文章编号:1671-4598(2025)05-0329-08 DOI:10.16526/j.cnki.11-4762/tp.2025.05.039 中图分类号:U666.1 文献标识码:A

星载混合异构处理器在轨重构方法

蒋孝勇[1]，周 军[1]，吴侃侃[1]，胡秀清[2]，汪少林[1]，
王 闯[1]，刘 波[1]，史亚菲[1]，杨 勇[1]
（1. 上海卫星工程研究所，上海 201109；2. 中国气象局国家卫星气象中心，北京 100081）

摘要：针对星载边缘计算算力紧张，算力资源复用难，星载混合异构在轨编程复杂等难题，提出基于CPU和操作系统管理协处理器模式的异构在轨编程方法，灵活使用文件系统版本管理维护，解决面向任务的异构计算在轨编程，既能重构CPU主控制器又能重构协处理器；为验证方法可行，基于ZYNQ处理器实现气象卫星星载边缘计算任务重构，完成异构计算在轨重构方法验证，实现一套硬件同时满足极轨卫星和高轨卫星边缘计算需求；算法验证为气象卫星光学相机逐像元定位定标算法，包括CPU端APP和协处理器加速算法，极轨气象卫星和高轨气象卫星混合异构算法均满足在轨实时处理要求，混合异构在轨编程方法真实可用，实现算力资源复用功能。

关键词：在轨重构；混合异构；星载边缘计算；FPGA管理器

On-orbit Reconstruction Method for Hybrid Heterogeneous Processors on Satellite

JIANG Xiaoyong[1], ZHOU Jun[1], WU Kankan[1], HU Xiuqing[2], WANG Shaolin[1],
WANG Chuang[1], LIU Bo[1], SHI Yafei[1], YANG Yong[1]
(1. Shanghai Institute of Satellite Engineering, Shanghai 201109, China;
2. National Center for Space Weather, China Meteorological Administration, Beijing 100081, China)

Abstract: On-board edge computing has the problems of limited computing power, difficulty in resource reuse, and complication in on-orbit programming of hybrid heterogeneous spacecraft, an on-orbit heterogeneous programming method based on CPU and operating system processing modes is proposed. This method flexibly uses file system version management and maintenance, solves the problem of task oriented heterogeneous computing in on-orbit programming, and reconstructs both the CPU main controller and co processors. To verify the feasibility oh this method, on-board edge computing for meteorological satellites based on ZYNQ processor is reconstructed, and heterogeneous computing on-orbit reconstruction method is verified. simultaneously meeting the edge computing needs of polar orbit satellites and high orbit satellites. This algorithm is validated for pixel by pixel positioning and calibration algorithm of meteorological satellite optical cameras, including the CPU side APP and co processor acceleration algorithms and hybrid heterogeneous algorithms, all of which meet the requirement of on-orbit real-time processing, realizing the real usability of hybrid heterogeneous on-orbit programming methods and the reuse of computing power resources.

Keywords: on-orbit reconstruction; hybrid heterogeneous; on-board edge computing; FPGA manager

0 引言

随着星载计算资源不断升级，芯片算力逐年提升，卫星使用对在轨编程的需求已由传统的CPU修漏洞打补丁的重构升级为异构计算下的任务重构、算法模型重构。混合异构星载边缘计算处理器包括CPU、FPGA、GPU、NPU等，在轨计算模式由多种处理器相互协同实现，算法任务一般由CPU作为主控芯片与GPU等协处理器共同完成，星载边缘计算重构模式由单纯的重构APP升级为多种处理器混合重构。人工智能兴起背景下迫切要求混合异构处理器具备AI模型、异构计算算子在轨升级能力。

针对有限的星载边缘计算资源，灵活方便的在轨任

收稿日期:2024-03-28； **修回日期**:2024-05-06。
作者简介:蒋孝勇(1992-),男,工程师。
引用格式:蒋孝勇,周 军,吴侃侃,等.星载混合异构处理器在轨重构方法[J].计算机测量与控制,2025,33(5):329-336,360.

务重构方法有利于实现多载荷多任务对算力资源的有效复用。本文基于 ZYNQ 处理器和气象卫星在轨预处理算法，结合下一代气象卫星星座高低轨协同观察需求，实现一套星载边缘计算硬件平台同时满足极轨卫星和高轨卫星星上处理需求。高低轨协同观测卫星星座能够实现高轨卫星引导低轨卫星观测，观测结果返回到高轨卫星进行星载边缘计算。因此星载边缘计算需要满足高低轨不同载荷运算需求，针对有限边缘计算算力资源，采用软件重构和在轨编程方式能够满足高低轨协同观测需求。

