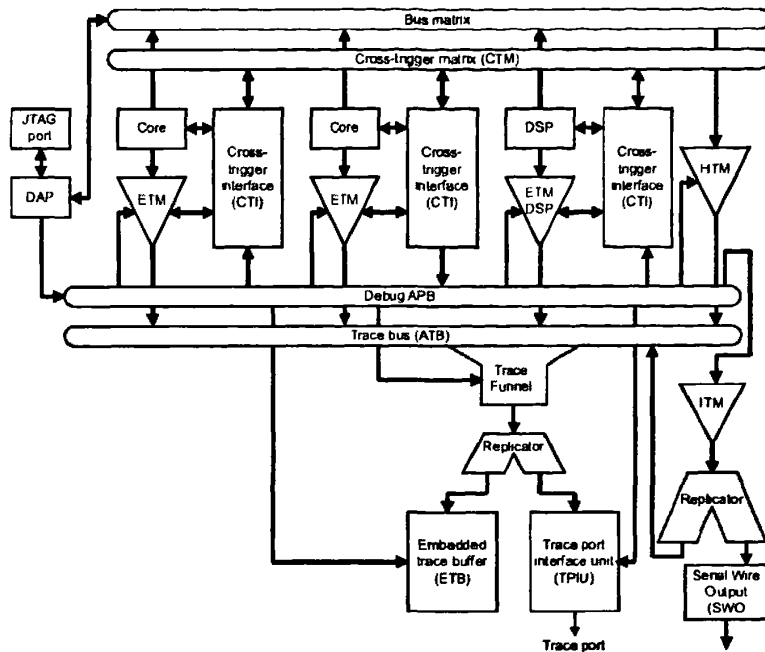


图 2-1 典型的 CoreSight 调试结构

使用 Trace Port 接口进行调试还需要专用的跟踪器 (Tracer)，如图 2-2 所示，ARM 公司的开发工具 RVDS 中 RVT (RealView Tracer) 就是这种跟踪器，其价格较为昂贵，因此 RVDS 更适合 SoC 的设计和开发。

仿真器 RealView ICE 是一种基于 JTAG 的调试解决方案，用于调试运行在基于 ARM 体系结构的处理器上的软件。跟踪器 RealView Tracer 为那些在具有深层嵌入式处理器内核的前沿芯片上系统设备中运行的软件提供实时跟踪功能。RealView Trace 2 还可使数据直接流向调试主机中的开发工具，以执行实时硬件平台性能分析^[7]。三者可结合使用。

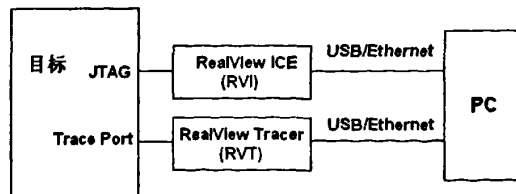


图 2-2 RVDS 工具方案

为了提供廉价的实时调试工具，ARM 公司在其针对 MCU 的开发工具 RealView MDK^[8]中，增加了使用图 2-3 中的 SWO 接口进行实时跟踪的功能，使用仿真器 ULINK 2 的 JTAG 接口中的 2 根引脚作为 SWO 接口，可以对 Cortex-M3 处理器的实时调试分析，能从 ITM 单元中实时获取内存单元读写信息、Printf 打印信息、操作系统任务信息等。

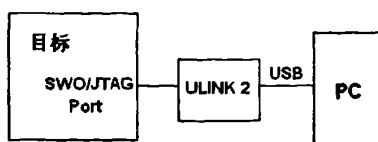


图 2-3 MDK 工具方案

2.3 嵌入式常用调试技术

相对于传统的应用程序开发，嵌入式软件开发具有以下几个特点：

1) 在嵌入式软件开发过程中，由于目标机不具有自举开发的能力，通常有宿主机和目标机之分，宿主机是执行编译、链接、定址过程的计算机，目标机是运行嵌入式软件的硬件平台，而传统应用程序的整个开发过程都在 PC 机上完成。

2) 嵌入式系统开发中，宿主机和目标机运行的指令集一般是不一样。

3) 嵌入式软件所运行的硬件平台资源有限，一般没有复杂的操作系统支持，即使有操作系统支持，也主要是用于嵌入式应用软件的支撑，而不是用于目标机的开发环境平台。

4) 嵌入式系统一般没有强大的图形用户界面，即使有图形用户界面，也主要是用于嵌入式应用软件，而不是用于提供开发环境支撑。

5) 嵌入式软件需要特殊的软硬件将其固化到嵌入式系统中。

由于嵌入式软件开发的这些特点，使得它的调试过程与传统应用程序的调试很不一样。通常程序员利用调试器来跟踪程序的执行情况，快速有效地定位错误，并改正错误。在传统的应用程序开发中，采用本地调试方式，即调试器与被调试的程序是运行在同一台机器上、相同操作系统上的两个进程，调试进程通过操作系统提供的调试接口控制、访问被调试进程。而嵌入式软件开发中

多采用远程调试^[9]（交叉调试）方式，让调试器运行在主机上，被调试程序运行在目标机上，主机和目标机之间通过某种通信手段（如并口线、USB 总线、以太网、仿真器等）连接起来，这样调试器就可以控制、跟踪被调试的程序，如图 2-4 所示。下面介绍几种常用的调试技术^[10,11]。

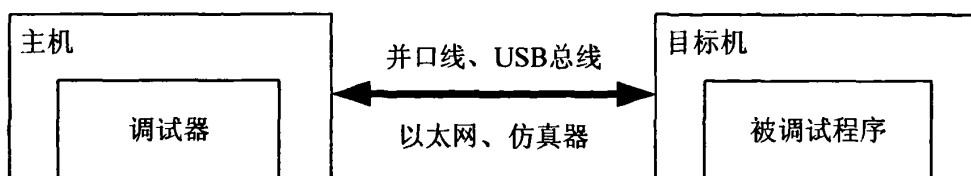


图 2-4 远程交叉调试

2.3.1 软件监控技术

软件监控（通常称为“插桩”，英文称 Stub）技术就是在目标机和调试器内分别加入某些功能模块，二者互通信息来进行调试。软件监控方式调试架构如图 2-5 所示，使用这种技术最常见的就是 GDB^[12]（GNU Debugger）远程调试。

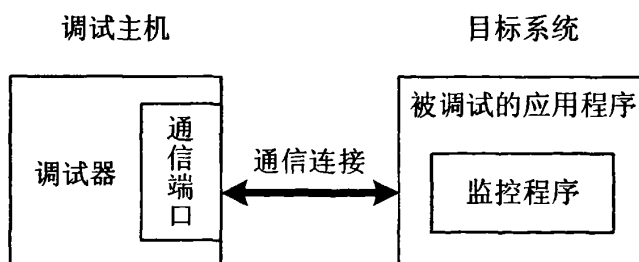


图 2-5 软件监控方式的调试架构

一个完整的 GDB 远程调试系统可分为三部分：GDB 调试器、GDBStub 和远程通信协议。GDB 调试器是驻在 PC 机上的上位机调试器，GDBStub 是目标系统上运行的一段程序，它是调试主机与被调试程序之间的一个媒介。根据 GDB 调试器的功能和作用，Stub 实现以下基本功能：负责执行调试器的命令，包括读写存储、查询和设置寄存器、单步执行和设置断点等；当程序发生异常时，Stub 作为中断服务处理程序与调试器交互信息并显示给用户，远程通信协议负责调试主机与目标系统的通信。

软件监控技术的主要包括以下几个方面：调试主机与目标系统在遵循远程调试协议下通过指定通信端口（串口、网口）进行通信；当被调试程序产生异常时及时通知调试器；调试器通过发送调试命令对被调试程序进行访问和控制，调试器的这些命令实际上都将转换成对被调试程序的地址空间或目标系统上的某些寄存器的访问，目标系统接收到这样的请求后可以直接进行相应的处理。

这一方案的实质是在调试主机上集成调试软件环境，用软件接管目标系统的全部异常处理及部分中断处理，在目标系统上插入调试端口通信模块，与主机的调试器交互。

软件监控技术的优点是几乎不需要修改硬件，只需要编写监控程序和使用一定的通信接口即可。但它的缺点也很明显。它只能在目标系统初始化，特别是调试通信端口初始化完成后才起作用；它必须接管目标系统全部异常处理，不适用于底层软件开发，所以一般只用于调试运行于目标系统之上的应用程序，而不宜用来调试目标机上的操作系统，特别是无法调试目标机操作系统的启动过程。而且由于它必然要占用目标平台的某个通信端口，该端口的通信程序就无法调试了^[13]。

2.3.2 在线仿真器

在线仿真器 ICE^[14]（In-Circuit Emulator）是一种专用的调试设备。如图 2-6 所示，它一般由两部分组成：其中一部分用来连接调试主机；另外一部分是被插入到目标系统的处理器插槽的仿真探头，它包含一个与被替代微处理器功能完全相同的处理器，但是为了达到调试的目的，仿真探头里的处理器根据调试需求做了一些特殊处理。它能产生并捕获外部电路所需的所有信号，从而满足用户查看处理器内部运行状态的需求。

一般而言 ICE 实现对目标处理器完全物理替代，用户要将目标系统上的处理器完全拔出，然后将仿真探头插入目标系统的微处理器插槽，在整个系统开发期间目标系统上的微处理器被 ICE 完全替代。因而目标系统的所有操作都可以通过 ICE 传送到调试主机接口被监控。ICE 上的处理器不同于普通处理器，它是一种绑出（Bond-out）处理器，即将处理器的某些内部信号连接到外部管脚上，从而使得外部逻辑可以通过这些管脚监视和控制内部处理器的运行状态，进而达到调试的目的。

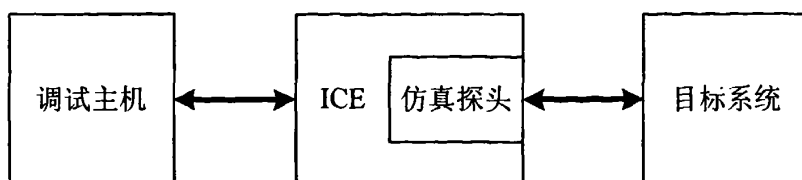


图 2-6 在线仿真的调试架构

在线仿真器的工作流程可以总结如下：仿真器复位，微处理器开始运行待调试程序；在线仿真器实时监测仿真接口信号，并根据检测到的信号（如指令地址）判断是否有断点触发；一旦断点触发，仿真器暂停待调试程序的运行，通过切换程序运行的地址空间开始执行监控程序；将当前微处理器的状态信息（如寄存器的值）通过通信接口发送给调试主机；用户通过分析这些调试反馈判断待调试程序在目标系统上的执行过程；监控程序执行完毕后，再次切换程序运行地址空间，继续执行待调试程序；用户也可以在切换程序运行地址空间之前通过修改微处理器内部寄存器的值，控制微处理器的运行。

ICE 是一种基于硬件支持的调试方法，它的优点是不消耗目标系统资源，可以对微处理器进行实时跟踪，也可以实时观察目标系统上的存储与 I/O，这对于没有软硬件调试支持的目标系统的调试起到很好的效果^[15]。但是它也有明显的缺点：开发成本高、价格昂贵、可扩展性差。而且随着嵌入式设备集成度的提高，目标系统上的微处理器不再单独存在，而是与其它外围集成在一起，即 SoC。这时在线仿真器就无法通过对目标系统上微处理器的替换实现基本的调试功能，而只能对目标系统上的除了微处理器以外的其它模块和软件系统进行调试。

2.3.3 片上调试技术

片上调试^[16]（On-Chip Debug，简称 OCD）通过在目标系统上集成单独的调试控制模块接管微处理器的调试异常处理。用户可以通过与调试控制模块的通信实现对调试异常和中断的设置。一般来说，采用两种模式来实现，即将 CPU 的工作模式分为运行模式和调试模式两种。当 CPU 处于调试模式时，它不再从存储器中读取指令而是从调试端口读指令，这样调试器可以直接向目标机发送调试命令，完成各种调试功能，从而实现对存储单元、寄存器的读写访问，控制被调试程序的运行过程。

OCD 实现了调试协议与通信协议的分离。调试协议可以由用户开发，也可

以遵循一定的标准，它基于片上的硬件调试支持逻辑的特性和调试接口的特性。而通信协议则比较灵活，可以根据需要设计各种不同的通信接口以提高调试速度和效率。一般通过调试协议转换器进行调试协议与通信协议之间的信号转换。该技术的调试架构如图 2-7 所示。

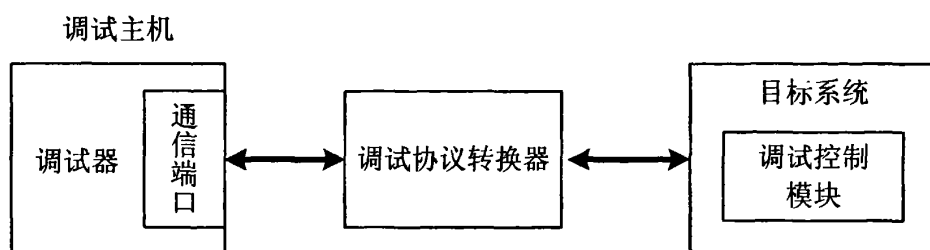


图 2-7 片上调试架构

OCD 调试方案具有以下特点：在目标系统上嵌入特定的调试模块，当满足一定的触发条件（该触发条件一般由用户通过调试主机设置）时微处理器进入调试状态；进入该状态后，首先执行一段微处理器当前状态的保护程序，以保证处理器跳出调试态后能正确的运行原有应用程序。

这种调试方式的优点是不占用目标机上的资源，调试环境和程序最终的运行环境一致。与插桩方式相比较，OCD 不占用目标系统的通信端口与存储单元，无需修改目标操作系统，可方便调试操作系统的启动过程，大大方便了系统开发人员。与 ICE 相比，OCD 将调试控制模块集成在微处理器内部，可以更方便的实现对硬件系统的调试，虽然 OCD 的开发会增加一些硬件单元，但随着系统集成能力的提高，增加一些专用于调试的电路是完全必要且可行的^[17]。

2.3.4 TRACE 调试技术

片上调试技术是典型的 run-control 调试方式，调试过程可概括为：设置断点——程序暂停——观察程序运行状态——返回继续运行。在调试中，有些 Bug 比较明显，它产生后就会留下“证据”，微处理器通过 run-control 的方式停顿后仍然可以检测系统中的 Bug。但有些 Bug 非常“狡猾”，当微处理器停顿以后就无法被检测到^[18]。这时就需要对程序的运行进行实时跟踪，通过分析 Bug 的产生过程找出其产生的原因。于是，Trace 的调试方法被提出，它的主要思想是：在调试时用户首先设置监视点，监视点一旦被触发，内核的当前状态信息(如 PC、

Data、Address^[19]等)经过处理后被写回特有缓存,调试器首先读取缓存信息,然后解析这些信息从而得知监视点触发后内核的运行状态,进而达到调试目的。

该方案的技术特点如下:与 OCD 技术一样,在调试协议与通信协议之间需要增加一个调试协议转换器进行信号转换;在目标系统上集成触发点产生单元,用于设置跟踪的起始点或结束点,增加跟踪存储单元用来存储跟踪信息,增加跟踪控制单元用来控制触发点的设置与触发,选择和压缩存储信息等;调试主机可以在微处理器实时运行过程中查看触发点的调试反馈。

2.3.5 多核调试

随着 SoC 中集成度的提高,在芯片上集成多个处理器核提高性能或集成异质的核来处理不同的事务已成为发展的趋势。随之而来的问题就是如何对多个处理器核进行调试,要做到用较少的管脚开销实现对多个核的可观察性和可控性,同时要尽可能地重用原来处理器核上的调试设计。多核调试已成为可调试设计一个新的问题。目前,能够支持同时调试多处理器的调试器非常少。当系统中包含多种不同性质的处理器时,调试过程变得更加困难。多 CPU 的调试的另一个挑战是,多处理器之间的互动可能产生难以发现和解决的问题,除非调试工具可以在所有内核上同时执行^[20]。而且,嵌入式产品中多媒体应用的增加、使用数字信号处理功能的增加,意味着软件工程师更多地关心对于多核芯片的编程和调试问题。

在多核调试方面,ARM 公司提出了 CoreSight 系统调试解决方案,该调试解决方案提供了一个简单但是功能强大的交叉触发机制,能够在内核间传送调试信息,在一个内核中发生的停止或者触发会马上发送信号到其它的内核或者智能外设,使得它们能够同步停止或者发生触发^[21]。更进一步地,在某些应用中,ARM 处理器发送一个控制请求给 DSP, DSP 却返回一个随机的状态数据,这种状况会导致控制请求得不到正确的处理。这样,开发人员除需要看到内核的上下文环境外,还要知道 DSP 的状态。而 CoreSight 提供的有关联的跟踪分析功能,能够帮助找到原因。对于握手、mailbox 锁问题、回调算法、活锁、死锁和很多其它的处理器间的情况,追踪分析功能都能够很好地提供信息,而传统的停住内核的方式却不能够。ARM 公司的 CoreSight 多核追踪系统提供了同步的对于多个源的追踪。这样的追踪对于每个时钟周期都是精确的,并且各个触发点都是相互关联的。

多核调试需要在提供对单核调试访问的基础上, 提供对多个核同时调试的能力。TRACE 调试技术也逐渐地被应用到多核调试上, 因为它非常适合用于系统性能分析, 并观察到各种复杂情况下核之间交互的情况。TRACE 与传统的控制程序运行 (run-control) 方式相结合是多核调试发展的趋势^[22]。

2.4 Eclipse 平台相关技术

2.4.1 Eclipse 简介

2001 年 11 月 Eclipse 诞生, Eclipse^[23]最初是 IBM 的产品, 后来 IBM 把它无偿捐献给了开源组织 Eclipse.org。Eclipse 出色而具独创性的平台特性及开放源代码的特点, 吸引了众多大公司加入到 Eclipse 平台的发展上来, 这些大公司有 IBM、Borland、Oracle、Sybase 和 RedHat 等。开源软件的蓬勃发展, 更是强化了 Eclipse 作为开发工具方面的优势, 如 Spring、Struts 等开源框架都会附带提供各自的 Eclipse 插件工具。

Eclipse 是一个开放源代码的、基于 Java 的可扩展开发平台^[24]。就其本身而言, 它只是一个框架和一组服务, 用于通过插件组件构建开发环境。Eclipse 附带了一个标准的插件集, 包括 Java 开发工具 (Java Development Tools, JDT)。

虽然大多数用户很乐于将 Eclipse 当作 Java IDE 来使用, 但 Eclipse 的目标不仅限于此。Eclipse 还包括插件开发环境 (Plug-in Development Environment, PDE), 这个组件主要针对希望扩展 Eclipse 的软件开发人员, 因为它允许他们构建与 Eclipse 环境无缝集成的工具。由于 Eclipse 中的每样东西都是插件, 对于给 Eclipse 提供插件, 以及给用户提供一致和统一的集成开发环境而言, 所有工具开发人员都具有同等的发挥场所。

这种平等和一致性并不仅限于 Java 开发工具。尽管 Eclipse 是使用 Java 语言开发的, 但它的用途并不限于 Java 语言; 例如, 支持诸如 C/C++、COBOL 和 Eiffel 等编程语言的插件已经可用, 或预计会推出。Eclipse 框架还可用来作为与软件开发无关的其他应用程序类型的基础, 比如内容管理系统。

2.4.2 Eclipse 平台体系结构

Eclipse 平台是一个具有一组强大服务的框架, 这些服务支持插件, 比如 JDT 和插件开发环境 (PDE)。它由几个主要的部分^[25]构成: 平台运行库、工作区、

工作台、团队支持和帮助。Eclipse 平台的主要角色是为工具提供商提供一套使用和遵循的机制,使各种工具能够无缝地集成。这些机制体现在明确定义的 API、接口、类和方法。平台还提供一些有用的构建模块和框架,以方便新工具的开发。Eclipse 平台体系结构如图 2-8 所示。

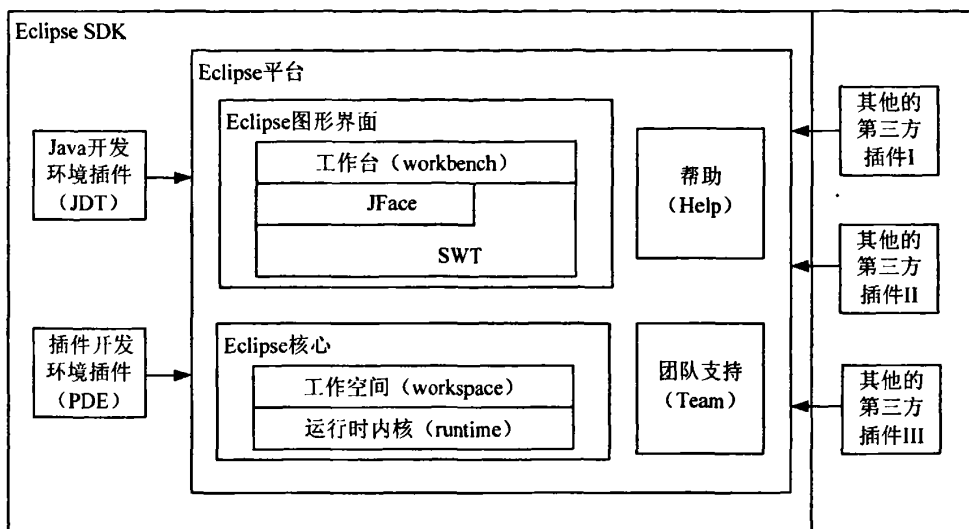


图 2-8 Eclipse 平台体系结构

平台运行库 (runtime) 是内核,它在启动时检查已安装了哪些插件,并创建关于它们的注册表信息。为降低启动时间和资源使用,它在实际需要任何插件时才加载该插件。除了内核外,其他每样东西都是作为插件来实现的。

工作区 (workspace) 是负责管理用户资源的插件。这包括用户创建的项目、那些项目中的文件,以及文件变更和其他资源。工作区还负责通知其他插件关于资源变更的信息,比如文件创建、删除或更改。

工作台 (workbench) 为 Eclipse 提供用户界面。它是使用标准窗口工具包 (SWT) 和一个更高级的 API (JFace) 来构建的;SWT 是 Java 的 Swing/AWT GUI API 的非标准替代者, JFace 则建立在 SWT 基础上,提供用户界面组件。

SWT 比 Swing 或 AWT 更紧密地映射到底层操作系统的本机图形功能,这不仅使得 SWT 更快速,而且使得 Java 程序具有更像本机应用程序的外观和感觉。使用这个新的 GUI API 可能会限制 Eclipse 工作台的可移植性,不过针对大多数流行操作系统的 SWT 移植版本已经可用。

Eclipse 对 SWT 的使用只会影响 Eclipse 自身的可移植性——使用 Eclipse 构建的任何 Java 应用程序都不会受到影响，除非它们使用 SWT 而不是使用 Swing/AWT。

团队支持（Team）组件负责提供版本控制和配置管理支持。它根据需要添加视图，以允许用户与所使用的任何版本控制系统交互。多个团队存储库的提供者可以在平台里和平相处。Eclipse 平台包含了对 CVS 存储库的支持，可以通过 pserver、ssh、extssh 协议进行访问。

帮助（Help）组件具有与 Eclipse 平台本身相当的可扩展能力。与插件向 Eclipse 添加功能相同，帮助提供一个附加的导航结构，允许工具以 HTML 文件的形式添加文档。

2.4.3 Eclipse 插件扩展机制

2.4.3.1 Eclipse 插件机制

Eclipse 的价值是它为创建可扩展的集成开发环境提供了一个开放源码平台。这个平台允许任何人构建与环境和其它工具无缝集成的工具。

工具与 Eclipse 无缝集成的关键是插件。除了小型的运行时内核之外，Eclipse 中的所有东西都是插件^[26]。从这个角度来讲，所有功能部件都是以同等的方式创建的。Workbench 和 Workspace 是 Eclipse 平台的两个必备的插件——它们提供了大多数插件使用的扩展点，如图 2-9 所示。其他的插件包括 Eclipse 的图形 API（SWT/JFace）、Java 开发环境插件（JDT）、插件开发环境（PDE）等，同时也包括第三方扩展的插件。

Eclipse 还对这些插件的协同工作提供了良好的支持，每个插件可能依赖于其他插件提供的服务；反过来，这一插件又可能提供其他插件所依赖的服务。这种模块设计让自己成为离散的功能块，更容易地重用于构建应用程序。

插件的依赖项和服务都在名为 plugin.xml 的插件清单文件中声明。在这个文件中，您要声明的是：从哪里接收其他插件的添加项，以及您的插件要添加到其他插件的什么位置；您是要集成到 Eclipse 平台自身还是其他基于 Eclipse 的工具。这个结构方便于插件代码的延迟加载，从而减少 Eclipse 的启动时间和内存占用。Eclipse 启动后，插件加载器扫描 plugin.xml 文件，寻找每个插件，并构建一个包含这些信息的结构。当真正要用到某个插件时，该插件才会被调入内

存实例化；当该插件不再被使用时，它就会在适当的时候被清除出内存。因此，即使装了一大堆插件在 Eclipse 中，也不必担心某些不常用的插件白白消耗内存。

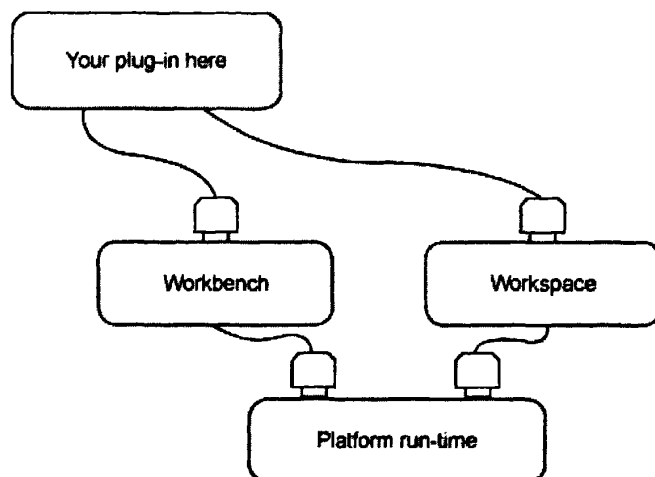


图 2-9 Eclipse Workbench 和 Workspace: 必备的插件支持

最初，Eclipse 的插件机制是自己设计的，在 OSGi 成为动态插件机制方面的标准后，Eclipse 从 3.0 版开始按 OSGi 标准重新实现了自己的插件机制，并形成了一个独立的项目 Equinox，此项目是 OSGi 标准的一个实现。OSGi 是一套开放标准，Eclipse 引入 OSGi 主要是考虑网络范围内的互操作性。一个 OSGi 模块最重要的就是 Bundle 和 Service，我们可以认为 Bundle 是一种插件管理器，主要是通过 BundleActivator 管理模块的生命周期，而 service 则是这个模块可暴露对外的服务对象。这里体现了 OSGi 和传统的 Plug-in Framework 不同的一个地方，管理和静态结构分开，每个 Bundle 拥有自己的 ClassLoader 以及 context，通过 context 可进行服务的注册、卸载等，这些操作都会通过事件机制广播给相应的其他的 Bundle；一般来说都为通过在 Bundle 中编写初始需要注册的服务的方法来完成 Bundle 可供外部使用的服务的暴露功能。

插件扩展机制是 Eclipse 最突出的特点和优势，它使 Eclipse 提升到了一个平台的高度。我们可以利用 Eclipse 的插件开发环境 PDE 来开发自己的 Eclipse 插件，随己所需地扩展 Eclipse 的功能。这样的插件形式是多种多样的，它可以是一种编程工具（如 C/C++、JSP、PHP），也可以是 J2EE 开发模块（如 MyEclipse 和 Lomboz），还有建模（如 Together for Eclipse）、数据库、GUI（如 Draw 2D）

等开发插件，甚至是一个桌面应用系统，插件机制使得 Eclipse 具有了无限扩展的可能。

2.4.3.2 扩展点

扩展 (Extension) 是 Eclipse 中一个关键的机制，plug-in 利用扩展向 Eclipse 平台添加新功能。但是扩展不能随意地创建，必须按照扩展点 (extension point) 定义的规范进行明确的声明，Eclipse 才能认出这些扩展。在 Eclipse 插件体系结构中，最重要的概念就是扩展点，Eclipse 框架的灵活性主要来源于其扩展点。所谓扩展点，就是系统定义出来可以让你扩展的地方，可以认为是一些扩展的契约；而扩展，则是你对这些扩展点的实现。我们不仅可以使用 Eclipse 提供的众多现成的扩展点，而且还可以定义新的扩展点，并在该扩展点上进行扩展。

Eclipse 通过定义扩展点可以进行扩展，同时，每个插件均可以定义自己的扩展点。任何 Eclipse 插件定义的扩展点都能够被其它插件使用，反之，任何 Eclipse 插件也可以遵从其它插件定义的扩展点。

每个扩展点都有唯一的标识符，它由插件的唯一标识符、句点和仅包含字母、数字字符和下划线的简单标识符组成。声明扩展点时，仅使用简单标识符。声明扩展点的扩展时，则使用扩展点的完全标识符。每个扩展点可以具有一个模式 (schema)，用于定义如何使用扩展点。Eclipse PDE 使用该方案进行扩展的基本自动验证，以及为扩展点自动生成类似 Javadoc 的文档。模式是一个 XML 格式文件，一般具有名称 <extension-point-id>.exsd，位于插件安装目录下的模式子目录中。

2.5 本章小节

本章主要研究了与课题相关的各种技术。首先介绍了 CoreSight 跟踪调试技术及其调试体系架构。接着分析比较了几种常用的嵌入式调试技术，包括软件监控、在线仿真、片上调试、TRACE 调试、多核调试等。最后介绍了 Eclipse 平台相关技术，包括 Eclipse 平台体系架构及其插件扩展机制。

第 3 章 Event Viewer 系统简介

3.1 需求分析

Event Viewer 系统^[27]作为 RVDS 集成开发环境中调试模块的组成部分，其基本功能是从原始数据源接受原始数据，经过数个阶段的解析处理，产生适当的数据包，并通过显示模块在 Eclipse 中以可视化的形式重构目标板程序的运行状况。系统的数据处理流程如图 3-1 所示。数据流在到达宿主机是实时的。如果数据传输率过高，超过系统的处理能力，数据将被转储到磁盘上，并可以在以后以非实时的形式在系统中回放。

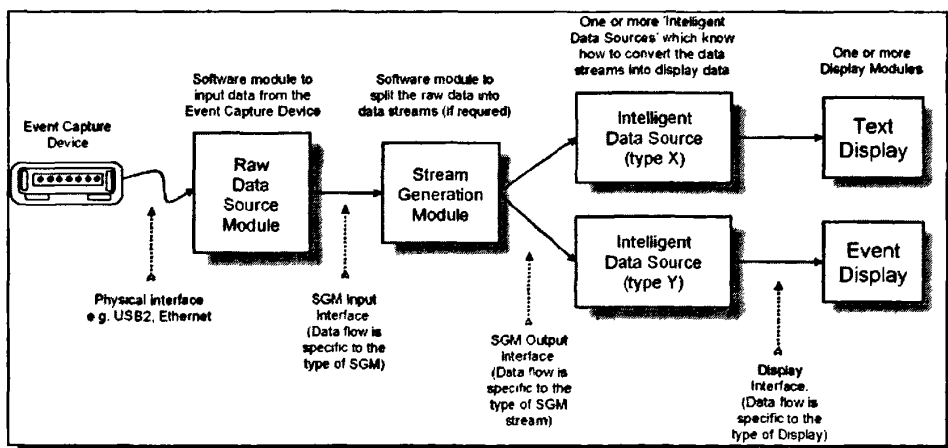


图 3-1 数据处理流程

系统处理的数据是一种 push 形式的数据流。当原始数据源接受到启动命令时，它将产生一个新的线程（独立于控制线程）来接受原始数据流，原始数据流实际来自目标板上的 trace 跟踪数据流，经由捕获设备（如 RVI、RVT 等）捕获后再通过 USB、以太网等传输给宿主机的。当原始数据被接受后，再以 push 的形式被送到下一个模块（即流生成器），流生成器将把数据流拆分成数据包，然后交给智能数据源解析，在那里数据包将被解析成相应的记录，然后送到相应的显示模块显示。每个模块都应该将接受和处理数据的工作放到两个独立的

线程里面去做。

Event Viewer 系统的主要需求如下：

- 1) 解析原始数据源接受的 trace 跟踪数据流，恢复程序运行信息以供调试和性能分析，此即系统的基本功能。
- 2) 系统允许第三方产品提供商产生并处理自定义的 trace 跟踪数据。此设计要求将整个数据处理流程划分成功能细分、各自独立的子模块（即原始数据源、流生成器、智能数据源三个子模块），每个模块提供公共的接口以方便第三方进行扩展。
- 3) 提供一套标准的显示模块。不论是 Arm Core 原始的 trace 跟踪数据流还是第三方产品提供商产生的自定义 trace 跟踪数据，解析后都使用同样的显示模块来显示。
- 4) 每个模块都有相应的配置功能。通过配置，原始数据源模块应该知道需要连接的物理设备类型（比如 RVL,RVT,DISK），智能数据源应该了解到所需要解析的数据类型（文本或是事件等等），显示模块需要得到一些用户的个人化信息（颜色，线条风格等等）。
- 5) 高级配置主要涉及到模块之间的关系，每个模块应该可以描述此模块接受的数据类型和产生的数据类型，整个配置从选择原始数据源开始，一旦用户选择了一个特定的原始数据源以及最终的显示类型，那么所有能解析该原始数据源所产生的数据类型并产生所匹配的显示类型的模块（智能数据源）将被列出来，用户来选择需要的下个模块来完成配置。

3.2 概要设计

3.2.1 系统开发环境

本系统的开发基于以下环境（运行环境、测试环境均与开发环境相同）：

- Windows XP sp2
- Eclipse3.3
- JDK1.6.0_03

3.2.2 系统总体结构

按照以上的分析，Event Viewer 系统一共划分为以下六个子系统，系统总体

结构^[28]如图 3-2 所示。

- 1) Configuration, 负责实现系统的进入点, 向导及各种配置对话框, 配置信息的持久化, 及与用户工程的交互。
- 2) RDS (Raw Data Source), 原始数据源, 负责从目标硬件读取数据, 并传送给下一子系统。
- 3) SG (Stream Generation), 流产生器, 负责将原始数据解析, 并将解析后的数据传送给下一子系统。
- 4) IDS (Intelligent Data Source), 智能数据源, 负责将SG解析后的数据翻译成最终用户可读的信息, 并将信息传递给下一子系统。
- 5) Display, 视图, 负责显示IDS翻译后的数据。
- 6) Platform, 公共架构支撑平台, 负责实现RDS、SG、IDS、Display间的接口交互。

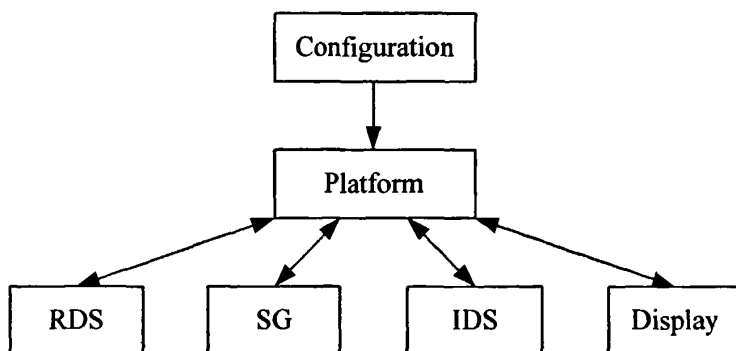


图3-2 系统总体结构

本系统是Arm RVDS集成开发环境的一部分, 实现此系统的目的在于向使用Arm Core的最终用户提供采集、解析并显示Arm Core trace data的功能, 向第三方产品提供商提供扩展接口的功能, 以方便第三方产品提供商产生自定义的trace data。这里所说的第三方产品提供商, 是指在Arm Core的基础上进行功能扩展的公司或个人。最终用户是指直接使用Arm Core或者在第三方产品提供商提供的产品基础上进行软件开发的相关人员。

其中, RDS模块负责trace跟踪数据的采集, SG和IDS模块负责对采集到的跟踪数据进行解析, Display模块负责对解析后的结果进行显示。RDS、SG、IDS和Display四个模块构成了整个系统的主体框架, 配以Configuration模块提供系统

的入口点和其他配置，Platform模块提供一个公共的支撑平台，进行RDS、SG、IDS、Display间的数据转发及运行模式切换管理工作。

3.2.3 子系统间的协作

子系统间的协作如图3-3所示，现分步骤说明如下：

- 1) 用户通过子系统Configuration提供入口点开始使用程序，然后通过对话框配置RDS，SG，IDS，Display信息。
- 2) 用户配置完成后，Configuration将数据发送给Platform，Platform通知RDS开始读取数据，RDS读取一定数据后传回给Platform。
- 3) Platform将得到的原始数据传送给SG，然后SG将自己解析后的数据传回Platform。
- 4) Platform根据不同的Channel ID将数据传送给对应的IDS，然后IDS将自己翻译后数据传回给Platform。
- 5) Platform将数据传送给与每个IDS对应的Display，完成显示。

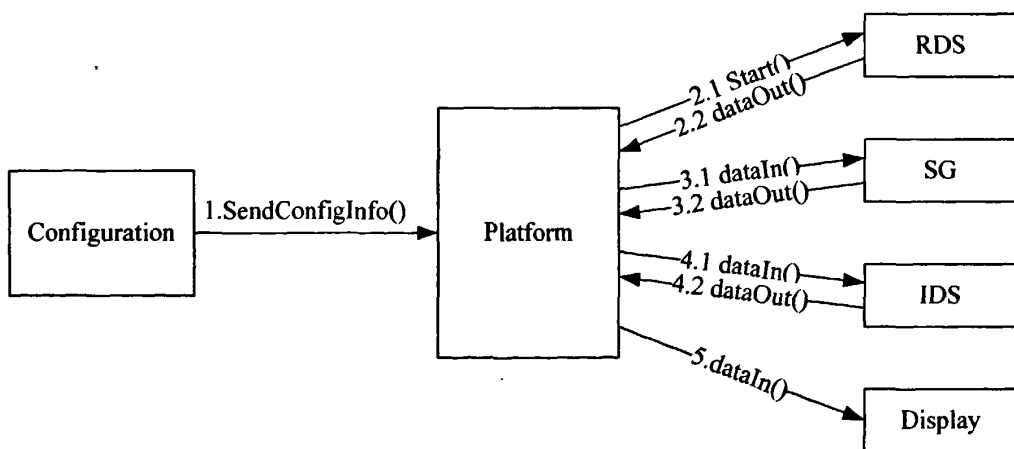


图3-3 子系统间协作

3.2.4 子系统划分的原则

子系统的划分主要体现了以下几个方面的原则：

- 保证了整个系统的低耦合

- RDS、SG、IDS、Display间实现了完全分割，可以完全不用考虑其他部分的存在，第三方产品提供商可以随意的提供自己的扩展，或者使用别人的扩展
- Platform承担并统一了所有接口间的交互工作，易于管理，并且易于被第三方产品提供商使用
- 在不改变各子系统外部行为的情况下，各子系统内部可以实现任意的重构

3.3 第三方扩展设计

在系统中，Arm CoreSight跟踪数据流存在多种，同时系统又允许第三方产品提供商产生并处理自定义的trace跟踪数据，这些都导致跟踪数据流的格式是不确定的，而对于不同格式的数据流，在采集、解析过程中是存在差异的。因此，RDS、SG、IDS三个部分均需允许第三方扩展，即第三方可以编写自己的RDS、SG、IDS然后加入到本系统，而所有的显示工作必须通过本系统提供的标准Display实现，第三方不能扩展Display。本次扩展的功能较好的契合了Eclipse扩展点机制。本系统采用Eclipse扩展点机制来实现第三方扩展功能，第三方的扩展必须以Eclipse插件的形式提供。按照自定义扩展点的基本步骤即可在插件项目中添加RDS、SG、IDS三个部分的扩展点。

3.3.1 扩展点

本系统共实现三个扩展点，RDS、SG、IDS，每个扩展点均应配置以下元素：

- 1) **class**: JAVA类，用来实现扩展接口，必须配置，不可选
- 2) **id**: 字符串，第三方应保证此id的唯一性，系统内部将按此id区分不同的扩展，必须配置，不可选
- 3) **name**: 字符串，用来指明扩展的名字，必须配置，不可选
- 4) **description**: 字符串，用来简单说明当前扩展的意义，可选