1 星载混和异构需求分析

近年来航天遥感数据出现了爆发式发展，卫星遥感数据应用也迎来了新的发展时代。传统卫星作为传感器端只产生遥感数据，通过卫星数传分系统将遥感数据传到地面，由地面进行数据分析再将分析结果传递给用户使用。面临的问题是用户对遥感数据使用时效性不强，遥感信息由于链路冗长，信息处理不及时导致遥感数据“不好用、不能用”。星上处理技术的发展为大量遥感数据在轨实时处理提供了新的解决思路，遥感卫星不再是传统的传感器端，星载边缘计算实现卫星在轨数据处理，用户地面算法上天，遥感信息结果数据直接落地，大大提升卫星应用效能。

AI 算力正逐年攀升，芯片水平也随之芯片工艺能力提升得到飞速发展。目前星载边缘处理算法水平已经达到几十 TOPS，且在逐年上升，使得算力水平不再是制约星载边缘计算发展的瓶颈。星载边缘计算架构也由单一的 CPU 架构发展到 CPU 加协处理器的混合异构处理架构。协处理器种类众多，包括：FPGA、GPU、NPU、DPU 等多种形式，是星载数据加速处理的重要硬件资源。

混合异构星载边缘计算作为星上实时处理关键组成部分是星上处理算法的载体，星载混合异构在轨编程技术能够有效解决星载算力资源紧张情况下的星载计算资源复合应用。混合异构技术重点解决单一处理器无法满足遥感数据在轨实时处理需求情况下采用 CPU 加协处理器加速混合方式达到实时处理目的。

国外混合异构星载边缘处理典型案例如“黑杰克”项目中的 Pitboss 星载处理器，该单机可实现星座内卫星互联互通互操作，支持在轨遥感数据实时处理核心处理器为 ZYNQ。Pitboss 实现星座自主任务规划，星间接力，检测结果直面用户下发等功能。具有小型化、低功耗、可扩展等特点。国内混合异构星载边缘处理在轨较早应用的有长光卫星、武汉大学、长沙天仪研究院等单位。长光卫星公司在其研制的长光卫星中应用了 NVIDIA-GPU 处理器可实现空中飞机的实时监视。混合异构星载边缘计算的广泛使用必然需要相应的混合异构在轨软件升级维护技术。

星载边缘计算是遥感卫星应用的接口，在轨编程更新是星上处理软件在轨迭代的关键技术。星载边缘计算在轨编程包括 CPU 编程和协处理器编程，目前单独的 CPU 在轨编程、FPGA 在轨编程基本具备，但混合异构在轨编程尚处于发展阶段，并没有形成统一方法。操作系统作为混合异构在轨计算的重要组成部分承担着硬件资源管理维护功能。基于操作系统的文件系统可以实现在轨编程软件版本管理，包括 CPU 端的 APP 算法软件管理和协处理器加速算法软件管理。

2 混和异构在轨重构流程

单独在轨重构 CPU 技术和在轨重构 FPGA 技术在航天领域已普片应用，将 FPGA 作为 CPU 的硬件加速协处理器，在轨管理并维护 FPGA 比特流文件以满足多种载荷在轨数据处理需求技术未见报道。由 CPU 实现 FPGA 在轨管理不仅需要动态加载 FPGA 比特流文件，还需要动态加载更新相应的设备树文件，以满足上层应用软件调用 FPGA 数字电路。CPU 和 FPGA 采用 AXI 总线数据交互。

根据任务需求每次在轨重构包括 3 个文件：CPU 端的应用 APP、FPGA 端的比特流文件、设备树文件、设备驱动文件。CPU 应用 APP 为 C 语言端的应用程序，FPGA 端的比特流文件为硬件加速的 Verilog 数字电路，设备树文件为 CPU 调用数字电路的设备接口 ID、设备驱动文件实现 CPU 对数字电路业务和管理操作。每次重构 FPGA 比特流文件均需要重构设备树文件和驱动文件。FSBL 启动文件、u-boot 文件、操作系统 kernel、文件系统以及混合异构管理 APP 不需要重构，保证混合异构设备任何情况下均有遥测下传，保证星载设备安全。

CPU 端的 APP 重构比较简单，通过 shell 脚本运行固定目录下的重构后 ELF 文件实现，FPGA 端的比特流文件和设备树文件重构方法为本文重点。

2.1 ZYNQ 加载启动流程

混合异构 ZYNQ 芯片包括 PS 端和 PL 端，PS 端为混合异构 CPU 端，架构为 ARM，包含 4 个 A53 处理器，PL 端为混合异构 FPGA 端，包含 Kintex-7 逻辑编程处理器。

在轨真实情况下启动镜像和代码存储在 QspiFlash 和 NandFlash 中。FSBL、u-boot、FPGA.bin（黄金版本）、kernel 存放在 QspiFlash 中。文件系统和部分依赖库、配置参数存放在 NandFlash 中。ZYNQ 启动过程如图 1 所示。