3.3.2 扩展接口

扩展接口是指第三方在扩展某一个具体的扩展点Java类必须实现的接口，这是扩展能够成功接入本系统的依据。每一个扩展应实现的接口说明如下：

1) RDS

Control接口，负责从开始或者停止从目标硬件读取数据

DataOut接口，负责将读取得数据送出

ModeListener接口，负责接收模式切换消息

RecordInfoOut接口，负责将Record模式下存盘信息输出

2) SG

DataIn接口，负责接收原始数据

DataOut接口，负责输出解析后的数据

3) IDS

DataIn接口，负责接收SG解析后的数据

DataOut接口，负责输出翻译后的数据

3.3.3 扩展角度下的总体结构

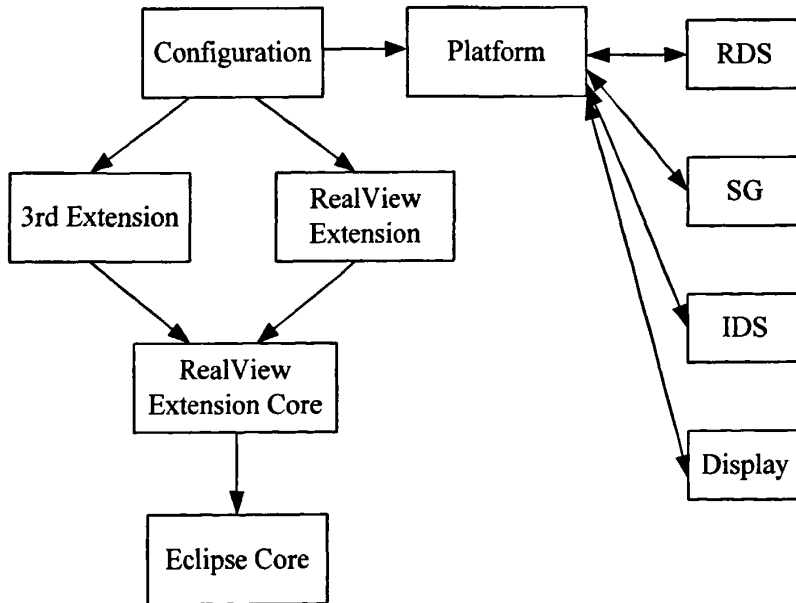


图3-4 扩展角度下的系统结构

在提供扩展点及扩展接口后，借鉴Eclipse开发的公平竞赛法则（Fair Play Rule）：“所有使用者应遵循同样的游戏规则，包括我自己”，于是本系统内部

的扩展也必须使用上面的扩展点和扩展接口来实现。本系统内部必须实现的扩展包括：

RDS: RVT2.Stream、RVI.SWO、RVI.DCC

SG: TPIU、DCC

IDS: ITM.SIP.Text、ITM.SIP.Events、ITM.HSP.DATA、DCC.Events

在本系统内部的扩展也采用扩展点之后，系统的结构变化如图3-4所示。其中RealView Extension Core由扩展点和扩展接口组成。Configuration子系统通过搜索扩展点的方式得到当前每一个扩展点的所有扩展。

3.4 本章小节

本章从系统的整体需求出发，首先，简单介绍了Event Viewer系统的总体结构及其子模块的划分；在此基础上，分析了各子系统间的交互协作及其划分原则；最后，讨论了系统的第三方扩展设计。

在Event Viewer系统的开发过程中，本人主要负责解析和显示两个部分的设计及编码，具体到子系统，即SG、IDS和Display三个子模块。本文接下来将分两个章节详细讨论CoreSight跟踪数据的解析及其显示。

第 4 章 CoreSight 跟踪数据的解析

4.1 CoreSight 跟踪数据的解析流程

用户通过目标板上 TPIU、SWO、JTAG 等端口获取的 CoreSight 跟踪数据，是一种字节流形式的数据，数据流经由 RVT、RVI 等设备捕获后，通过 USB、以太网等传输给调试主机。CoreSight 跟踪数据是由目标板上 CoreSight 调试结构内各种源部件（HTM、ETM、ITM 等）产生并发送的，这些数据包括内存单元读写、中断异常的发生与处理、操作系统任务之间的触发关系与运行过程等信息。当我们在调试主机上获得这些数据时，需要先进行一系列的解析处理，将数据流还原成数据包，并最终以可视化的形式将其包含的信息展现给用户，以供对程序进行调试和性能分析。CoreSight 跟踪数据解析流程如图 4-1 所示。

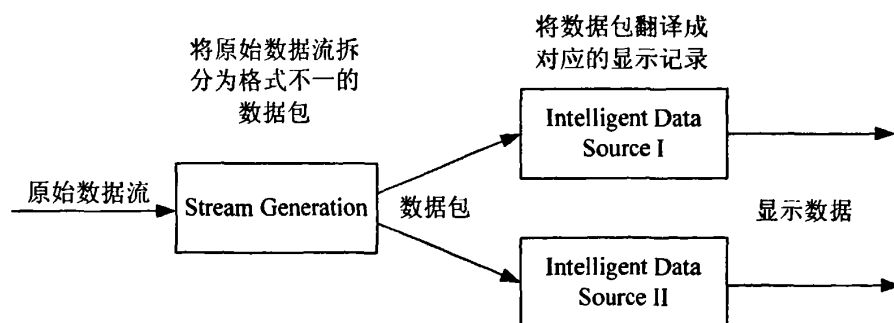


图 4-1 CoreSight 跟踪数据解析流程

在本系统中，解析流程包含两个模块：流生成器模块和智能数据源模块，他们分别完成各自不同的功能。流生成器负责将原始数据流拆分成格式不一的数据包，即将字节流形式的数据还原成源部件产生的包格式数据；智能数据源负责将流生成器产生的数据包按照指定的解析协议翻译成显示记录，并送至显示模块显示。

流生成器和智能数据源之间是一对多的关系。流生成器产生的格式不一的数据包需要由不同的智能数据源来进行翻译；同时每种数据包可能包含文本、事件、图表等不同的显示格式，这也需要对应的智能数据源。流生成器和智能数

据源之间通过 Channel ID 来确认发送通道，以保证流生成器产生的数据包送往正确的智能数据源。

4.2 解析模块的详细设计

4.2.1 流生成器

流生成器（Stream Generation，简称为 SG）的主要功能是将 RDS 产生的数据进行转换，并发送转换后的数据给子系统 Platform。因此 SG 主要是由一些数据处理和特定格式解析器组成，主要包括：

- 1) TPIU 数据预处理，负责从原始数据中分离出 TPIU 帧。
- 2) TPIU 解析器，负责 TPIU 帧的解析，并输出通道 ID 和 ITM 数据包。
- 3) ITM 解析器，负责 ITM 数据包的解析，并输出每一个 ITM 数据包的 Payload。
- 4) DCC 解析器，负责 DCC 数据的解析。

比较明显的 SG 内部可以划分为两个模块：

- TPIUSGModule，负责 TPIU 数据的处理
- DCCSGModule，负责 DCC 数据的处理

这两个模块之间是并列关系，无任何交叉。

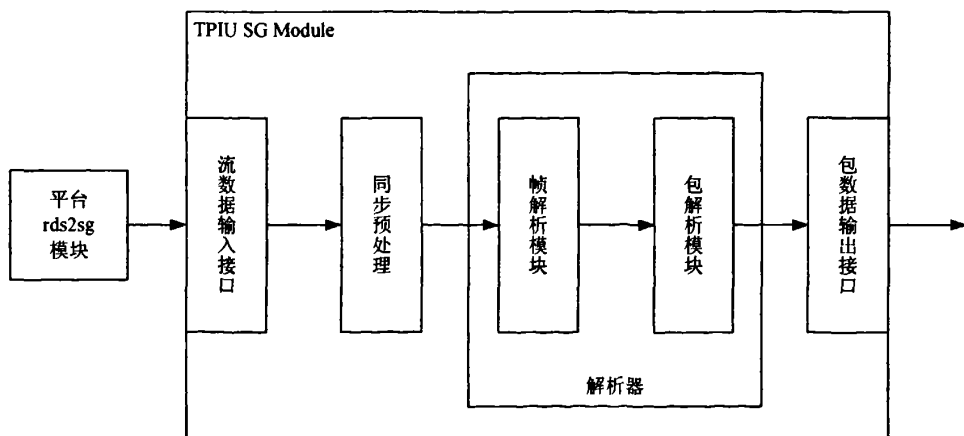


图 4-2 TPIUSG 模块

TPIUSGModule 中待解析处理的原始数据流实际来自目标板 TPIU 端口，这些数据在传送到片外前在 TPIU 汇集点按照 TPIU 帧的格式进行了组装，它包含有片内各种跟踪源（ETM、HTM、ITM 等）获取的信息，而本系统只需获取 ITM 跟踪源数据包含的信息。因此，在 TPIUSGModule 中首先需要对原始数据流进行同步预处理获得完整的帧格式数据；然后对帧格式数据进行解析，将各种跟踪源数据进行分离，以 Channel ID 加以区分；最后根据 Channel ID 解析 ITM 跟踪源数据，得到每一个 ITM 数据包的 Payload。TPIUSGModule 详细设计如图 4-2 所示，包括有同步预处理、帧解析、包解析三个处理模块，其中帧解析、包解析构成解析器主体。

DCCSGModule 中待解析处理的原始数据流来自目标板 JTAG 端口，其解析流程较为简单，模块详细设计如图 4-3 所示，主要包括一个 DCC 解析器。

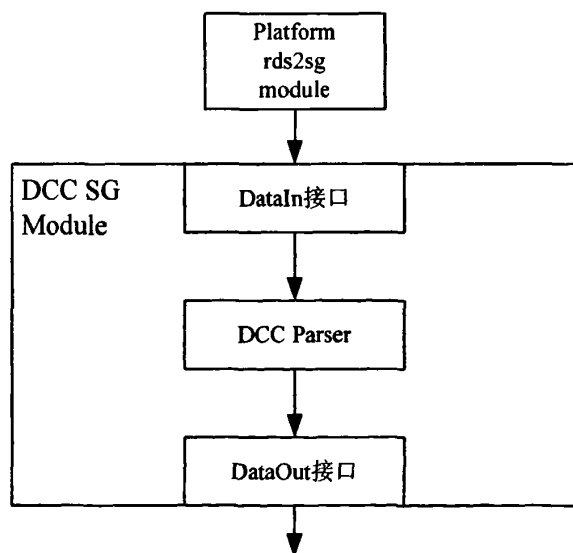


图 4-3 DCCSG 模块

SG 模块的 DataIn, DataOut 接口如下所示：

```

public interface ISGDataIn {
    public byte inputData();
    public byte[] inputDatas( );
}

```

```
public interface ISGDataOut {
    public void outputData( int id, IPacketType type, byte[] data );
}
```

4.2.2 智能数据源

智能数据源 (Intelligent Data Source, 简称为 IDS) 的主要功能是将 SG 产生的 Payload 翻译成具体的可以被显示的信息, 如事件名称, 事件发生时间, 事件停止时间等, 并将信息传回给子系统 Platform。

本系统中主要包含以下四个 IDS:

- 1) ITM.SIP.Text, 负责 ITM 数据中的 SIP 包中的 Text 格式 Payload 的翻译。
- 2) ITM.SIP.Events, 负责 ITM 数据中的 SIP 包中 Event 格式 Payload 的翻译。
- 3) ITM.HSP.Data, 负责 ITM 数据中的 HSP 包中 Data 格式 Payload 的翻译。
- 4) DCC.Events, 负责 DCC 数据中的 Event 格式数据的翻译。

比较明显的上面四个 IDS 内部都分别对应一个模块, 这四个模块是并列关系, 互不交叉, 互不影响。

IDSModule 的详细设计如图 4-4 所示。虽然每个 IDS 待解析的数据都不一样, 但其解析流程是一致的。模块中的主要组成部分是 IDS 解析器, 对于不同的 IDS, 提供各自的解析器, 按照指定的解析协议对 Payload 进行翻译, 最终得到对应的显示记录。

IDS 模块的 DataIn, DataOut 接口如下所示, 其中 DataOut 接口包括三个, 分别对应 text、event、analog 显示数据的输出。

```
public interface IIDSDDataIn {
    public IPayloadWithType inputData( );
}

public interface IIDSTextDataOut {
    public void writeTextData( ITextDisplayData data );
}

public interface IIDSEventDataOut {
    public void writeEventData( IEventDisplayData data );
}

public interface IIDSanalogDataOut {
```

```
public void writeAnalogData( IAnalogDisplayData data );
}
```

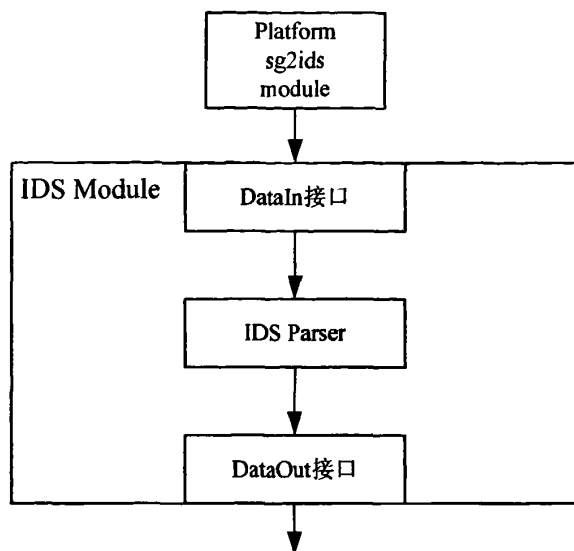


图 4-4 IDS 模块

4.3 基于 ITM 数据流的解析实现

本节将以 ITM 数据流为例，选取流生器 TPIUSG 模块和智能数据源 ITM.SIP.Text 模块来详细介绍 CoreSight 跟踪数据的解析实现。

4.3.1 ITM 包格式

来自于 TPIU 口的 ITM 数据包括以下数据：TPIU.ITM = ITM.IDLE | ITM.SYNC | ITM.OVERFLOW | ITM.TIMESTAMP | ITM.SIP | ITM.HSP。分别是空闲包，同步包，溢出包，时间标记信息包，软件仪器跟踪包，硬件源包^[29]。其中硬件源包是由 DWT 源发出，也称为数据检测点跟踪包，可细分为事件计数器回绕包（Event Counter Wrapping），异常跟踪包（Exception Tracing），PC 取样包（PC sampling）和数据跟踪包（Data Tracing）。

每个包的大小从 1 个字节到 5 个字节不等，其中有一个头字节，其余的为

payload 字节。同步包例外，有 6 个字节。表 4-1 简单列出了各个包的格式：

表 4-1 ITM 包格式

Description	Value	Payload	Category	Remarks
Idle	00000000	None	Wire	Only used for TPIU modes that emit a continuous clock.
Sync	S0000000	None	Synchronization	{47{1'b0}} followed by 1'b1 (Matches ETM protocol)
Overflow	01110000	0	Protocol	Overflow
Time Stamp	CDDD0000	0 to 4 bytes	Protocol	D = data (!=000) – time C = continuation
Extension	CDDDIS00	0 to 4 bytes	Protocol	S = source (ITM / DWT) D = data – page info C = continuation
Reserved	Cxxx0100	0 to 4 bytes	Reserved	C = continuation
Instrumentation	AAAAA0SS	1, 2 or 4 bytes	Software Source (Application)	SS = size (!=00) of payload A = SW Source Address
Hardware Source	AAAAA1SS	1, 2 or 4 bytes	Hardware Source (Diagnostics)	SS = size (!=00) of payload A = HW packet type discriminator

4.3.1.1 空闲包 (Idle)

空闲包由一个字节组成，为 0x00。

4.3.1.2 同步包 (Sync)

当同步计数器到 0 时，由时间标记信息包发出同步包，同步包的作用在于同步 SWIT/DWT 包与时钟计数器信息。同步包由六个字节组成，其中前面五个字节均为 0x00，第六个字节为 0x80。同步包的优先级最高。

4.3.1.3 溢出包 (Overflow)

SWIT 源，DWT 源和 TimeStamp 源都可以发出溢出包。条件为：当向激励寄存器中写入 SWIT/DWT 数据时，也就是向 FIFO 发包时，FIFO 满了，此时，SWIT/DWT 源会发出溢出包；当 TimeStamp 计数器计数达到 2097151 时，TimeStamp 源会发出一溢出包。溢出包仅包含一个字节，为 0x70。溢出包的优先级仅比同步包低。

4.3.1.4 时间标记信息包（Timestamp）

时间标记信息包包含有一个头字节和若干 payload 字节（0 个、1 个、2 个或 4 个），头字节的低四位均为 0，payload 的长度由标志位 C 来决定，时间标记信息包格式如图 4-5 所示。

	bit7						bit0
byte 1	C	TC[2]	TC[1]	TC[0]	0	0	0
byte 2	C	TS[6]	TS[5]	TS[4]	TS[3]	TS[2]	TS[1]
byte 3	C	TS[13]	TS[12]	TS[11]	TS[10]	TS[9]	TS[8]
byte 4	C	TS[20]	TS[19]	TS[18]	TS[17]	TS[16]	TS[15]
byte 5	0	TS[27]	TS[26]	TS[25]	TS[24]	TS[23]	TS[22]

图 4-5 时间标记信息包格式

C：称为持续位，即：C 为 1 时，此字节后接一字节；C 为 0 时，此字节后包结束。

TS[27:0]：增量时间标记信息，用于记录发包当前计数器时钟信息，即记录当前相对时间。

TC[2:0]：时间标记信息控制位，当时间标记信息包为单字节包或多字节包时，其意义各有不同，表 4-2、4-3 列出了它们的意义。

表 4-2 单字节包时（头字节 C=0）

时间标记信息控制位	意义
000	空闲
001-110	时间标记信息包向 ITM/DWT 数据包发出同步信息
111	ITM/DWT 数据包溢出

表 4-3 多字节包时

时间标记信息控制位	意义
000	同步
001~011	保留
100	时间标记信息包向数据包发出同步信息
101	时间标记信息包发送延迟
110	ITM/DWT 数据包发送延迟
111	时间标记信息包及 ITM/DWT 数据包发送延迟

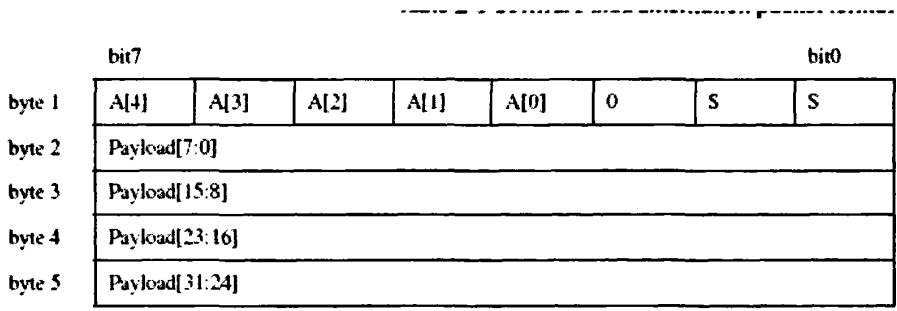
注：1)当时间标记信息包准备发送而 FIFO 未准备好接受此包时，使用 Timestamp 延迟标记（100）；而当时间标记信息包由于更高优先级的事务而导致发送延迟时，Timestamp 将继续计时，并且不使用 Timestamp 延迟标记。

2)当 ITM/DWT 数据包准备发送而 FIFO 未准备好接受些包或由于更高优先级的事务需要处理时，使用 packet 延迟标记（110）。

3)时间标记信息包的优先级最低。

4.3.1.5 软件仪器跟踪包（SIP）

软件仪器跟踪包由一个头字节和若干 payload 字节（1 个、2 个或 4 个）组成，头字节的 bit2 为 0，payload 的长度由头字节低两位来决定，软件仪器跟踪包格式如图 4-6 所示。



SS = Payload Size in bits

图 4-6 软件仪器跟踪包格式

SS 位用于表示 SIP 包 Payload 的大小，表 4-4 列出了 SS 位的标志信息。

表 4-4 SS 位的标志信息

SS 位	标志信息
01	payload 为 8 位，SIP 包大小为两字节
10	payload 为 16 位，SIP 包大小为三字节
11	payload 为 32 位，SIP 包大小为五字节
00	不可用