ZYNQ 在轨为安全启动模式，ROM 中的自举启动程序上电自动运行，将 FSBL 从 QspiFlash 读入片上存

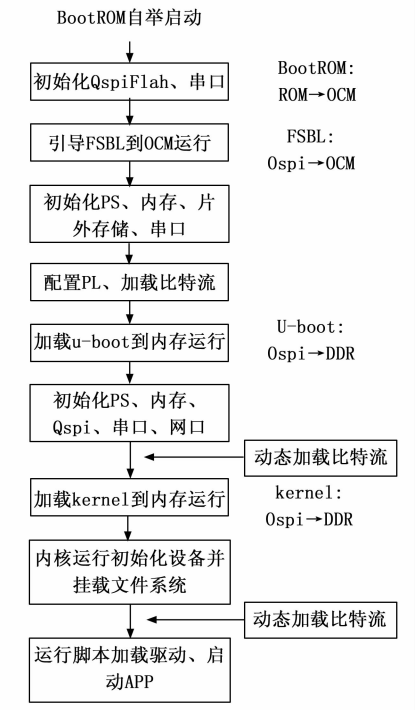

图 1　ZYNQ 启动流程

储 OCM 中运行，并对内存、网口、串口、Flash 进行初始化。FSBL 运行后引导启动 u-boot，u-boot 引导启动 kernel，Linux 启动完成后挂载文件系统并自运行管理软件。ZYNQ 启动流程中有三次加载 FPGA 比特流的机会，FSBL 引导加载、u-boot 引导加载、Linux 引导加载。FSBL 加载为自动引导加载，u-boot 加载采用 shell 命令加载（FPGA 命令），Linux 加载采用 shell 命令加载（fpgautil 命令）。本文采用 FSBL 加载和 Linux 加载，FSBL 完成初始黄金版本加载，Linux 完成 FPGA 在轨重构加载。

2.2　PL 端重配置 DevC

在轨重构 PL 需要由 PS 端对 PL 端进行配置，配置依赖于设备配置单元 DevC。DevC 提供连接到 AES、HMAC 和 PCAP 模块的接口，用于实现对 ZYNQ 内 PL 的配置及镜像数据的解密。AES 和 HMAC 具有解密功能，ZYNQ 从外部 Flash 中恢复及解密 PS 镜像，同时使用 PL 内的 AES 和 HMAC 硬件进行认证。AXI-PCAP 桥实现 PS 端和 PL 端的链接，将 32 位 AXI 格式的数据转换为 32 位的 PCAP 协议。数据搬运由 PCAP 中的 DMA 引擎实现。配置流程如图 2 所示。

配置流程包括：1）BootROM 完成自举，确认 PL 上电，开始解密 FSBL；2）BootROM 使用 DevC DMA 引擎，通过 PCAP 将加密的 FSBL 发送到 PL 内的 AES 和 HMAC 进行解密；3）PCAP 将解密后的 FSBL 返回到 PS 端的 OCM 中执行；4）FSBL 使用加密比特流配置 PL。此过程完成 FSBL 阶段黄金版本的 FPGA 比特流配置。

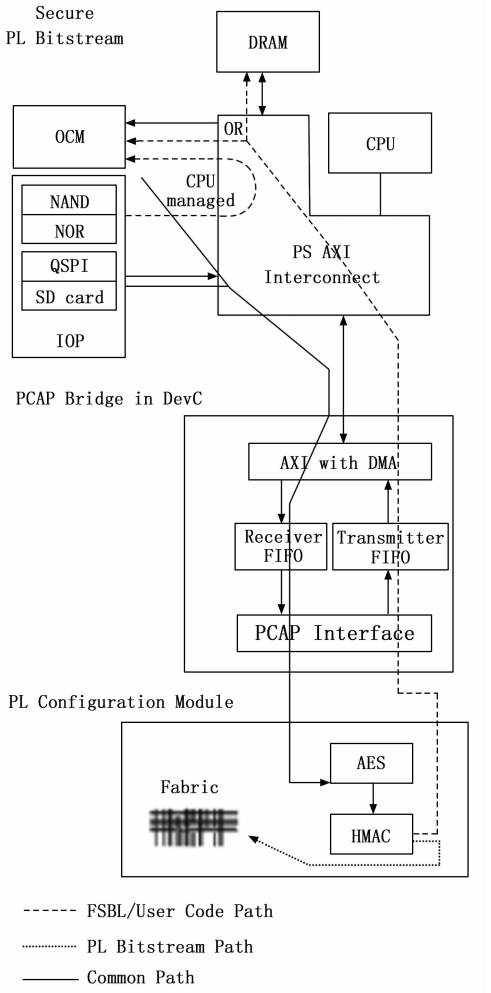

图 2　ZYNQ 安全启动流程

PL 端的重配置由 Linux 下的 DevC 驱动实现 PCAP 对 PL 端的重配置，重配置后由 Linux 下驱动程序调用 DevC DMA 引擎将重配置比特流文件发送到 PL 端，实现 PL 端在轨重构。