A[4:0]: 这五位用来指示发出 SIP 包的激励端口号，即激励寄存器中 32 位

中的哪一位 (0-31)。

Payload[{31:0}{15:0}{7:0}]: 应用程序或操作系统向激励寄存器中写入的数据。

4.3.1.6 硬件源包 (HSP)

硬件源包与软件仪器跟踪包格式上类似, 区别在于头字节的 bit2, 硬件源包为 1, 软件仪器跟踪包为 0。

4.3.1.7 扩展包 (Extension Packets)

扩展包由头字节和若干 payload 字节 (0 个、1 个、2 个或 3 个) 组成, 头字节的低两位均为 0, 低四位为 1。扩展包格式如图 4-7 所示。

	bit7				bit0			
byte 1	C	EX[2]	EX[1]	EX[0]	1	SH	0	0
byte 2	C	EX[9]	EX[8]	EX[7]	EX[6]	EX[5]	EX[4]	EX[3]
byte 3	C	EX[16]	EX[15]	EX[14]	EX[13]	EX[12]	EX[11]	EX[10]
byte 4	C	EX[23]	EX[22]	EX[21]	EX[20]	EX[19]	EX[18]	EX[17]
byte 5	EX[31]	EX[30]	EX[29]	EX[28]	EX[27]	EX[26]	EX[25]	EX[24]

图 4-7 扩展包格式

C: 称为持续位, 即: C 为 1 时, 此字节后接一字节; C 为 0 时, 此字节后包结束。

EX[N:0]: 扩展信息。

SH: 0 代表软件源, 1 代表硬件源。

4.3.1.8 保留编码

保留编码由五字节组成, 头字节低四位为 0x04, 保留编码格式如图 4-8 所示。

	bit7				bit0			
byte 1	C	E2	E1	E0	0	1	0	0
byte 2	C	x	x	x	x	x	x	x
byte 3	C	x	x	x	x	x	x	x
byte 4	C	x	x	x	x	x	x	x
byte 5	x	x	x	x	x	x	x	x

图 4-8 保留编码格式

4.3.2 TPIUSG 模块的实现

系统启动时，TPIUSG 以创建线程的方式加入到平台中，接受平台的统一管理。TPIUSG 模块的解析流程如图 4-9 所示，包括如下步骤：

- 1) TPIUSG 模块通过 ISGDataIn 接口从平台获取原始数据流；
- 2) 调用预处理器进行帧同步得到帧格式数据；
- 3) 帧解析模块 TPIUParser 解析帧格式数据得到 Channel ID 和对应的包数据，并根据 ID 送入不同的缓冲区；
- 4) 获取缓冲区内 ITM 包数据，调用包解析模块 ITMParser 进行解析，获得每一个 ITM 数据包的 payload；
- 5) TPIUSG 模块通过 ISGDataOut 接口将每一个 ITM 数据包的 payload 送至平台。

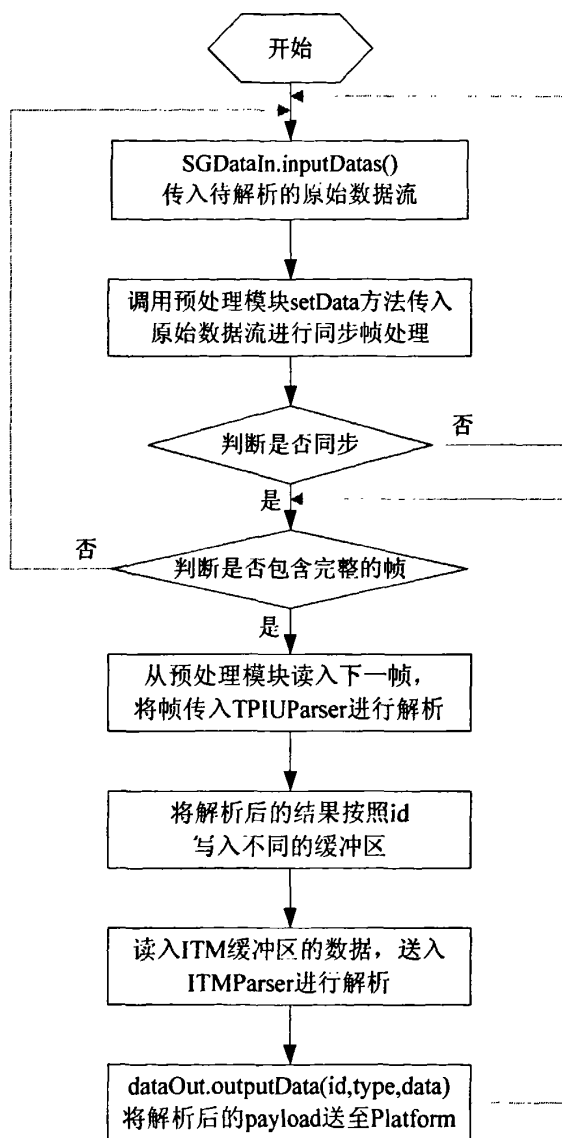


图 4-9 TPIUSG 模块解析流程

4.3.3 ITM.SIP.Text 模块的实现

TPIUSG 模块解析生成的 payload 按照 Channel ID 送入平台不同的缓冲区，IDS 模块通过 IIDSDatIn 接口从平台获取对应 ID 缓冲区内的 payload 进行解析。ITM.SIP.Text 同样以创建线程的方式加入到平台中，接受平台的统一管理。

ITM.SIP.Text 模块的解析流程如图 4-10 所示，包括如下步骤：

- 1) ITM.SIP.Text 通过 IIDSDataIn 接口从平台获取对应 ID 的 ITM 数据包；
- 2) 判断 ITM 包是否为 SIP 包，并将 SIP 包写入 SIPBuffer 内；
- 3) 获取 SIPBuffer 内 SIP 包数据，调用 SIPTextParser 进行解析，获得 text 显示数据；
- 4) ITM.SIP.Text 通过 IIDSTextDataOut 接口将 text 显示数据送至平台。

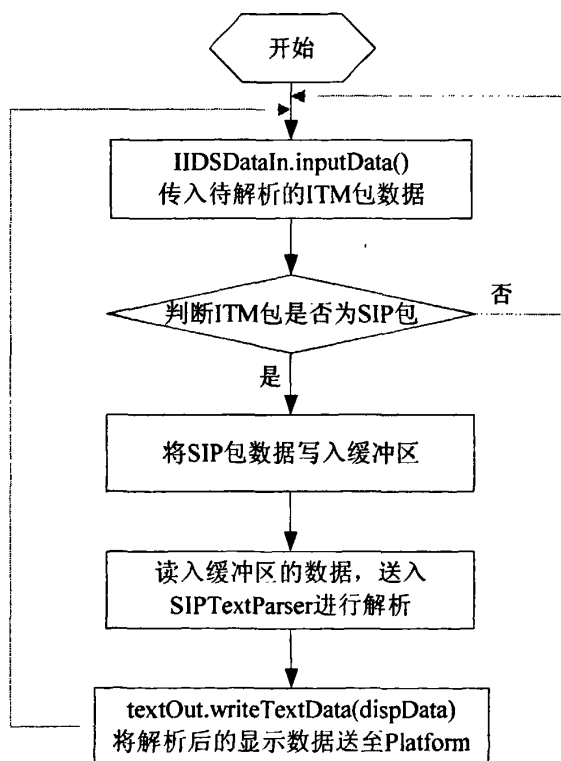


图 4-10 ITM.SIP.Text 模块解析流程

4.3.4 解析器

4.3.4.1 工厂模式简介

工厂模式^[30]专门负责将大量有共同接口的类实例化。工厂模式可以动态决定将哪一个类实例化，不必事先知道每次要实例化哪一个类。工厂模式有以下

几种形态：简单工厂模式（Simple Factory），工厂方法模式（Factory Method），抽象工厂模式（Abstract Factory）。

1) 简单工厂模式

简单工厂模式又称静态工厂方法模式，是由一个工厂类根据传入的参量决定创建出哪一种产品类的实例。简单工厂模式的示意性 UML 图如图 4-11 所示。包含两种情况：只有一个产品对象的简单工厂模式；带有一个抽象产品对象的简单工厂模式。

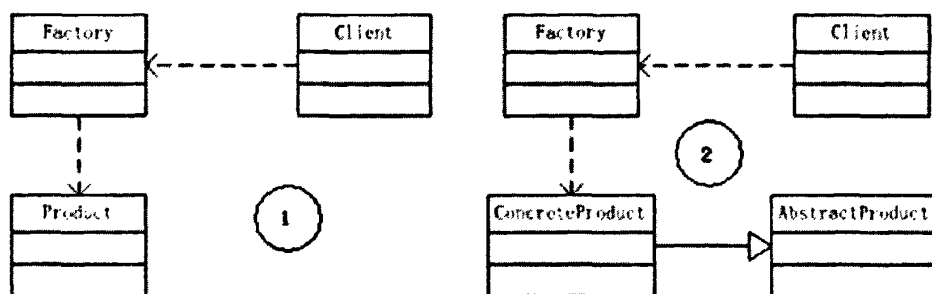


图 4-11 简单工厂模式

简单工厂模式涉及到工厂角色、抽象产品角色以及具体产品角色。

- 工厂类（Factory）角色：担任这个角色的是工厂方法模式的核心，含有与应用紧密相关的商业逻辑。工厂类在客户端的直接调用下创建产品对象，它往往由一个具体 Java 类实现。
- 抽象产品（AbstractProduct）角色：担任这个角色的类是由工厂方法模式所创建的对象之父类，或它们共同拥有的接口。抽象产品角色可以用一个 Java 接口或者 Java 抽象类实现。
- 具体产品（Concrete Product）角色：工厂方法模式所创建的任何对象都是这个角色的实例，具体产品角色由一个具体 Java 类实现。

简单工厂模式一旦添加新产品就不得不修改工厂逻辑，因此，它是不满足 OCP 原则的。OCP（Open-Closed Principle，开闭原则）：一个软件的实体应当对扩展开放，对修改关闭。即对于一个已有的软件，如果需要扩展，应当在不需修改已有代码的基础上进行。

2) 工厂方法模式

工厂方法模式是简单工厂模式的进一步抽象和推广，克服了简单工厂模式的缺点，完全满足 OCP 原则。在工厂方法模式中，核心的工厂类不再负责所有产品的创建，而是将具体创建的工作交给子类去做。这个核心工厂则变为抽象工厂角色，仅负责给出具工厂子类必须实现的接口，而不接触哪一产品创建的细节。工厂方法模式缩略类图如图 4-12 所示。

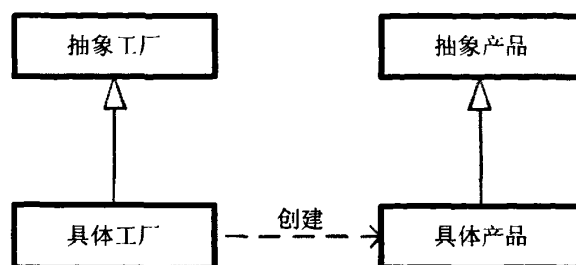


图 4-12 工厂方法模式

工厂方法模式的系统涉及到了以下角色：

- 抽象工厂角色：担任这个角色的是工厂方法模式的核心，它是与应用程序无关的，任何在模式中创建对象的工厂必须实现这个接口。
- 具体工厂角色：担任这个角色的是实现了抽象工厂接口的具体 Java 类。具体工厂角色含有与引用密切相关的逻辑，并且受到应用程序的调用以创建产品对象。
- 抽象产品角色：工厂方法所创建产品对象的超类型，也就是产品对象的共同父类或共同拥有的接口。
- 具体产品角色：这个角色实现了抽象产品角色所声明的接口。工厂方法所创建的每个具体产品对象都是某个具体产品角色的实例。

3) 抽象工厂模式

抽象工厂模式又称工具箱模式，大致和工厂方法相同。

4.3.4.2 工厂模式在解析器中的应用

工厂模式包含有三种形式，本系统最终采用简单工厂模式来创建并管理解析器，理由如下：

- 1) 工厂类含有必要的判断逻辑,可以决定在什么时候创建哪一个产品类的实例,客户端可以免除直接创建产品对象的责任,而仅仅“消费”产品。简单工厂模式通过这种做法实现了对责任的分割。
- 2) 虽然简单工厂模式不满足 OCP 原则,但它同样具备良好的扩展性:扩展的时候仅需要修改少量的代码(修改工厂类的代码)就可以满足扩展性的要求了。
- 3) 从结构复杂度比较,简单工厂模式只需一个工厂类,而工厂方法模式和抽象工厂模式的工厂类随着产品类个数增加而增加,这无疑会使类的个数越来越多,从而增加了结构的复杂程度。
- 4) 从客户端编程难度比较,工厂方法模式和抽象工厂模式在客户端编码中需要对工厂类进行实例化。而简单工厂模式的工厂类是个静态类,在客户端无需实例化。

其中抽象产品角色由 `IParser` 接口担任,具体产品角色包括 `TPIUParser`、`ITMParser`、`SIPTextParser` 等,工厂类角色由 `ParserFactory` 类担任。

抽象产品角色 `IParser` 接口定义如下:

```
public interface IParser {
    public void setData(byte[] buf); // 向解析器传入待解析的字节流数据
}
```

具体产品角色 `TPIUParser`、`ITMParser`、`SIPTextParser` 等解析器均实现 `IParser` 接口。

简单工厂模式的核心工厂类角色 `ParserFactory` 提供了静态工厂方法 `createParser(ParserType)`,该方法根据参数解析器类型创建对应的解析器,返回对象为 `IParser` 类型。具体代码如下:

```
public class ParserFactory {
    public enum ParserType { TPIU, ITM, SIPTEXT }
    public synchronized static IParser createParser(ParserType type) {
        if (type == null)
            return null;
        if (type == ParserType.TPIU)
            return new TPIUParser();
        if (type == ParserType.ITM)
```

```

        return new ITMParser();
    if (type == ParserType.SIPTEXT)
        return new SIPTextParser();
    .....
    return null;
}
}

```

在使用简单工厂方法模式后，降低了系统间的耦合度，创建解析器的工作从 SG、IDS 模块转到 ParserFactory 内。在 SG、IDS 模块内，只需调用 ParserFactory 类的静态工厂方法 createParser(ParserType)，便可得到相应的解析器。

4.3.4.3 ITMParser 的实现

本节以 ITMParser 为例详细介绍解析器的实现及其工作流程。

ITMParser 作为简单工厂模式中的具体产品角色，实现了 IParser 接口，并由核心工厂类 ParserFactory 提供创建。ITMParser 解析器主要完成 TPIUSG 模块中的包解析工作。ITMParser 内部解析流程如图 4-13 所示。

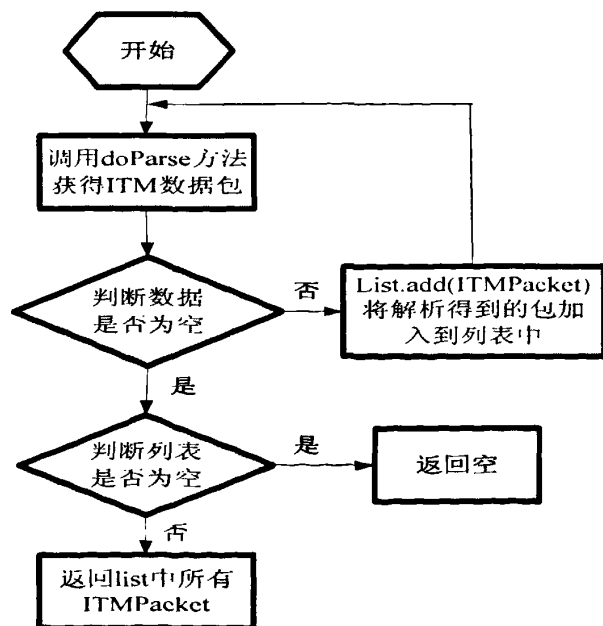


图 4-13 ITMParser 内部解析流程

除 setData 方法外, ITMParser 中还包含有如下公共接口:

1) public ITMPacket[] getResult(), 从字节流数据中解析获取 ITM 数据包的 payload, 此接口调用私有方法 doParse 进行解析。

2) public int getBytesConsumed(), 得到解析器已经完成解析的字节流数据长度, 未解析的字节流将会在下次调用 setData 方法时重新传入解析器。

其中, ITM 数据包的 payload 定义如下:

```
public class ITMPacket {
    private byte[] data;
    private IPacketType type;
    public ITMPacket( byte[] data, IPacketType type ){
        this.data = data;
        this.type = type;
    }
    public byte[] getData(){
        return data;
    }
    public IPacketType getType(){
        return type;
    }
}
```

ITM 数据包类型定义如下:

```
public enum ITMPacketType implements IPacketType {
    TimeStamp, TimeStick, HSP, SIP, Sync,
    Overflow, Idle, Extension, Reserved, Error
}
```

doParse 方法按照 ITM 包格式进行解析, 具体过程如下:

- 1) 读入下一字节 b;
- 2) 如 b=0, 按同步包格式解析;
- 3) 如 b=0x70, 按溢出包格式解析;
- 4) 如 b&0x0f=0, 即字节的低四位均为 0, 按时间标记信息包格式解析;
- 5) 如 b&0x0b=0x08, 即字节的低两位均为 0, 低四位为 1, 按扩展包格式

解析;

- 6) 如 $b \& 0x0f = 0x04$, 即字节的低三位为 1, 按保留格式解析;
- 7) 如 $b \& 0x03 \neq 0$, 即字节的低两位起码有一位不为 0, 按 SIP、HSP 包格式解析。

4.3.5 回溯控制

流生器或智能数据源调用相应解析器的 `setData` 方法传入待解析的字节流, 在解析的过程中, 存在这种情况: 字节流的最后若干个字节不能组成完整的解析数据格式, 需要与后续传入的字节流一起才能得到完整的解析。这时, 未解析的字节将会进行回溯, 并在下次调用 `setData` 方法时重新传入解析器。

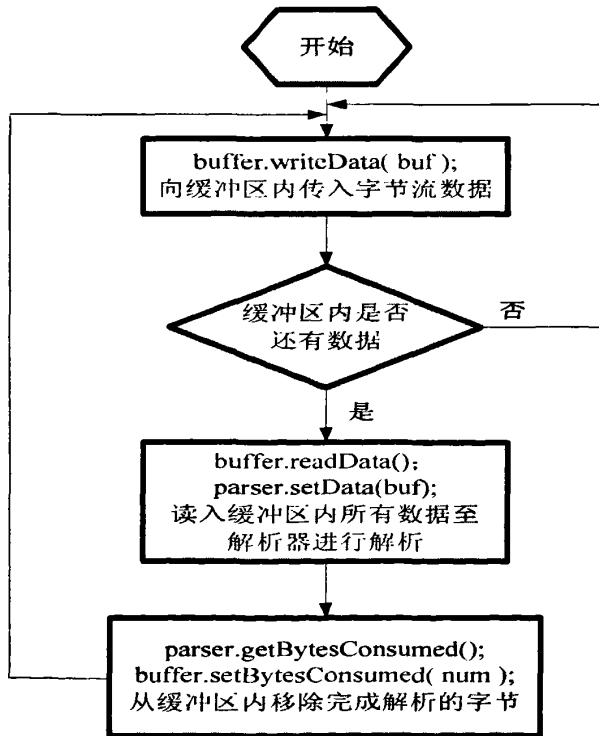


图 4-14 缓冲区控制回溯流程图

解析模块直接向解析器内传入字节流数据是无法控制回溯的, 因此, 我们在解析模块内部提供了相应的缓冲区来解决这个问题, 如 TPIUSG 的 `ITMBuffer`,

ITM.SIP.Text 的 SIPBuffer。字节流数据首先被送入到缓冲区内，然后解析器再从缓冲区内获取数据进行解析。解析模块利用缓冲区控制回溯的流程如图 4-14 所示，步骤如下：

- 1) 向缓冲区内送入待解析的字节流数据；
- 2) 从缓冲区内读入数据至解析器进行解析；
- 3) 解析完成后，从缓冲区内清除已完整解析的字节，未解析的字节仍存于缓冲区，等待下次获得完成解析，重复前面两个步骤。

4.4 第三方扩展实现

本系统利用 Eclipse 插件体系结构和扩展机制进行开发，为 RDS、SG、IDS 模块创建各自的扩展点，以达到向第三方产品提供商提供扩展接口的功能，即第三方可以按照模块提供的扩展点编写自己的 RDS、SG、IDS 然后加入到本系统中。第三方的扩展必须以 Eclipse 插件的形式提供。本节将以智能数据源模块为例，详细介绍系统中第三方扩展的实现。

4.4.1 自定义扩展点

插件利用扩展这个机制向 Eclipse 平台添加新功能，所使用的扩展可以依据 Eclipse 提供的众多现成的扩展点，而这些扩展点有时并不能满足用户的需要。这时，用户就需要定义新的扩展点供插件扩展，以此来实现新的功能。

自定义扩展点的基本步骤如下所述^[31]：

- 1) 选择可以让其他人扩展或配置的现有代码。

扩展点有两种类型：创建新的功能（如创建新的视图）或向现有的插件添加新的功能（如向现有的视图添加菜单项）；定义只有数据的配置扩展（如帮助添加项）。

如果在扩展点的属性中，有一个或多个属性引用了某个类，那么这种扩展点就属于第一种类型的扩展（RDS、SG、IDS 扩展点即属此类）。扩展点的第二个通用类型是那些只包括配置数据的类型。

- 2) 声明新扩展点的存在。

声明过程中，有三个参数用于<extension-point>标记。IDS 扩展点

<extension-point>标记结构如下所示：

```
<extension-point
    id = "com.arm.trace.core.ids"
    name = "Intelligent Data Source"
    schema = "schema/ com.arm.trace.core.ids.exsd"/>
```

<extension-point>元素的三个属性中，id 是简单的标识符，它在插件内唯一；name 是这个扩展点显示给用户的名称；schema 是这个扩展点的模式配置。

schema 包括配置扩展点元素和相关的元素属性（attribute）。对于每一个扩展点元素属性（attribute），具有若干与它们关联的项目（property）。一些重要的项目（property）及其说明如表 4-5 所示。

表 4-5 项目说明

项目（property）	说明
Name	属性（attribute）的名字，出现在扩展声明中
Type	属性的类型，boolean、string、java、resource
Use	表示扩展中是否要求该属性。required:必须显示声明；optional:可选属性；default:属性取默认值
Default Value	Use 设定为 default 时，用来设定默认值
Extends	Type 设定为 java 时，表示类的完全限定名（该属性必须扩展这个类）
Implements	Type 设定为 java 时，表示接口的完全限定名（该属性必须实现这个接口）

IDS 扩展点配置了唯一一个扩展点元素，即 IDS。IDS 元素包含以下一些属性，如表 4-6 所示。

表 4-6 IDS 元素属性

元素属性（attribute）	意义
id	IDS 扩展的唯一标识符，必须配置
name	IDS 扩展的名称，必须配置
class	IDS 扩展实现特定接口的类，必须配置
description	简单描述 IDS 扩展点，可选

3）为添加进来的类的预期行为定义扩展接口或抽象类。

在第一步中提到过，有些情况下扩展点只用于把配置信息传递给插件，也就

是说，它并不指定任何引用某个类的属性，更不会在运行时实例化该类。但更多的情况是，扩展点需要在可执行代码的帮助下才能完成它的任务。

定义的扩展接口或抽象类是扩展点创建者和添加者之间的约定。从 ITM.SIP.Text 抽象出 IIDS 接口如下所示：

```
public interface IIDS extends Runnable {
    //设置从 SG 模块获取的配置参数
    public void setConfigParam( HashMap<String, String> map);
    public void setDataIn(IIDSDataIn dataInObj);//设置数据输入
    public void setDataOut(IIDSTextDataOut textOut);//设置 text 数据输出
    public void setDataOut(IIDSEventDataOut evtOut);//设置 event 数据输出
    //设置 analog 数据输出
    public void setDataOut(IIDSAanalogDataOut analogOut);
    public int getChannelID();//获取 IDS 模块接受解析数据的通道 ID
    public void stop();//停止运行模块线程
}
```

这个接口被指定为扩展点模式定义（testId.exsd）的一部分，IDS 元素 class 属性中的 Implements 项目（property）就是用来定义接口的。

4) 为扩展点处理注册表项。

一旦您的插件被加载，作为扩展点的创建者，您就要负责处理任何对它进行的扩展。对您的插件扩展点进行扩展的那些插件指定的任何信息（包括所有的属性和子标记），都将保存在插件的扩展注册表中。您首先要决定的是，何时读入扩展注册表，并处理那些应用于插件定义的扩展点上的条目。一些插件在它们的插件类被加载时，通过覆盖 start 方法来处理它们所有的扩展点项，不过这样会影响到 Eclipse 的性能。推荐的处理方式是，直到插件确实需要这个扩展点条目时才处理它，并且只处理当时代码中确实需要的那些注册表项。

扩展点处理代码的基本结构如下代码所示：

```
IExtensionRegistry registry = Platform.getExtensionRegistry();
IExtensionPoint extensionPoint=registry.getExtensionPoint(extensionPointId);
IExtension[] extensions = extensionPoint.getExtensions();
for (IExtension e : extensions) {
    IConfigurationElement[] elements=e.getConfigurationElements();
```

```
for (IConfigurationElement element : elements) {
    .....//处理过程 }
.....//处理过程
}
```

4.4.2 使用扩展点

在插件定义好一个新的扩展点后，这个扩展点就可以被用来扩展一个新的功能了。它的使用方法和 Eclipse 中已有的插件扩展点使用是一致的。借鉴 Eclipse 开发的公平竞赛法则（Fair Play Rule）：“所有使用者应遵循同样的游戏规则，包括我自己”，于是本系统内部的扩展也必须使用定义好的扩展点和扩展接口来实现。

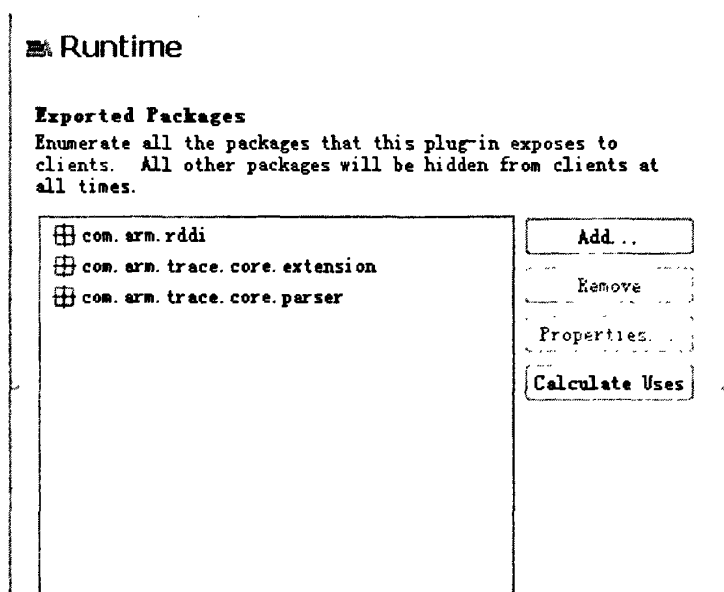


图 4-15 导出 IDS 扩展点接口

首先，新定义的扩展点可以直接在此插件中对其进行扩展，更常见的用法是在另外一个插件中对这个扩展点进行扩展^[32]。如果要在其他的插件中扩展这个新的扩展点，需要做如下一些设置（这里我们将定义了新的扩展点的插件称作原插件，使用原插件提供的扩展点的插件称为扩展插件）：将原插件中所包含的自定义扩展点的接口或抽象类导出，同时在扩展插件中设置依赖于原插件。本

系统中定义的 IDS 扩展点需要为第三方提供在自己的插件内扩展此接口的功能，属于第二种用法。IDS 扩展点接口的导出如图 4-15 所示，Runtime 中使用 add 按钮进行导出，IDS 扩展点接口包含在 `com.arm.trace.core.extension` 内。

4.5 本章小结

本章首先简单介绍了系统中 CoreSight 跟踪数据的解析流程；然后，给出了解析模块的详细设计，包括流生成器模块和智能数据源模块。

接着，着重讨论了基于 ITM 数据流的解析实现。首先给出了 ITM 包数据格式说明，然后详细分析了 TPIUSG 模块和 ITM.SIP.Text 模块的解析流程，并给出了内部解析器的具体实现。其中包括一些关键技术的研究：简单工厂模式在解析器中的应用，利用缓冲区控制数据流的回溯等。

最后，以 IDS 模块为例，详细介绍了本系统中第三方扩展的实现。

第 5 章 CoreSight 跟踪数据的前台展示

5.1 CoreSight 跟踪数据的前台展示概述

CoreSight 跟踪数据经过流生器模块的解析，智能数据源模块的翻译后，最终获得其包含的具体信息（包括文本内容、事件名称、事件发生时间、事件停止时间、变量值等）。接下来的任务就是根据用户的配置需求，将这些信息以可视化的形式（包括文本、事件、模拟等类型）展示给用户，为用户提供全方位的系统观察以供后续的性能分析和程序调试。

5.1.1 基本体系架构

CoreSight 跟踪数据的前台展示采用类似于 Web 开发中 JSP Model1 体系架构进行设计，JSP Model1 在一定程序上实现了 MVC^[33]。MVC 是一种流行的软件设计模式，它把系统分为 3 个模块：模型（Model）、视图（View）、控制器（Controller）。各个模块的功能说明如表 5-1 所示。

表 5-1 MVC 的三个模块

MVC 模块	描述
模型	代表应用程序状态和业务逻辑
视图	提供可交互的客户界面，向客户显示模型数据
控制器	响应客户的请求，根据客户的请求来操纵模型，并把模型的响应结果经由视图展现给客户

各个模块间的相互作用情况如图 5-1 所示。客户可以从视图提供的客户界面上浏览数据或发出请求，客户的请求由控制器处理，它根据客户的请求调用模型的方法，完成数据更新，然后调用视图的方法将响应的结果展示给客户。视图也可以直接访问模型，查询数据信息，当模型中数据发生变化时，它会通知视图刷新界面，显示更新后的数据。

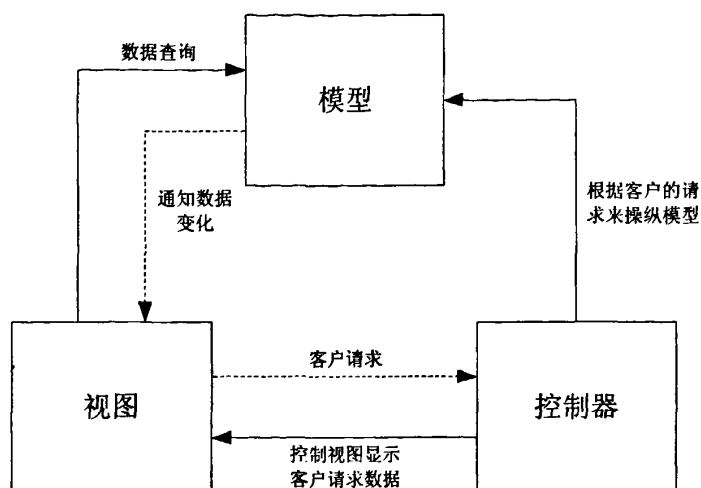


图 5-1 MVC 模型图

在 JSP Model1^[34]体系中，JSP 页面响应用户请求并将处理结果返回用户。JSP 既要负责业务流程控制，又要负责提供表示层数据，同时充当视图和控制器的角色；bean 则充当模型的角色。

CoreSight 跟踪数据的前台展示体系架构如图 5-2 所示。

其中，Display 模块和 Display 配置模块相当于 JSP Model1 中的 JSP，负责数据显示和控制业务逻辑的处理；模型相当于 JSP Model1 中的 bean，负责数据的存储管理。使用 JSP Model1 后，前台展示部分的数据与显示进行了分离，松散了系统间的耦合度。

Display 模块接收平台 ids2display 模块传递过来的显示数据后，并不直接显示，而是先将数据送至模型，由模型部分来管理所有的显示数据。这时，Display 模块起到了控制器的作用，调用模型的方法，向模型中添加数据，更新状态。当 Display 模块从模型中获取数据进行显示时，则相当于视图。

Display 配置模块提供各 Display 的配置对话框，允许用户配置各个 Display 的细节信息，如字体、颜色、线型等。此时，各配置对话框即充当了视图的角色。当完成配置，更新模型时，Display 配置模块又相当于控制器，调用模型的方法，改变模型中数据的配置。

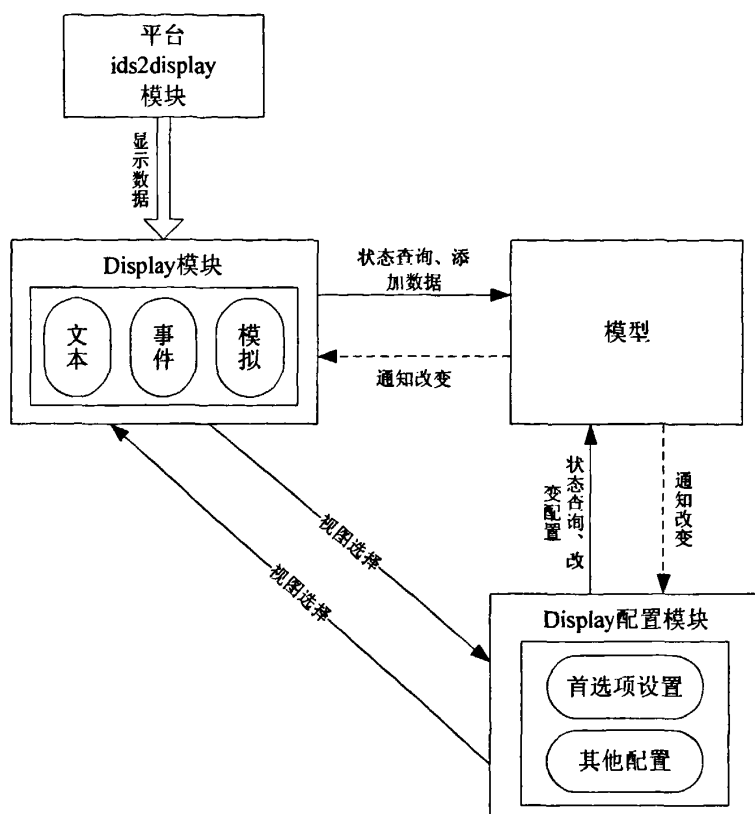


图 5-2 前台展示体系架构

5.1.2 功能模块划分

通过架构图可以看出，前台展示部分主要划分为如图 5-3 所示的几个功能模块。

Display 模块构成前台展示系统的主体，包括文体、事件、模拟三种类型。

首选项设置和其他配置一起组成完整的 Display 配置模块。

首选项设置模块提供对 Display 模块初始化的设置，包括字体、颜色、背景、网格、刷新频率等信息。总体设置是对 Display 模块全局的设置；文本显示设置、事件显示设置、模拟显示设置则分别对 Display 中的文本、事件、模拟显示子模块提供初始化设置；模式转换设置包括模式切换时数据存储路径、数据转存策略等信息。

其他配置则包括线型、线条粗细、线条颜色等信息。这些配置信息都是根

据 Display 模块的显示记录动态产生的，因此不能集成在首选项设置中，而需要另外提供配置对话框。

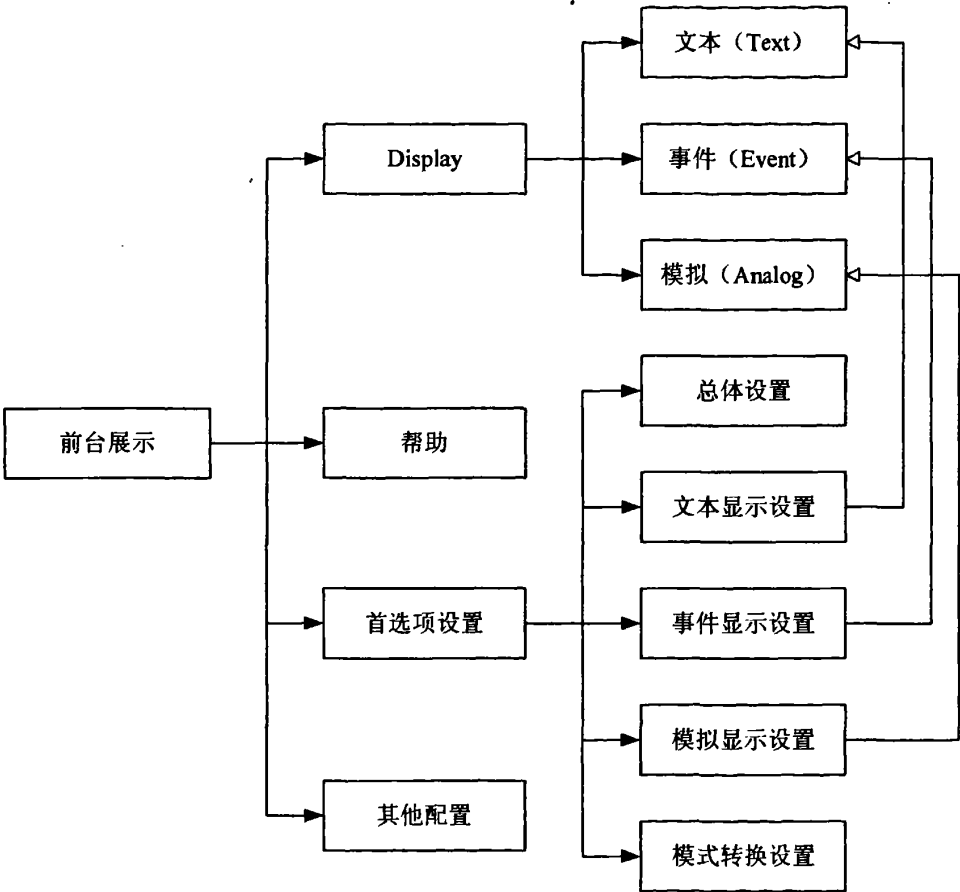


图 5-3 前台展示部分的功能模块划分

5.2 显示数据

5.2.1 显示数据的格式

5.2.1.1 Text 显示数据格式

Text 显示数据由文本产生的时间（或时间偏移）和文本数据两个部分组成，其中文本数据是必须包含的。文本分为普通文本和控制台转义字符两种，格式

如下所示:

TextDisplayData= [TextDisplayTime | TimeOffset] + TextDisplayText

TextDisplayText = TextDisplayPlainText | TextDisplayEscapeText

Text 显示数据接口定义如下:

```
public interface ITextDisplayData {
    public String getTime();// 获取文本产生的时间
    public long getTimeOffset();// 获取时间偏移
    public String getPlainText();// 获取普通文本
    public ICSICommand getEscapeCode();// 获取控制台转义字符
}
```

5.2.1.2 Event 显示数据格式

Event 显示数据由事件产生的时间(或时间偏移)和事件数据两个部分组成,其中事件数据部分是必须包含的。事件由事件 ID、关联事件 ID、事件类 ID、事件源 ID、和事件属性组成。事件属性包括属性 ID 和属性值两个部分。格式如下所示:

EventDisplayData = [EventTime | EventTimeOffset] + EventData

EventData = EventID + [EventLinkedID] + EventClassID + EventSourceID + {EventAttributeID + EventAttributeValue}

Event 显示数据接口定义如下:

```
public interface IEventDisplayData {
    public String getTime();// 获取事件产生的时间
    public long getTimeOffset();// 获取时间偏移
    public int getEventID();// 获取事件 ID——事件唯一标识符
    public int getEventLinkedID();// 获取关联事件 ID
    public int getEventClassID();// 获取事件类 ID
    public int getEventSourceID();// 获取事件源 ID
    public String getDescription();// 得到关于该事件的描述信息
    public IEventAttribute[] getEventAttribs();// 获取事件属性信息
}
```

5.2.1.3 Analog 显示数据格式

Analog 显示数据由数据产生的时间（或时间偏移）和 Analog 数据两个部分组成，其中 Analog 数据部分是必须包含的。Analog 数据由数据所属的通道 ID 和数据的变量取值组成。格式如下所示：

AnalogDisplayData = [Time | TimeOffset] + AnalogData

AnalogData = AnalogChannelID + AnalogValue

Analog 显示数据接口定义如下：

```
public interface IAnalogDisplayData {
    public String getTime();// 获取 Analog 数据产生的时间
    public long getTimeOffset();// 获取时间偏移
    public int getAnalogChannelID();// 获取 Analog 数据所属的通道 ID
    public int getAnalogValue();// 获取 Analog 数据的变量值
}
```