2.3　Linux 下 FPGA Manager 配置

在轨编程跟新依靠 Linux 操作系统实现，设备配置单元 DevC 作为底层硬件支撑，Linux 下的 DevC 驱动由 FPGA Manager Driver 实现。FPGA Manager 框图如图 3 所示。

DevC 驱动对应的设备树节点如下所示，Linux 启动后自动加载 DevC 设备树节点运动相应驱动对 DevC 进行初始化。

遥感卫星混合异构在轨重构 PL 端数字电路作为 Linux 上层应用 APP 的协处理器设备，协处理器设备需要编写额外的驱动，根据遥感数据处理算法不同，配置不同数字电路和相应驱动程序。动态加载 FPGA 比特流文件时需要动态加载设备树文件，向 Linux 用户层注册设备，方便应用层 APP 调用协处理数字电路。动

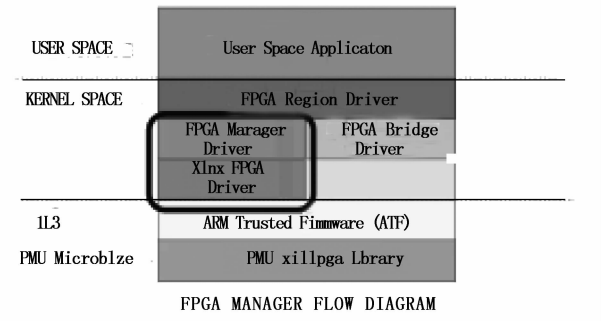

图 3 FPGA Manager 系统框图

态加载设备树时，自动加载协处理数字电路驱动。

DevC 设备树节点文件：

```
devcfg: devcfg@f8007000 {
    compatible = "xlnx,zynq-devcfg-1.0";
    interrupt-parent = <&intc>;
    interrupts = <0 8 4>;
    reg = <0xf8007000 0x100>;
    clocks = <&clkc 12>, <&clkc 15>, <&clkc 16>, <&clkc 17>, <&clkc 18>;
    clock-names = "ref_clk", "fclk0", "fclk1", "fclk2", "fclk3";
    syscon = <&slcr>;
};
fpga_full: fpga-full {
    compatible = "fpga-region";
    fpga-mgr = <&devcfg>;
      address-cells = <2>;
      size-cells = <2>;
};
```

2.4 FPGA 协处理器驱动动态加载

无 FPGA-manager 配置时，设备树节点随 kernel 一起编译加载，在/proc/device/amba _ pl@0/节点下含有 pl.dtsi 下的全部设备节点。具有 FPGA-manager 后，需要动态添加协处理器设备节点。FPGA 协处理器驱动使用 Device-Tree-Overlay 动态加载。在 FPGA-manager 模式下，需要动态添加设备节点文件 pl.dtbo，加载成功后在 Linux 文件系统相应文件节点/proc/device/amba/下生成新的协处理器设备节点。Overlay 设备树文件如下所示。

Overlay 设备树节点文件：

```
/dts-v1/;
/plugin/;
/ {
    fragment@0 {
        target = <&fpga_full>;
        #address-cells = <1>;
        #size-cells = <1>;
        __overlay__ {
        #address-cells = <1>;
        #size-cells = <1>;
        firmware-name = "design_1_wrapper.bit.bin";
      };
    };
  fragment@1 {
        target = <&amba>;
        __overlay__ {
          axi_gpio_0: gpio@a0000000 {
            #gpio-cells = <2>;
            compatible = "xlnx,xps-gpio-1.00.a";
            gpio-controller ;
            reg = <0x0 0xa0000000 0x0 0x10000>;
            xlnx,all-inputs = <0x0>;
            xlnx,all-inputs-2 = <0x0>;
            xlnx,all-outputs = <0x1>;
            xlnx,all-outputs-2 = <0x0>;
            xlnx,dout-default = <0x00000000>;
            xlnx,dout-default-2 = <0x00000000>;
            xlnx,gpio-width = <0x8>;
            xlnx,gpio2-width = <0x20>;
            xlnx,interrupt-present = <0x0>;
            xlnx,is-dual = <0x0>;
            xlnx,tri-default = <0xFFFFFFFF>;
            xlnx,tri-default-2 = <0xFFFFFFFF>;
          };
        };
    };
};
```