5.2.2 显示数据的传递

5.2.2.1 Eclipse 中的 Job 机制

Job^[35]可以说是 Eclipse 中比较核心又在我们使用中经常碰到的一种机制。本质上是 Eclipse 中的多线程的一种表现，与 IRunnable 功能类似。但是更加对象化，容易扩展和控制，属于 EclipseUI 的一部分。

Eclipse 为 Job 提供了 3 个扩展：WorkspaceJob、UIJob、WorkbenchJob，算上 Job 本身构成了 Eclipse 对多线程的支持。

- 1) WorkspaceJob 是为修改资源文件增加的扩展，常见的对文件的打开、保存等等操作一般需要在这个类中执行。与 WorkspaceJob 对应的是 IWorkspaceRunnable。
- 2) UIJob 是在 UI 线程上干活，可以直接在它里面运行改变 UI 的代码，所有继承 UIJob 的类应该覆写 runInUIThread 方法。与 UIJob 对应的是：

```
display.asyncExec (new Runnable () {public void run () {}});
```

SWT 对于所有来自非 UI 线程的调用，将触发 SWTException。这些调用必须来自 UI 线程。

- 3) WorkbenchJob 是 UIJob 的扩展，也是在 UI 线程上干活，目的是仅当工

作台正在运行时才能调度或运行作业。

Eclipse 中的 Job 封装了线程池的实现。当我们启动一个 Job 时, Eclipse 不会马上新建一个 Thread, 它会在它的线程池中寻找是否有空闲的线程, 如果有空闲线程, 就会直接用空闲线程运行你的 Job。一个 Job 终止时, 它所对应的线程也不会立即终止, 它会被返回到线程池中以备重复利用。这样, 我们可以节省创建和销毁线程的开销。

Eclipse 的核心包中提供了一个 JobManager 类, 它实现了 IJobManager 接口, Eclipse 中 Job 的管理和调度都是由 JobManager 来实现的。JobManager 维护有一个线程池, 用来运行 Job。当我们调用 Job 的 schedule 方法后, 这个 Job 会被 JobManager 首先放到一个 Job 运行的等待队列中去。之后, JobManager 会通知线程池有新的 Job 加入了运行等待队列。线程池会找出一个空闲的线程来运行 Job, 如果没有空闲线程, 线程池会创建一个新的线程来运行 Job。一旦 Job 运行完毕, 运行 Job 的线程会返回到线程池中以备下次使用。可以看出, JobManager 介入了一个 Job 运行的全过程, 它了解 Job 什么时候开始, 什么时候结束, 每一时刻 Job 的运行状态。JobManager 将这些 Job 运行的信息以接口的方式提供给用户, 同时它也提供了接口让我们可以介入 Job 的调度等, 从而我们拥有了更加强大的控制 Job 的能力。

5.2.2.2 UIJob 在显示数据传递过程中的应用

Event Viewer 系统前台展示采用 SWT/Jface 图形 API 编写, 显示数据通过平台传递给 Display 这一过程涉及到非 UI 线程对 UI 线程的访问。因此, 可以直接在其中运行改变 UI 代码的 UIJob 成为了选择。具体代码如下所示:

```
private Job createDataJob() {
    return new UIJob("UIJob") {
        public IStatus runInUIThread(IProgressMonitor monitor) {
            //数据为空, 返回状态 CANCEL
            if (data == null)
                return Status.CANCEL_STATUS;
            //视图不存在或已销毁, 返回状态 CANCEL
            if (displayView == null || displayView.isDisposed())
                return Status.CANCEL_STATUS;
```

```
//调用视图的 addData 方法，将显示数据传递给 Display;  
//此方法中包含有 UI 操作  
displayView.addData(data);  
return Status.OK_STATUS;  
}  
};  
}
```

在显示数据传递过程中，系统会创建大量的 UIJob。当我们调用 `schedule` 方法后，这些 Job 会被 `JobManager` 首先放到等待队列中去，至于什么时候运行我们就无法决定了。为了保证数据传递的准备性和次序性，即先解析出来的数据要保证先传递给 `Display`，UIJob 在调用 `schedule` 方法后，马上调用 `join` 方法，该方法阻断当前调用者，直到 job 运行完成后再继续。

5.3 Analog 展示的详细设计与实现

本人在系统前台展示模块的开发过程中，主要负责 Analog 展示方面的工作，包括 Analog display 视图、存储管理 Analog 显示数据的模型、相关的首选项设置和其他配置等几个部分。本节将详细介绍它们的设计与实现。

5.3.1 展示界面 Analog display 视图

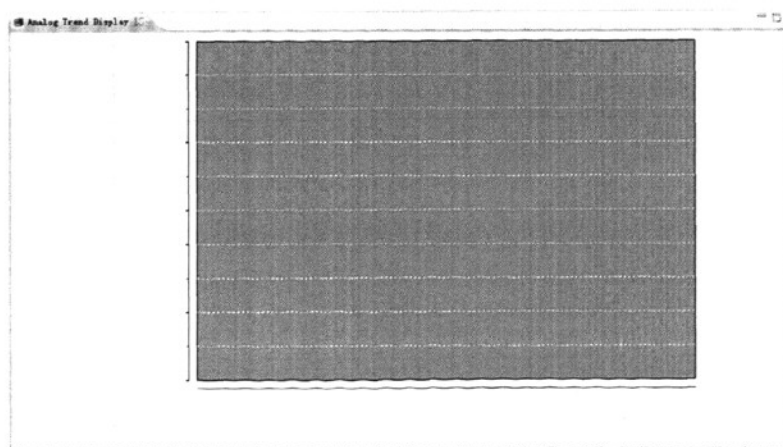


图 5-4 Analog display 视图初始界面

展示界面 Analog display 视图扩展于 Eclipse 的 view 扩展点，初始界面如图 5-4 所示。内部主体是两个画布（canvas）：其中右边的 canvas 利用实时曲线来动态反映 Analog 显示数据的变量值随时间的变化趋势，这是主要的画布，称为 Analog display canvas；左边的 canvas 用来标识 Analog display canvas 中每一条实时曲线对应的通道数据，称为 Analog channel canvas。具体代码如下：

// 创建视图中的界面组件

```
public void createPartControl(Composite parent) {
    parent.setLayout(new FillLayout());
    // 定义一个分割窗容器
    SashForm sf = new SashForm(parent, SWT.NONE);
    sf.SASH_WIDTH = 2;
    sf.setBackground(parent.getDisplay().getSystemColor(SWT.COLOR_TITLE
_BACKGROUND_GRADIENT));
    // 在分割窗容器中创建两个 canvas，默认是按从左至右的顺序依次排列，
    // 即左边为 Analog channel canvas，右边为 Analog display canvas，
    // 常量 SWT.DOUBLE_BUFFERED 设置画布使用双缓冲技术绘图
    Canvas channel=new Canvas(sf, SWT.DOUBLE_BUFFERED|SWT.NONE);
    Canvas canvas = new Canvas(sf, SWT.DOUBLE_BUFFERED|SWT.NONE);
    // 设置分割窗中各栏的宽度比例，左：右=15：100
    sf.setWeights(new int[] {15, 100});
    // 定义存储管理显示数据的模型
    model = new AnalogDisplayModel(parent);
    displayCanvas = new AnalogDisplayCanvas(this, canvas, model);
    channelCanvas=new AnalogChannelCanvas(channel, displayCanvas, model);
    // 为 Analog channle canvas 添加绘制监听
    channelCanvas.addPaintListener();
    // 为 Analog display canvas 添加绘制监听和鼠标监听
    displayCanvas.addPaintListener();
    displayCanvas.addMouseListener();
    .....
}
```


5.3.2 存储管理 Analog 显示数据的模型

Analog 前台展示的模型部分类似于数据库，起到了存储管理显示数据的作用。通过平台传递过来的 Analog 显示数据首先被送到模型部分，根据所属通道 ID 分开存储，并为每个通道存储区内数据设置默认的显示细节，如线条颜色、线型、线条粗细等。Analog 数据存储流程如图 5-5 所示。

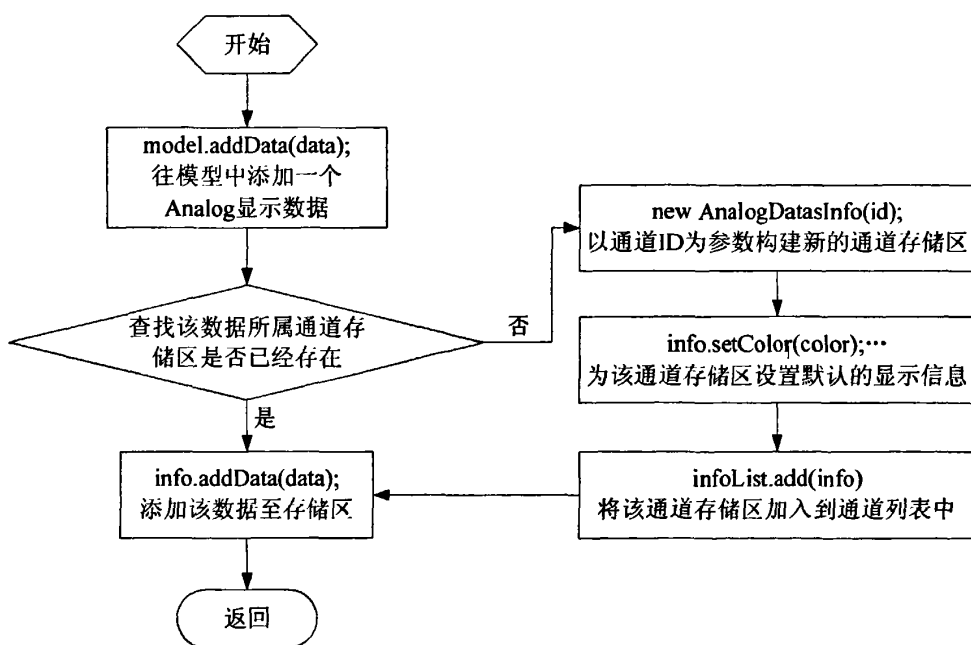


图 5-5 Analog 数据存储流程

5.3.3 Analog 首选项

Analog display 首选项如图 5-6 所示。主要包括如下一些设置：

- ◆ 刷新频率：设置 Analog display 视图实时曲线图重绘的周期，单位 ms。
- ◆ 时间跨度：设置 Analog display 视图一屏所能显示的时间范围，单位 s；当显示数据的跨度超过这个范围时，将会产生横向滚动条并动态显示最近一秒内的数据变化趋势。
- ◆ 数据缓冲区：设置显示数据缓冲区的大小，单位 kbyte；当缓冲区内存储的数据量超过缓冲区大小时，利用先进先出原则，从缓冲区内移除最早的数据。

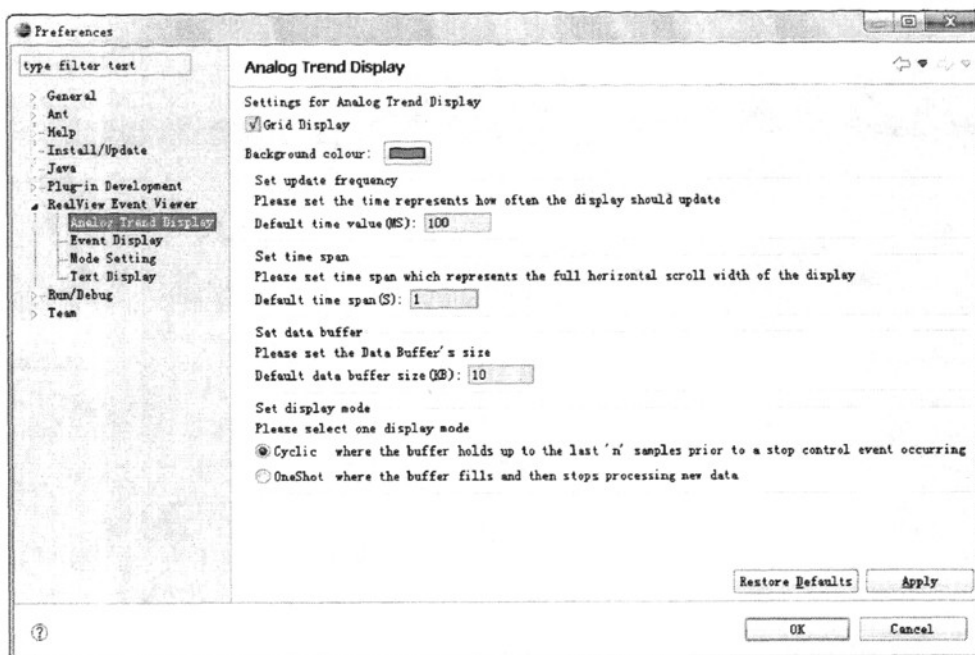


图 5-6 Analog display 首选项

首选项主要提供对显示模块的初始化设置，如果在运行过程中重新设置了首选项，则需要利用到属性改变监听器模式完成整个配置过程。

属性改变监听器模式中，有一个属性提供器，属性提供器暴露某些属性，并接受属性改变监听器的注册，在属性值发生改变时通知已经注册的监听器。属性提供器不需要实现特定的接口。属性提供器使用 `PropertyChangeEvent` 来创建一个可以有效填充和传播的事件，另外，属性提供器负责维护监听器列表并对它们进行回调。

Eclipse 中已提供 `PreferenceStore`^[35] 作为属性提供器存在，我们只需实现属性改变监听器，并调用 `PreferenceStore` 的 `addPropertyChangeListener` 方法注册该监听器即可。属性改变监听器必须实现 `org.eclipse.jface.util.IPropertyChangeListener` 接口，以便允许属性改变提供器对它进行回调。`AnalogPropertyListener` 类实现了该接口，提供首选项属性改变监听器，部分代码实现如下：

```
public class AnalogPropertyListener implements IPropertyChangeListener {
    // 必须实现的接口方法，监听属性事件的变化
    public void propertyChange(PropertyChangeEvent event) {
```

```

// 判断首选项属性
if (event.getProperty().equals(PreferenceConstants.
        P_ANALOG_DISPLAY_BUFFER_SIZE)) {
    // 调用模型相关方法重置缓冲区大小
    model.setBufferSize(Integer.parseInt(store.getString(PreferenceConstants
        .P_ANALOG_DISPLAY_BUFFER_SIZE)));
    // 刷新视图
    canvas.redraw();
}
}
}

```

提供首选项属性改变监听器后，Analog display 视图中注册该监听器的代码实现片断如下：

```

public void createPartControl(Composite parent) {
    // 获取首选项属性提供者 PreferenceStore
    IPreferenceStore store = Activator.getDefault().getPreferenceStore();
    // 构造首选项属性改变监听器
    AnalogPropertyListener analogLisntener=new AnalogPropertyListener(
        this, displayCanvas, model, store);
    // 接收该监听器的注册
    store.addPropertyChangeListener(analogLisntener);
}

```

5.3.4 其他配置

除了首选项之外，Analog display 视图中还提供其他的动态配置对话框对 Analog 展示进行设置，主要包括如下一些配置对话框：

- ◆ 线条颜色（Line Color）配置对话框：配置视图中每条实时曲线的颜色；
- ◆ 线条粗细（Line Thick）配置对话框：配置视图中每条实时曲线的粗细；
- ◆ 线条类型（Line Style）配置对话框：配置视图中每条实时曲线的线型；
- ◆ 纵坐标值域（Line Range）配置对话框：配置视图中每条实时曲线的纵坐标值域范围。

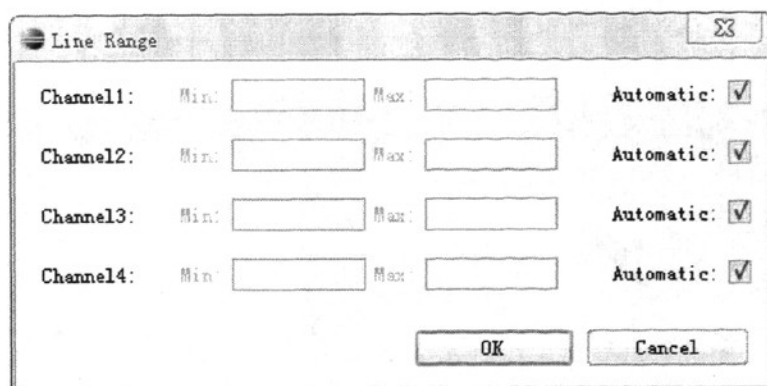


图 5-7 纵坐标值域配置对话框

图 5-7 所示为动态生成的纵坐标值域配置对话框。Automatic 表示值域根据通道内数据的最大值最小值来确定；同时用户也可以自己设定想要观察的值域范围。设置完成后，点击 OK 即可完成视图更新。

使用动态配置对话框对 Analog 展示进行设置的具体流程如图 5-8 所示。

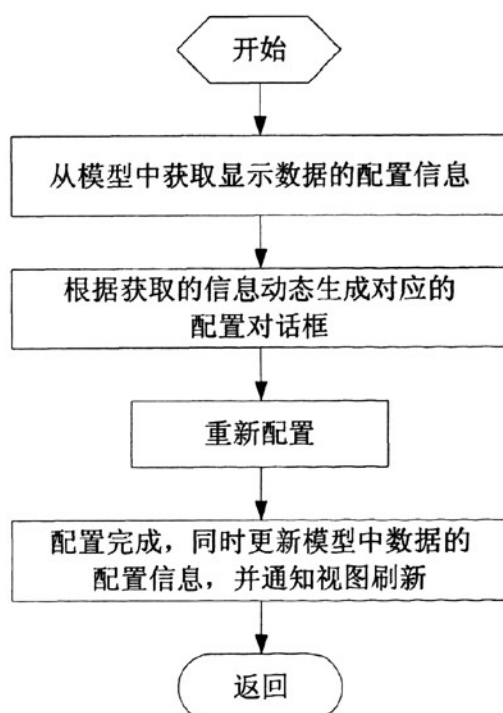


图 5-8 使用配置对话框进行配置的具体流程

5.3.5 实时监测 Analog 数据变化趋势

Analog display 视图中以实时曲线的形式动态表示 Analog 数据的变化趋势。用户通过观察实时曲线图，可以直观了解到某个时间跨度内来自同一数据源多个通道内 Analog 数据的变化趋势。

图 5-9 所示是来自 DCC 数据源的四个通道内的 Analog 数据的变化趋势，每一个通道的数据对应一条实时曲线。当用户暂停显示或显示完成后，用户可以使用滚动条回滚重新观察前面的信息。其中，横轴表示 Analog 数据产生的时间（或时间偏移），纵轴表示 Analog 数据的变量值。每一个 Analog display 视图至多可同时显示来自 16 个通道的数据，视图左边按照曲线颜色标明每条实时曲线代表的是哪一通道内的 Analog 数据。

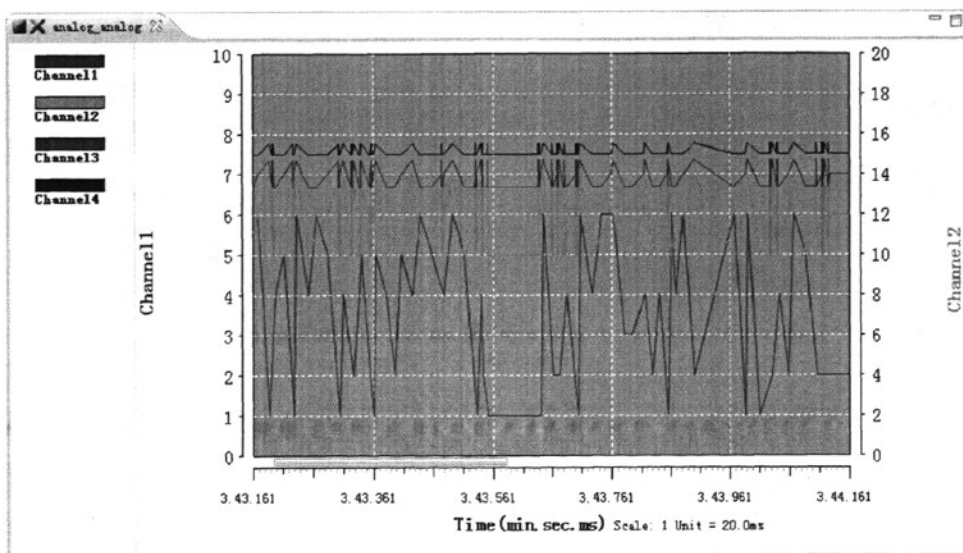


图 5-9 DCC 数据源四个通道内数据的变化趋势

5.3.5.1 实时曲线图的绘制

Analog display 视图中的 Analog display canvas 和 Analog channel canvas 使用 GC 对象来完成实时曲线图的绘制功能，主要用到了如下一些方法^[36]：