设备树节点添加完成后，添加协处理器驱动，驱动文件协处理器.ko，在 Linux 操作系统下通过 Insmod 命令添加。添加成功后在/dev/文件下生成供 APP 调用的协处理器设备名。

3 硬件平台搭建及工程创建

为验证混合异构在轨编程方法的可行性，搭建如下图所示硬件平台。核心处理器为 Xilinx ZU9EG-fbV1156，处理器内存 PS 端 2 GB，PL 端 2 GB。Qspi-Flash64 MB 用于存储启动镜像，NANDFlash 为 4 GB 用于存储文件系统。PS 和 PL 之间通过 AXI 总线连接，遥感数据通过 DMA 实现 PS 与 PL 间交互，ZYNQ 对外主要接口包含网口、RS422。通过 Petalinux 操作系统裁剪编译工具定制 Linux 操作系统。在轨编程各类型软硬件通过文件系统进行管理维护。

ZYNQ 在轨重构信息流入如图 4 所示。交换板通过 422 接收综电在轨重构上注包，并通过网络送给 ZYNQ 板，ZYNQ 板接收到在轨上注包后，依靠 ubifs 文件系统替换 nand 分区中的相应在轨上注文件。在轨重构包通过星上高速上行链路上注给星上综合电子单机，高速

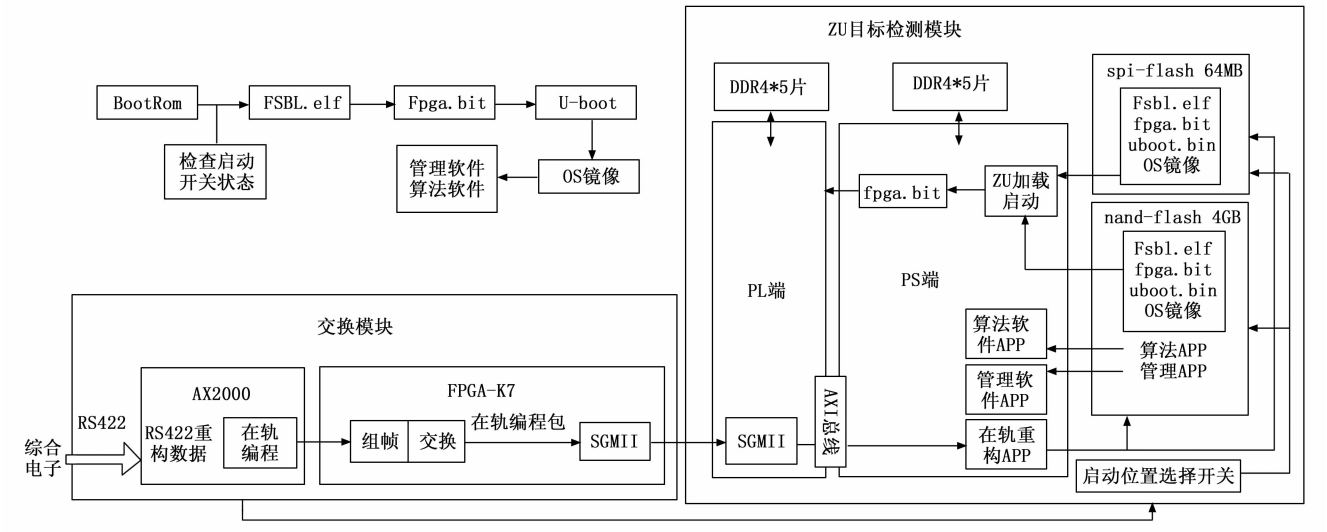

图 4　在轨编程信息流框图

上注码速率为 2 MB/秒，卫星过境测控链路时长为 10 min，可满足混合异构处理器在轨更新需求。

在轨编程依托 linux 文件系统，版本维护和管理依靠文件系统下不同文件路径。Flash 中存储原始黄金版本，默认启动黄金版本，且黄金版本在轨不修改。ZYNQ 模块启动方式中由交换模块中的反熔丝 FPGA 控制，通过指令设置启动位置选择开关，选择 QSPI 启动或 NAND 启动，两种启动方式互为备份增加系统可靠性。

算法数据验证采用风云卫星在轨真实数据和真实码速率进行验证。高轨卫星 FY-4 载荷速率按照 2.048 毫秒每 17 028 字节，极轨卫星 FY-3 载荷速率按照 1.5 秒每 372 862 8 字节。功能完成极轨气象卫星和高轨气象卫星光学相机图像预处理，完成遥感图像定位定标在轨实时处理功能。其中数据调度算法依靠 PS 端的 ARM 实现，图像加速算法依靠 PL 端的协处理数字电路实现。依靠在轨编程完成高轨卫星和低轨卫星数据处理切换。