1) drawLine(int x1, int y1, int x2, int y2): 绘制一条直线，其中 x1、y1 为直线的一个点坐标，x2、y2 为直线另一个点的坐标。

该方法用于视图右方的 Analog display canvas 中绘制横坐标轴、纵坐标轴、网格线以及实时曲线主体。绘制实时曲线时，对于同一通道内时间上相邻的两个数据点，都会使用该方法一次。

2) drawRectangle(int x, int y, int width, int height): 绘制矩形，其中 x、y 为矩形的左上角顶点，width 为矩形的宽，height 为矩形的高。

fillRectangle(int x, int y, int width, int height): 填充矩形。

这两个方法用于视图左方的 Analog channel canvas 中绘制标明曲线数据对的颜色块。

3) drawString(String string, int x, int y): 其中 string 为要显示的字符串，x 表示字符串所在的横坐标、y 表示字符串所在的纵坐标。


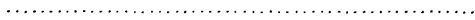

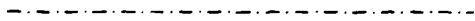

该方法用于绘制视图上所有相关字符文本，包括时间格式说明、横坐标的时间值、纵坐标的变量值、通道 ID 名等。

4) setLineStyle(int lineStyle): 设置线条的样式。

setLineWidth(int lineWidth): 设置线条的粗细，以像素为单位。

这两个方法主要用于 Analog display canvas 中设置实时曲线的样式和粗细。样式对应的常量及其效果如表 5-2 所示。

表 5-2 设置线条样式

样式常量	效果图
SWT.LINE_SOLID	
SWT.LINE_DOT	
SWT.LINE_DASH	
SWT.LINE_DASHDOT	
SWT.LINE_DASHDOTDOT	

5.3.5.2 定期刷新视图

Analog display 视图中添加绘制监听后，视图只会在其第一次创建时调用绘制方法，而 Analog 显示数据正源源不断地被送过来。因此，要在 Analog display 视图中动态地观察到 Analog 数据的变化趋势，还必须定期刷新视图。

java.util.concurrent.ScheduledExecutorService 类^[37]对于调度那些周期性执行的事务非常方便。常用的方法如下所示：

1) schedule(Runnable command, long delay, TimeUnit unit): 创建并执行在给

定延迟后启用的一次性操作。command, 要执行的任务; delay, 从现在开始延迟执行的时间; unit, 延迟参数的时间单位。

2) `schedule(Callable<V> callable, long delay, TimeUnit unit)`: 创建并执行在给定延迟后启用的 `ScheduledFuture`。callable, 要执行的功能; delay, 从现在开始延迟执行的时间; unit, 延迟参数的时间单位。

3) `scheduleAtFixedRate(Runnable command, long initialDelay, long period, TimeUnit unit)`: 创建并执行一个在给定初始延迟后首次启用的定期操作, 后续操作具有给定的周期; 也就是将在 `initialDelay` 后开始执行, 然后在 `initialDelay+period` 后执行, 接着在 `initialDelay + 2 * period` 后执行, 依此类推。如果任务的任一执行遇到异常, 都会取消后续执行。否则, 只能通过执行程序的取消或终止方法来终止该任务。command, 要执行的任务; initialDelay, 首次执行的延迟时间; period, 连续执行之间的周期; unit, initialDelay 和 period 参数的时间单位。

4) `scheduleWithFixedDelay(Runnable command, long initialDelay, long delay, TimeUnit unit)`: 创建并执行一个在给定初始延迟后首次启用的定期操作, 随后, 在每一次执行终止和下一次执行开始之间都存在给定的延迟。如果任务的任一执行遇到异常, 就会取消后续执行。否则, 只能通过执行程序的取消或终止方法来终止该任务。command, 要执行的任务; initialDelay, 首次执行的延迟时间; delay, 一次执行终止和下一次执行开始之间的延迟; unit, initialDelay 和 delay 参数的时间单位。

Analog display 视图使用 `ExecutorService` 对象的 `scheduleAtFixedRate` 方法完成定期刷新, 即时更新视图以获得最新的反映数据变化趋势的实时曲线图。代码实现片断如下:

```
public void createPartControl(Composite parent) {
    ScheduledExecutorService scheduler;
    // 创建需要定期执行的任务, 任务内部完成实时曲线的重绘
    TimerTask task = new TimerTask(parent);
    // 创建唯一的调度对象 ScheduledExecutorService
    if(scheduler == null) {
        scheduler = Executors.newScheduledThreadPool(1);
    }
}
```

```
// 从 PreferenceStore 中获取刷新时间
int update = store.getInt(PreferenceConstants
                        .P_ANALOG_DISPLAY_UPDATE);
// 在 0ms 后执行第一次刷新视图任务，以后每隔 update 毫秒刷新一次
scheduler.scheduleAtFixedRate(task, 0, update, MILLISECONDS);
}
```

5.3.5.3 双缓冲解决画面闪烁

定期刷新视图在调用 canvas 的 redraw 方法时会出现画面的闪烁，带来不流畅的视觉感，双缓冲技术使用内存缓冲区可以解决由多重绘制操作造成的闪烁问题。

双缓冲技术的工作原理：当启用双缓冲时，所有绘制操作首先呈现到内存缓冲区，而不是屏幕上的绘图图面。所有绘制操作完成后，内存缓冲区直接复制到与其关联的绘图图面。因为在屏幕上只执行一个图形操作，所以消除了由复杂绘制操作造成的图像闪烁。

SWT 中双缓冲的实现较为简单，构建 canvas 时设置画布特征为双缓冲即可，SWT 中提供了内部的实现细节。代码实现片断如下：

```
Canvas canvas = new Canvas(composite, SWT.DOUBLE_BUFFERED);
```

5.4 本章小节

本章首先从总体上分析了 CoreSight 跟踪数据前台展示的基本体系架构，并依照体系架构对前台展示进行了功能模块的划分以及各个模块的简要说明。

接着，给出了显示数据的格式说明，并分析了显示数据的传递过程。

最后，本章着重讨论了 Analog 展示部分的设计与实现，Analog 展示以实时曲线图的形式提供对 Analog 数据变化趋势的直观描述，包括 Analog display 视图、存储管理 Analog 显示数据的模型、相关的首选项设置和其他配置等几个部分。其中涉及到一些关键技术的研究：利用 ScheduledExecutorService 完成视图的定期刷新，利用双缓冲解决重绘时的画面闪烁问题等。

第 6 章 测试与分析

测试作为软件开发过程中的重要一环，能够即时发现软件的缺陷，为提高软件产品的质量提供了可靠性的保证。Event Viewer 系统的测试主要包括单元测试、集成测试、系统测试以及功能测试。其中功能测试为黑盒测试，由专门的测试人员负责，他们担当着类似于第三方客户的角色，从软件产品的界面、架构出发，依照系统的功能需求，编写出来对应的测试用例，输入数据在预期结果和实际结果之间进行评测。

本文对 Event Viewer 系统的测试主要针对基于 ITM 数据流的解析模块的测试，包括解析器单元测试和解析模块集成测试。

6.1 解析器单元测试

本节以 TPIUSG 模块中 ITMParser 测试为例详细介绍解析器单元测试的流程，其中用到了 JUnit 测试工具来实现自动单元测试。

ITMParser 中包含三个公共接口方法：setData()，往解析器中送入待解析的字节流数据；getResult()，得到解析后的 ITM 包数据；getBytesConsumed()，得到完成解析的字节流数据长度。ITMParser 单元测试包括对这些接口的测试，最终，通过分析实际输出结果与预期结果是否一致来判断解析器模块内部处理流程的准确性。

JUnit^[38]是一个开发源代码的 Java 回归测试框架，用于编写和运行可重复的测试。在 Eclipse 中开发、运行 JUnit 测试相当简单。因为 Eclipse 本身集成了 JUnit 相关组件，并对 JUnit 的运行提供了无缝的支持。它包括以下特性^[39]：用于测试期望结果的断言；用于共享共同测试数据的测试工具；用于方便的组织和运行测试的测试套件；图形和文本的测试运行器。

使用 JUnit 对 ITMParser 进行自动单元测试的流程^[40]如下：

1) 创建测试用例 ITMParserTest，继承自 TestCase 类，覆写 setUp 和 tearDown 方法。其中，setUp 方法用来在测试开始前建立测试所需的测试环境，而 tearDown 方法用来在测试后销毁该环境。

```
public class ITMParserTest extends TestCase{
```

```

private ITMParser itmParser;
// 调用工厂方法构建一个 ITMParser 解析器对象
public void setUp() {
    itmParser = (ITMParser)ParserFactory.createParser(
        ParserFactory.ParserType.ITM);
}
public void tearDown() {}
}

```

2) 在 ITMParserTest 中添加自己的测试方法, JUnit 推荐的做法是以 test 作为待测试的方法的开头, 这样这些方法可以被自动找到并被测试。接下来在测试方法内部调用 itmParser 的相关公共接口方法传入字节流数据, 获得解析结果; 并使用 JUnit 提供的相应的断言方法对实际输出结果与预期结果进行比较评判。

ITMParserTest 中添加了对应不同 ITM 包的测试方法, 其中测试 SIP 包的内部代码片断如下:

```

public void testSIP() {
    // 待解析的字节数组, 预期的解析结果为 3 个 SIP 包和 3 个空闲包
    byte[] b = new byte[] {
        (byte) 0x01, (byte) 0x89, // SIP 包, 2 字节长, port = 0
        (byte) 0x70, // 空闲包
        (byte) 0xFA, (byte) 0x79, (byte) 0xE2, // SIP 包, 3 字节长, port = 31
        (byte) 0x70, // 空闲包
        // SIP 包, 5 字节长, port = 6
        (byte) 0x33, (byte) 0x89, (byte) 0x00, (byte) 0x90, (byte) 0x70,
        (byte) 0x70, // 空闲包
    };
    // 将字节数组送入 itmParser 进行解析
    itmParser.setData(b);
    // 调用 itmParser 方法获取完成解析的字节数据长度
    int count = itmParser.getBytesConsumed();
    // 使用断言方法比较完成解析的字节数据长度是否与预期值一致
    assertEquals(count, b.length);
}

```

```

// 调用 itmParser 方法获取解析结果
ITMPacket[] pkts = itmParser.getResult();
// 判断解析结果是否刚好包含 6 个包数据
assertEquals(6, pkts.length);

// 获取第一个包数据, 判断其是否为 SIP 包, 以及包的长度
// 和端口号是否与预期值一致
ITMPacket p = pkts[0];
assertEquals(ITMPacketType.SIP, p.getType());
assertTrue(p.getData() instanceof byte[]);
assertEquals(2, ((byte[]) p.getData()).length);
assertEquals(0, ((byte[]) p.getData())[0]);

//获取第二个包数据, 判断其是否为空闲包
p = pkts[1];
assertEquals(ITMPacketType.Overflow, p.getType());
assertNull(p.getData());
.....
}

```

3) ITMParseTest 测试用例编写完成后, 运行 Run->Run As...->JUnit Test, 右侧会显示测试结果, 如图 6-1 所示。也可以在 ITMParseTest 中写一个 main() 方法以文本运行器的方式运行测试:

```

public static void main(String args[]) {
    junit.textui.TestRunner.run(ITMParseTest.class);
}

```

图 6-1 中所示为运行 ITMParseTest 测试用例后测试正确, 即通过 ITMParseTest 解析后得到的包数据是与预期结果一致的。

如果出现 failure, 表示所测试的产品代码有问题, 也就是受测的产品代码没有正确实现设计上所要求的功能。此时, JUnit 会报告哪一行结果不正确, 双击就可以快速定位到测试失败的方法调用上, 然后通过对产品代码进行检查、修改, 使得它的行为能够符合设计说明书上所预想的情况。而当报告 error 时, 则

可能是测试代码本身有问题，或者系统的运行环境出现了状况。比方说测试代码中所期望的值是错误的——也就是说有可能产品代码所返回的结果是正确的，但是测试代码期望了一个错误的答案；也可能是磁盘已满、网络中断等等外部环境失败所带来的影响。一般情况下，如果 JUnit 测试后报告有若干 failuer、若干 error，我们应该首先查找产生 error 原因，并且加以修复。在修复 error 之后，重新运行 JUnit 进行测试，如果没有出现 error 的话，我们再着手开始调查、修复 failure。

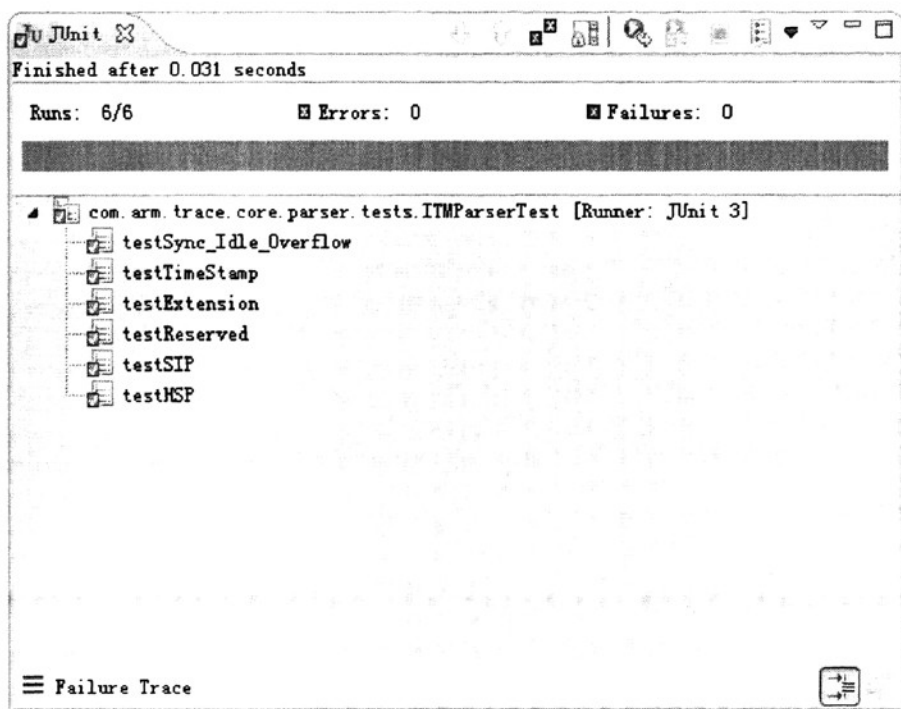


图 6-1 运行 ITMParserTest 测试正确

6.2 解析模块集成测试

在完成对解析器单元测试后，便可将解析器作为正确的组成部分加入到解析模块中，并进行下一步的解析模块集成测试。以 TPIUSG 模块为例，TPIUSG 中包括有同步预处理、帧解析器、包解析器，这些解析器单独测试是正确的，但不能保证连接起来也能正常的工作。因此需要对整个解析模块进行集成测试，

判断在把各个子模块连接起来的时候，穿越模块接口的数据是否会丢失；各个子功能组合起来，能否达到预期要求的父功能。TPIUSG 中的两个接口方法 setDataIn(ISGDataIn)、setDataOut(ISGDataOut)，用来设置模块的数据输入输出，集成测试过程中通过对其内部输入输出接口的调用，传入字节流数据，获取实际数据输出，并与预期结果比较来评判 TPIUSG 模块的正确性。

本文对 TPIUSG 模块的集成测试同样用到了 JUnit 自动化测试工具，测试流程如下：

1) TPIUSG 模块的数据输出接口 ISGDataOut 在系统中的实现是将数据直接送入平台缓冲区内；在测试过程中，为了获取这些数据以便观察，对 ISGDataOut 接口进行了重新实现，提供内部缓冲区存储输出数据。

```
private class SgDataOutForTest implements ISGDataOut {
    ArrayList<IPacketType> types = new ArrayList<IPacketType>();
    ArrayList<byte[]> datas = new ArrayList<byte[]>();
    public void outputData(int id, IPacketType type, byte[] data) {
        if (id == 4) { // 判断是否为 ITM 包数据
            types.add(type);
            datas.add(data);
        }
    }
}
```

2) 在 setUp 方法内建立测试所需的测试环境，包括提供待解析的字节数组和创建 TPIUSG 对象。其中，由于字节数组长度较大，待解析的字节数组通过随机访问文件类 RandomAccessFile 从本地文件中读取；而 TPIUSG 实现了 Runnable 接口，对象创建后需要以线程方式启动。

```
public void setUp() {
    try {
        RandomAccessFile fileIn = new RandomAccessFile(
            "./src/com/arm/trace/core/sg/tests/m3_itmtst.trb", "r"); // 读文件
        int len = (int) fileIn.length(); // 文件包含的字节数
        testData = new byte[len]; // 构建同字节数长度的字节数组
        fileIn.read(testData); // 从文件流读入字节到字节数组 testData 中
    }
}
```

```

        fileIn.close();
    } catch (Exception e) {
        fail(e.getMessage());
    }
    src = new RDSDataOut( null );
    //创建 TPIUSG 对象，并设置数据输入输出接口
    sg = new TPIUSG();
    out = new SgDataOutForTest();
    in = new SGDataIn( null );
    sg.setDataIn(in);
    sg.setDataOut(out);
    //启动 TPIUSG 线程
    runner = new Thread(sg);
    runner.start();
}

```

3) 添加自己的测试方法 `testOutputData`、`testOutputDatas`，将字节数组传入 `TPIUSG` 对象，通过 `SgDataOutForTest` 对象获取输出结果，并使用 `JUnit` 提供的相应的断言方法对实际输出结果与预期结果进行比较评判。其中，`testOutputDatas` 的内部代码片断如下：

```

public void testOutputDatas() {
    // 清空 SgDataOutForTest 对象缓冲区内的数据
    out.types.clear();
    out.datas.clear();
    sg.reset();
    // 传入待解析的字节数组
    src.outputData(testData);
    try {
        Thread.sleep(500);
    } catch (InterruptedException e) {
        fail(e.getMessage());
    }
}

```

```

// 判断输出数据的个数是否为 16
assertEquals(out.types.size(), out.datas.size());
assertEquals(16, out.types.size());
// 判断每个输出数据的类型是否与预期的一致
assertEquals(ITMPacketType.SIP, out.types.get(0));
assertEquals(ITMPacketType.TimeStamp, out.types.get(1));
.....
// 判断每个输出数据的长度和数据内容是否与预期的一致
byte[] buf = out.datas.get(0);
byte[] b = new byte[] { (byte) 0, (byte) 0x48, (byte) 0x65, (byte) 0x6C,
                        (byte) 0x6C, };
assertEquals(b.length, buf.length);
for (int i = 0; i < b.length; i++) {
    assertEquals(b[i], buf[i]);
}
.....
}

```

4) 测试结果。运行 TPIUSGTest 测试用例，同样显示测试正确，即通过 TPIUSG 整个模块解析后得到的输出数据是与预期结果一致的。

6.3 本章小节

本章详细介绍了使用 JUnit 工具构建解析器单元测试和解析模块集成测试的过程，并对测试结果进行了分析，单元测试和集成测试的正确通过也为后续的系统测试提供了基础和保障。

第 7 章 总结

7.1 论文总结

本文在研究和分析 ARM CoreSight 调试体系结构的基础上,结合 ARM 公司 RVDS 集成开发环境中调试模块组成部分 Event Viewer 系统的开发,实现了对通过原始数据源采集到的 CoreSight 跟踪数据的完整实时解析,并最终在显示模块中将其包含的信息以可视化的形式直观地展现给用户,以供后续的程序性能分析和嵌入式软件系统调试。本文所做的主要工作如下:

1) 研究和分析了 ARM CoreSight 调试体系结构、CoreSight 跟踪调试技术及其他常用嵌入式调试技术。

2) 在研究嵌入式集成开发环境国内外现状及其发展趋势的基础上,结合 Event Viewer 系统的整体需求,介绍了系统的总体设计及其功能模块划分,并给出了系统的第三方扩展设计。