硬件平台处理验证基于 Xilinx 处理芯片，建立 Vivado 工程，构建协处理器 IP 核及相关驱动。基于 Vivado 工程生成的硬件描述文件 hdf 文件构建 Petalinux 工程，裁剪编译形成 Linux 操作系统并烧录到 Qspi-Flash 中。

Linux 环境下编译并烧录文件包括 FSBL.elf、FPGA.bin、u-boot.bin、image.ub、rootfs.tar.gz。可动态加载文件包括 FPGA.bin、设备树文件 pl.dtbo、协处理器驱动文件 dma-proxy.ko 文件、应用程序 APP.elf。FPGA.bin 文件由 vivado2019.1 直接生成。pl.dtsi 文件由 vivado2019.1-SDK 生成，对 pl.dtsi 文件进行编辑，添加 DMA 相应设备节点，并用 Petalinux 工程中的/scripts/dtc/dtc 工具编译，生成 pl.dtbo 文件。设备驱动文件在 Petalinux 工具中编辑并随操作系统一起编译，生成到相应文件系统中。SDK 编写数据预处理 APP。

4　软件流程设计

图 5 为混合异构硬件加速原理框图。混合异构处理模式下，遥感数据处理算法既包含 PS 端的 APP 算法也包括 PL 端的硬件加速 IP 核。APP 和加速 IP 核均需要在轨重构，硬件加速 IP 核通过 APP 端的 API 接口驱动函数调用，常用驱动为 DMA 驱动和 UIO 驱动。DMA 驱动向协处理器 IP 核输入遥感图像数据，UIO 驱动为协处理器 IP 核初始化提供接口函数。

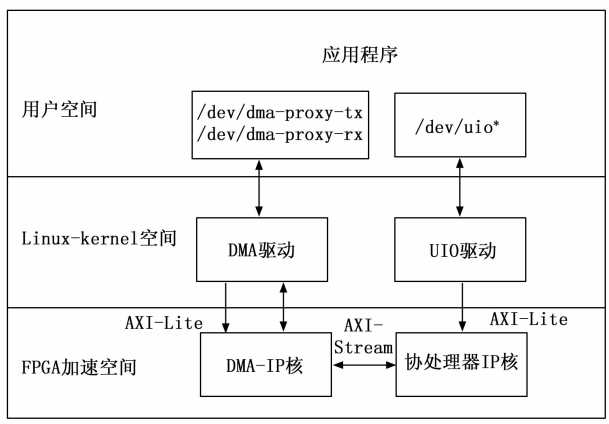

图 5　预处理硬件加速驱动使用框图

协处理器 IP 核可以由 Xlinx 公司提供的 HLS 工具将仿真完善的 C 程序转化为 veilog 程序并封装成标准接口的 IP 核。自定义 IP 核采用的控制接口为 AXI-Lite，数据流接口为 AXI-Stream。调用自定义 HLS-IP 核时采用 UIO 的方法，uio 全称为用户空间 IO（Userspace I/O），是一种在用户空间编写设备驱动程序的框架。

表 1 中为混合异构在轨编程系统中的全部软件配置项。软件设计工作包括 Vivado 软件工程、Petalinux 软件工程、驱动软件设计和 SDK 软件设计。本文针对下表的软件配置项进行相应软件设计。

表 1 混合异构在轨编程软件配置项

序号	配置项名称	配置项可执行文件	软件工具
0	Fsbl. elf	zynqmp_fsbl. elf	Petalinux
1	FPGA. bit	design_1_wrapper. bit. bin	Vivado2019. 1
2	u-boot	Boot. BIN	Petalinux 2019. 1
3	kernel	image. ub	
4	设备树文件	pcw. dtsi	
		pl. dtsi	
		zynqmp. dtsi/zynqmp-clk-ccf. dtsi	
		system-top. dts	
		system-conf. dtsi	
		system-user. dtsi	
5	Dma 驱动	Dma-proxyin. ko	
6	文件系统	rootfs. tar. gz	
7	测试 APP	APP. elf	SDK2019. 1

在轨编程更新中主要针对遥感数据处理算法更新，操作系统等基础服务软件在轨不做业务相关更新，只做可靠性维护更新。FSBL. elf 文件、uboot 文件、kernel、文件系统和驱动文件都由 petalinux 工具编译。

4.1 Vivado 软件工程设计

图 6 为混合异构硬件工程图。Vivado 为 Petalinux 提供硬件配置信息，硬件信息对应配置文件为 . hdf 文件。

在 Vivado2019. 1 软件中新建 ZYNQ 工程。在添加 DMA 模块和 HLS 自定义模块时需要将两个 IP 和的中断信号引用到 ZYNQ 中，如图 6 所示，在设置 DMA 核时需要勾选 SG。PL 端输出时钟频率选择 200 MHz。ZYNQ 端通用的接口选择包括网口、串口。DMA 的地址位宽选择为 64 bit，数据位宽选择 32 bit。

FPGA. bit 文件由 Vivado 软件编译得到，作为在轨更新硬件加速部分软件。Petalinux 在编译完成后可将 FPGA 的 bit 文件一起打包进去 BOOT. bin 文件，编译指令为：petalinux-package --boot --fsbl-fpga ～/work/petalinux/ * * * * 09/images/linux/system. bit --u-boot --force

通过 JTAG 仿真器烧录到 ZYNQ 的 qspi-flash 中，在 fsbl. elf 启动之后加载一起打包在 BOOT. bin 中的 FPGA-bit 文件。

4.2 Petalinux 软件工程设计

Petalinux 工具是 Xilinx 提供的 ZYNQ 混合异构操作系统裁剪编译工具，关键配置包括 Uboot 配置、kernel 配置、文件系统配置。

4.2.1 U-boot 配置

U-boot2019. 1 中的 image. ub 存放在 qspi-flash 中。u-boot 启动后自动加载 qspi-flash 中 kernel 区域的 image. ub 镜像文件。Qspi-flash 分区如上所示。QSPI-flash 的分区通过 petalinux-config 进行设置。nand-flash 的分区设置通过设备树文件进行设置，在 system-user. dtsi 文件中进行分区设置。

```
0. boot:0x0000000-0x1FC0000;
    1. bootenv:0x1FC0000-0x2000000;
    2. kernel:0x20000000-0x3E80000;
    3. bootscr:0x3E80000-0x4000000;
```

U-boot2019. 1 使用的制作工具为 petalinux2019. 1。Petalinux2019. 1 编译完成后，通过打包工具（petalinux-package --boot --fsbl --fpga --u-boot --force）生成 BOOT. bin，并通过 vivado 和仿真器将 BOOT. bin 烧入 qspi-flash 中。也可以将 image. ub 和 BOOT. bin 一起打包烧写入 QSPI-flash 中，或者通过 tftp64 网络将 image. ub 传入 ZYNQ-DDR，然后使用 sf 命令，写入 qspi-flash 固定区域。烧写 qspi-flash 命令行如下所示。

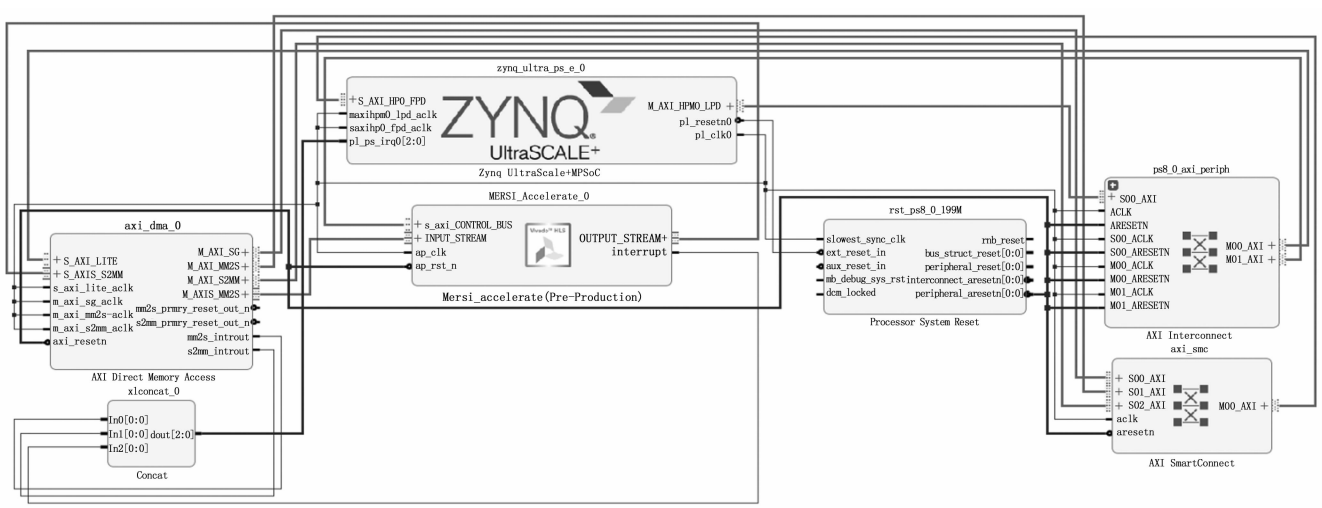

图 6 vivado 工程框图

```
image. ub 烧写入 qspi-flash
```

```
tftpboot image. ub
    sf probe
    sf erase 0x2000000 0x1b00000
    sf write 0x10000000 0x2000000 0x1b00000
```