3) 在分析 CoreSight 跟踪数据解析流程的基础上,对系统中解析模块进行了详细设计,并完成了基于 ITM 数据流的解析实现。

4) 介绍了 CoreSight 跟踪数据前台展示的基本体系架构和功能模块划分,包括 Text、Event、Analog 三种类型;着重讨论了 Analog 展示部分的详细设计和实现,将解析后得到的 Analog 数据信息以实时曲线图的形式展现给客户,提供对 Analog 数据变化趋势的直观描述。

5) 研究了 Eclipse 开源平台体系架构和插件开发技术,利用 Eclipse 插件扩展点机制,划分解析模块提供对外扩展,实现了系统向第三方产品提供商提供扩展接口的功能,第三方产品提供商可以在此基础上提供自己的解析处理。

7.2 未来工作

Event Viewer 系统是 Eclipse 平台上开发的插件项目,系统实现后,将其插件打包后得到 JAR 文件复制到 eclipse\plugins 目录下,重启 Eclipse 平台即可使用该软件系统来重构目标板程序运行状况,进行性能分析。目前,系统已经完成 1.0 版本的开发,实现了对目标板上跟踪数据进行采集、解析、展示整个流

程，并提供了可供第三方扩展的扩展点接口。在接下来的版本中，还有以下几个方面需要完善：

1) 当跟踪数据流量较大时，前台展示部分每次重绘视图时都需要重新绘制大量数据，特别是视图不停刷新时，会导致 CPU 占有率激增。因此，需要采取更有效的绘制方式来提高系统前台展示的性能，提升用户的体验。

2) 系统中的配置模块还没有完全实现，需要进一步的设计和开发，主要包括对配置向导中各模块配置细节方面的取舍。

3) 系统提供的可供第三方扩展的扩展点接口，还需要在以后第三方产品提供商开发自己的采集、解析模块时进一步检测其通用性。

参考文献

- [1]Jake Chen.ARM 又一秘密武器 RVDS 4.0 专业版横空出世, 支持多核 Cortex-A9.电子系统设计.http://www.ed-china.com/ART_8800029829_400004_500012_HN_6401eb43.HTM.
- [2]ARM.RVDS——ARM 系统开发的首选工具.IQ, Spring 2007:31~35.
- [3]风河风河领军 Eclipse 联盟设备软件开发项目[J].工业控制计算机, 2005,18(6).
- [4]<http://www.embedinfo.com/chinese/product/ide/main.asp>.
- [5]庖啸, 陈书明.面向多核片上 Trace 数据流合成的队列调度算法设计及实现.计算机研究与发展,2008,45(3):417~427.
- [6]李宁,宋微,周薇.嵌入式开发工具发展趋势.单片机与嵌入式系统应用,2008,12:5~8.
- [7]RealView Development Suite 入门指南.<http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0255hc/BGBBFDEJ.html>.
- [8]ARM.RealView MDK——引领 ARM 开发工具新潮流.IQ, Spring 2008:65~69.
- [9]张群忠.ARM 嵌入式系统调试技术研究及实现:[硕士学位论文].上海:华东师范大学信息学院计算机科学技术系,2006.
- [10]王玥.嵌入式 SoC 可调试设计的研究:[硕士学位论文].浙江:浙江大学通信与信息系统专业,2007.
- [11]钟耿.嵌入式系统片上调试研究:[硕士学位论文].浙江:浙江大学通信与信息系统专业,2008.
- [12]Stallman R, Pesch R, Shebs S.Debugging with GDB,Ninth edition[M].Free Software Foundation,Boston,USA.2004.
- [13]Chen HM, Kao CF, Huang IJ.Analysis of Hardware and Software Approaches to Embedded In-Circuit Emulation of MicroProcessors[A].In Proceedings of the Seventh Asia-Pacific Computer Systems Architecture Conference(ACSAC'2002)[C].Melbourne, Australia. 2002. 127~133.
- [14]J.Huang, T.A.Lu. ICEBERG:an embedded in-circuit emulator synthesizer for microcontrollers[J].In Proceedings of the 36th ACM/IEEE conference on Design automation.pp. 580~585,1999.
- [15]A.B.T.Hopkins, K.D.McDonald-Maier.Debug support strategy for systems-on-chips with multiple processor cores[J].IEEE Transactions on Computers.vol.55, pp.174~184,2006.
- [16]K.D.Maier.On-chip debug support for embedded Systems-on-Chip[J].In Proceedings of the 2003 International Symposium on Circuits and Systems, ISCAS'03.vol.5, 2003.

- [17]N.Stollon, R Leatherman, B Ableidinger, etal.Multi-core embedded debug for structured ASIC Systems[J].In Proceeding ofDesignCon.2004.
- [18]S.Tang, Q.Xu. A multi-core debug platform for NoC-based systems[J].In Proceedings of the conference on Design, automation and test in Europe.PP.870~875, 2007.
- [19]MIPS Technologies.The PDtrace Interface and Trace Control Block Specification[OL].
- [20]Hopkins ABT,McDonald-Maier KD.Debug Support Strategy for Systems-on-Chips with Multiple Processor Cores[J].IEEE Transactions on Computers.2006-02.55(2):174~184.
- [21]Orme W, ARM 公司 . 建立多核 SoC 中的 Debug 和 Trace 标准 .
<http://article.ednechina.com/2006-07>.
- [22]Vranken H.Debug Facilities in the TriMedia CPU64 Architecture[J].Journal of Electronic Testing.2000.16(3):301~308.
- [23]刘洪星,谢玉山.Eclipse 开发平台及其应用[J].武汉理工大学学报:信息与管理工程版,2005,27(2).
- [24]Gamma E, Beck K.Contributing to Eclipse:Principles, Patterns, and Plug-ins[M].Addison Wesley,2003.
- [25]张云涛,龚玲.Eclipse 精要与高级开发技术.北京:电子工业出版社,2005.
- [26]陈刚.Eclipse 从入门到精通(第 2 版).北京:清华大学出版社,2007.
- [27]Tony Armitstead.Event Viewer Product Specification System Design Division.ARM,2007,12.
- [28]丁言华.RealView Event Viewer 系统设计报告.深圳英蓓特信息技术有限公司,2007,12.
- [29]ARM Limited.CoreSight Components Technical Reference Manual,2004.12-2~12-8.
- [30]阎宏.Java 与模式.北京:电子工业出版社,2002.
- [31]Eric Clayberg,DanRubel.Eclipse 插件开发.北京:人民邮电出版社,2006.
- [32]甘树满,王秀明.Eclipse 插件开发方法与实战.北京:电子工业出版社,2006.
- [33]孙卫琴,李洪成.Tomat 与 Java Web 开发技术详解.北京:电子工业出版社,2004.233~234.
- [34]孙卫琴.精通 Struts:基于 MVC 的 Java Web 设计与开发.北京:电子工业出版社,2004.9~13.
- [35]Jim D'Anjou,Scott Fairbrother.Eclipse 权威开发指南(第 2 版).束尧,丁凡,许国梁,译.北京:清华大学出版社,2006.
- [36]那静.Eclipse SWT/JFace 核心应用.北京:清华大学出版社,2007.257~261.
- [37] Bruce Eckel. Thinking in Java. Pearson, 2005.
- [38]Vincent Massol, Ted Husted.JUnit IN ACTION.鲍志云,译.北京:电子工业出版社,2005.
- [39]Andrew Hunt,David Thomas.单元测试之道 Java——使用 JUnit.电子工业出版社,2005.
- [40]王东刚.软件测试与 JUnit 实践.人民邮电出版社,2004.

致谢

在硕士学位论文完成之际，我想向曾经给过我帮助和支持的人们，表示衷心的感谢。

首先要感谢我的导师张能立老师，他在学习和科研方面给了我很大的帮助。感谢他为我提供了良好的科研环境，让我能够有机会接触实际的开发项目。张老师一丝不苟的作风，严谨求实的态度，踏踏实实做事的精神，不仅授我以文，而且教我做人，让我终生受益无穷。我对张老师的感激之情是无法用言语表达的。在此，祝愿张老师身体健康，全家幸福！

感谢在武汉理工大学英蓓特嵌入式研发中心实习的日子里，给予我很大帮助的李宁、库少平、方安平老师，他们是我在学习专业知识和编写项目代码的良师益友，从他们那里得到很多宝贵的建议，他们是我学习的指路灯。

感谢 Eclipse 开发小组的项目经理丁言华，软件工程师傅翔和周智猛，感谢他们在项目开发过程中给予我的耐心帮助。

感谢我的师兄姐们，同时感谢我的同学郭慎平，高庆和刘立，在与他们的工作交流中我学习到了很多的知识。感谢他们在这三年来对我的友爱和帮助。

感谢生我养我的父母，他们给了我无私的爱，我深知他们为我求学所付出的艰辛和汗水，而我至今仍无以为报。祝他们幸福、安康！

我的朋友们也在默默支持着我，化解我的失落和悲伤，分享我的成功和喜悦。还有很多师长和友人，他们给予了我很多指导和帮助，在此衷心的感谢，他们的名字我一直铭记在心！

最后，衷心感谢在百忙之中抽出宝贵时间审阅本论文的专家教授。

攻读硕士学位期间发表的论文

[1]王阳赞,张能立.Eclipse 插件扩展点技术的研究及其在嵌入式调试工具开发中的应用. 软件导刊, 2008. 10

作者: [王阳赞](#)
学位授予单位: [武汉理工大学](#)

相似文献(10条)

1. 学位论文 [谢楷 基于嵌入式系统的通用电子测量仪器硬件平台的研究](#) 2006

嵌入式系统是一种具有特定功能的计算机系统,它与通信技术和网络技术的结合,极大的增强了设备的网络和通信的灵活性和智能性。随着信息技术的不断发展和用户需求的不断增长,嵌入式系统逐渐走进国民生产的方方面面,其应用也日益广泛。在此基础上,将嵌入式系统融入到测控仪器设备

的开发无疑将增强测控仪器的通信、交互、数据处理等能力,并扩展其使用范围。
本文旨在提出一种基于32位嵌入式系统的通用的仪器仪表平台构架。该平台以S3C2410为核心,拥有丰富的系统资源,包括2410核心系统、具有高分辨率触摸屏的人机交互设备、大容量Flash存储卡和硬盘接口、USB主从接口、100M以太网口、VGA接口、音频接口等,可以满足各种仪器仪表的交互式操作、存储、通讯的要求。本文详细介绍了各个功能模块的硬件实现方法。

本文讨论并制定了仪器模块的接口方案,在主系统上可插接各种测量模块而构成不同功能的仪器。并实现以下高端仪表中所必需的功能模块:校准用高精度基准源的产生模块、6位半数字电压表模块、阻抗特性分析仪模块、数字存储示波器模块、微电容测量模块等、VXI接口模块、GPIB接口模块等。并将部分模块产品化。

2. 学位论文 [黄樱 嵌入式系统中USB HOST技术研究](#)与实现 2008

USB是现今在PC领域被广泛采用的总线接口技术,在一些嵌入式系统中,人们也希望有USB设备的出现,而在USB的拓扑结构中,USB设备无法脱离居于核心地位的PC机而存在。本文针对USB设备的应用局限,对USB协议架构及其通信流程做了深入的分析,确定嵌入式USB主机必须具备的软硬件条件,给出设计思路。并在此基础上,以应用最为广泛的U盘为USB设备,设计并实现了基于USB Host接口芯片SL11HS,可以与U盘进行通信的通用USB Host嵌入式系统模块(USB Host Demo)。这种模块实现了USB Host在嵌入式系统中的应用,从而使得嵌入式设备能够脱离PC,直接与传统的USB外设进行通信。在方案中选中了应用广泛且利于调试的AT89C55单片机作为假想嵌入式系统的核心MCU,设计并验证了整个模块;采用中断任务调度机制设计软件系统,并设计了一套API,使得整个模块的通用性提高,能方便的移植到不同处理器的嵌入式系统中,间接提高存储容量,应用到不同的场合。

按照论文中提出的方法,实现了USB主机系统的构建,对开发嵌入式USB主机,使USB应用脱离PC具有普遍意义。

3. 学位论文 [胡淑军 基于MPC8270的嵌入式系统研究和低功耗设计](#) 2008

MPC8270处理机属于Freescale公司的PowerQUICC II系列嵌入式通信处理机家族中MPC8280处理器系列。它是目前网络和通信领域应用非常广泛的一款嵌入式通信处理机。本文对基于MPC8270的嵌入式系统的开发进行了较为详细和深入的研究,同时针对MPC8270的特点,研究了基于MPC8270嵌入式系统的低功耗设计并把它用于系统的硬件设计中。

本文的主要内容包括:研究了MPC8270嵌入式通信处理器的基本组成结构,分析了MPC8270的内部功能模块,为以后基于MPC8270的研究和设计奠定了基础;阐述了嵌入式系统低功耗研究的意义、功耗产生的原因和低功耗设计的一般原则,并针对MPC8270的特点,提出了基于MPC8270的嵌入式系统的低功耗设计技术并为系统的硬件平台选定了具体的低功耗器件;根据所选定的硬件元器件和低功耗设计的基本原则进行了基于MPC8270的嵌入式系统的硬件设计,包括硬件的概要设计和详细设计;对VxWorks嵌入式实时操作系统、板级支持包BSP和系统的启动流程进行了分析,并进行了基于MPC8270的BSP的开发,给出了硬件平台初始化、串口驱动程序和网口驱动程序这三个部分的具体设计与实现,重点对串口驱动程序与网口驱动程序的设计进行了深入研究。最后,对基于MPC8270的硬件平台和BSP进行了测试,并给出了整个系统的预估功耗和实际功耗测试结果。

4. 学位论文 [李泉泉 基于嵌入式Linux和MiniGUI的数控加工功能模块研究](#)与开发 2006

本文对基于嵌入式Linux和MiniGUI的数控加工功能模块进行了研究。文章对RTLlinux的原理和特征进行了分析,总结了RTLlinux环境的关键编程技术。依据开放式数控系统的模块划分方法,设计并实现了模块间的缓冲机制;根据开放式数控系统的时序设计原则,设计了RTLlinux环境中的数控系统时序实现的具体方案。在搭建系统架构的基础上,编程实现了主控、界面、译码、插补等模块。

5. 期刊论文 [陆洲章 基于多线程技术嵌入式导航功能模块研究](#) -大众科技2008(11)

从软件上提高导航效率是当前的导航系统开发技术的关键。文章针对嵌入式系统的运行效率受设备自身处理器和内存资源的限制,研究实现了由用户界面主控线程和一些功能模块(如导航位点、路径规划和地图数据库查询)用工作者线程在后台异步运行的多线程技术方案。

6. 学位论文 [冯宏伟 基于多性能指标评价的SoC软硬件划分方法研究](#) 2008

嵌入式系统的诞生,标志着一个新的时代的到来。目前,嵌入式系统已被广泛地应用于包括航空航天、武器装备、通信设备、工业控制、个人电子产品等在内的各个领域,形成了巨大的产业。随着嵌入式系统与微电子技术的飞速发展,硬件的集成度越来越高,这使得将CPU,存储器和I/O设备集成到一个硅片上成为可能,SoC应运而生,并以其集成度高、可靠性好、产品问世周期短等特点逐步成为当前嵌入式系统设计技术的主流。传统的嵌入式系统设计开发方法中的弊端使其已无法满足当代SoC设计的特殊要求,这给系统设计人员带来了巨大的挑战和机遇,因此针对SoC的嵌入式系统设计方法学已经成为当前研究的热点课题,嵌入式软硬件协同设计方法学应运而生。而软硬件划分又是软硬件协同设计的重要环节,因此,研究SoC设计中的软硬件划分方法,找到一种合理的系统描述模型,提出划分算法并对其进行优化改进,将有十分重要的理论及应用价值。

本文针对存在多种因素影响嵌入式系统综合性能的实际情况,详细分析了影响嵌入式系统性能的各项性能指标,引入了一种基于多性能指标评价的软硬件协同划分思想。利用SoC可重用的特性,将IP核复用及软件构件重用引入到软硬件划分算法当中。通过功能模块层的抽象,将复杂的嵌入式系统构成映射到理论中的DAG之上。引入了性能指标优先级概念,通过在算法中加入对给定的参数数据预先处理及运筹学中分支定界的思想,对条件遍历算法进行了优化,并通过实验证明了此算法求解收敛速度较之原算法更优。

7. 学位论文 [冯博 基于展讯平台的移动终端应用软件的设计与实现](#) 2008

手机作为移动终端产品的典型代表,给人们的生产生活提供了极大的便利。其中人机交互接口(MMI)所反映出来的界面美观度和操作的简易性极大的影响了移动终端产品的市场占有率。

本文首先说明目前手机发展现状和课题背景,提出了课题的研究意义及主要任务。基于手机嵌入式系统介绍了应用软件开发平台(SAP),并以一个手机应用的典型-电话簿为例,对人机交互界面部分(MMI)进行软件架构再设计的同时,又深入到了应用层软件功能模块的具体实现,完成了数据结构定义,应用间的交互设计;后期对手机自动化测试的应用作了一定研究,提出一个可行的自动化测试方案,并以实例对其可行性做出验证。

本课题主要进行手机应用软件的设计,总结出的手机平台MMI二次开发的经验,具有一定的通用性和参考价值。

8. 学位论文 [李金山 基于Nios II的VGA控制器设计与实现](#) 2007

作为软核CPU的Nios II处理器,固化在FPGA内部,Nios II软核占用一定的片内资源,实现强大的CPU功能。相对于常见的硬核CPU,Nios II软核的成本更低,灵活性更高,实现更简易,而且开发更快速。基于Nios II的嵌入式系统可应用领域很广,包括无线通信、医疗器械、交通、消费电子、工业控制、军事和航空航天等。

对于VGA显示模块来讲,通常需要一个控制其显示的控制器。一般都采用外接控制器的方法,但随着嵌入式系统的发展,对体积和功耗的要求也越来越高,用IP核的方式来实现VGA控制器要比外接控制器的方法减小更多的体积和功耗。

本论文的VGA控制器是以IP核的形式开发,然后挂载到Avalon总线上同其它外设进行通信。首先,本文对VGA的原理、技术组成和Avalon总线的读写时序进行详细的介绍。接着,对VGA控制器IP核所要实现的功能进行了分析和总体描述,并采用自上而下的设计流程对系统功能进行了模块的划分和功能实现。功能模块主要包括,时序发生器模块、DMA控制器模块、寄存器模块、颜色处理器模块、FIFO模块、仲裁模块和Avalon总线接口模块等。本文对各

个模块进行了结构分析和接口描述。

在文章的最后总结了一些在设计过程中遇到的一些问题，以及一些设计心得，并对设计中的不足和今后需要完善的地方进行了说明。

9. 会议论文 [李巍, 贾克斌 MPlayer播放器在嵌入式平台上的应用](#) 2007

通过对MPlayer播放器进行分析和研究，设计出一种适用于嵌入式系统的便携式媒体播放器，并成功应用到三星ARM9嵌入式平台上。针对播放器整体结构设计、数据处理流程以及分流器，音、视频解码，音、视频同步等各个关键功能模块都作了详细的描述，并给出了详细的移植过程。

10. 学位论文 [邵明 嵌入式数控系统及其功能模块的研究](#) 2008

与工控机的CPU相比，基于RISC技术的32位ARM9微处理器具有功耗小、成本低、可靠性高、主频可达200M以上等优点。ARM微处理器已广泛应用于消费电子、通信系统等类产品。然而，在机床数控领域，ARM9微处理器的成熟应用至今还较少。实际上，由ARM微处理器构成的嵌入式系统以其优越性能，在机床数控领域有着广阔的应用前景。为此，本文在ARM9内核的微处理器S3C2410硬件平台的基础上，进行了嵌入式数控系统功能控制核心单元模块的研究。

在分析数控系统体系结构和嵌入式PLC控制技术的基础上，提出了一种嵌入式PLC系统的开发方法。嵌入式PLC系统是嵌入式数控系统中功能控制系统的核心单元模块。根据PLC指令系统的结构特点，采用链表数据结构来存储PLC指令文件，逐行提取指令逐行解析来实现对PLC基本指令系统的解释。还设计了基于ARM9的CAN总线通讯系统并应用于数控系统中，实现数控系统的信息传输。论文最后对伺服驱动模块中的关键技术(PWM逆变)进行了研究，给出了SVPWM调制算法和高频调制方波的实现算法。系统最终采用了MATLAB仿真的方法得出了结果。

本文链接: http://d.g.wanfangdata.com.cn/Thesis_Y1475168.aspx

下载时间: 2010年5月25日