4.2.2　kernel 配置

Kernel 制作工具采用 petalinux2019.1，在制作 kernel 镜像时使用的都是内存文件系统。但是 kernel 的文件系统挂载位置可以通过 u-boot 中的 bootargs 环境变量参数传递到 kernel 内核中。Bootargs 可以设置挂载的文件类型和文件位置。如下所示：文件类型为 ubifs，挂载文件系统的位置为 nand-flash 的 rootfs 分区：bootargs＝console＝ttyPS0，115200n8 noinitrd ubi. mtd＝1 rootfstype＝ubifs root＝ubi0：rootfs rw init＝/sbin/init。

在 petalinux-config 设置中 image 的文件类型中选择 other 类型，这样通过 u-boot 传递 rootfstype＝ubifs root＝ubi0：rootfs 参数时，就可以将文件系统类型配置为 ubifs 型。在 u-boot 运行启动后，通过 tftp64 将 image. ub 传入 ZYNQ-DDR 中，通过 sf、nand 命令将 kernel 写入存储器相应的分区中。

4.2.3　文件系统配置

本文制作的文件系统存储在 nand-flash 中，文件系统的类型为 ubifs 类型，文件系统文件为 rootfs. tar. gz。如使用 Ubuntu 文件系统，可以直接通过 ubuntu 网站下载编译。

表 2　设备树加载命令行

命令行	说明
cat /proc/mtd mtd0：04000000 00100000 "kernel" mtd1：3c000000 00100000 "rootfs" mtd2：04000000 00100000 "fpga"	//查看 mtd 128MB 存放备用 kernel 用于存放 ubuntu 文件系统 用于存放 FPGA-bit 文件
ubiformat /dev/mtd1 ubiattach /dev/ubi_ctrl -m 1 ubimkvol /dev/ubi0 -N rootfs -m mount -t ubifs /dev/ubi0_0 /mnt	格式化 mtd1　　首次挂载 将 mtdblock1 与 ubi 建立连接 ubi2 上创建一个 volume，rootfs 为 bootargs 中 root＝ubi0_0：rootfs 的“rootfs”，名字不一致会出现 VFS 挂载根文件系统失败错误 将 nand 挂载到/mnt 目录下，
ubiattach /dev/ubi_ctrl -m 1 mount -t ubifs /dev/ubi0_0 /mnt	第二次挂载 将 nand 挂载到/mnt 目录下。

将文件系统烧入 nand 中，需要通过 kernel（RAM 型）启动成功后，将 rootfs. tar. gz 通过网络传入 ZYNQ 中，挂载 nand-rootfs 分区，然后将 rootfs. tar. gz 解压到 nand-flash 挂载的分区中。重新启动 ZYNQ 处理板，并在 u-boot 下设置 kernel 的根文件系统类型，启动 kernel 后，自动挂载文件系统。u-boot 的环境变量设置为：bootargs ＝ console ＝ ttyPS0，115200n8 noinitrd ubi. mtd＝1 rootfstype＝ubifs root＝ubi0：rootfs rw init＝/sbin/init。需要在 linux 系统下，对 nand：rootfs 分区进行格式化、网路将 ubuntu 文件系统压缩包传入 linux 系统、解压 rootfs. tar. gz 到 nand 分区。

5　测试及实验结果分析

根据前几章卫星混合异构星上实时处理软件在轨编程方法原理分析，本章对设计结果进行真实数据实验验证。主要测试混合异构在轨编程方法的功能性能指标。首先动态极轨卫星图像预处理算法，测试完成后动态加载高轨卫星图像预处理算法。

5.1　动态加载测试步骤

将 pl. dtbo 和 design _ 1 _ wrapper. bit. bin 文件通过网络拷贝到/lib/firmware/目录下，如果 lib 下没有 firmware 文件。则使用 mkdir -p /lib/firmware 添加文件夹。表 3 中的命令行用于加载 pl. dtbo 文件。

表 3　设备树加载命令行

	命令
0	mkdir -p /lib/firmware
1	echo 0 ＞ /sys/class/fpga_manager/fpga0/flags
2	mkdir /configfs
3	mount -t configfs configfs /configfs
4	cd /configfs/device-tree/overlays/
5	mkdir full
6	echo -n "pl. dtbo" ＞ full/path
7	insmod /lib/modules/4.19.0-xilinx-v2019.1/extra/dma-proxy.ko
8	rmdir full
9	mkdir full1
10	echo -n "pl. dtbo" ＞ full1/path

动态加载成功后串口打印信息如图 7 所示，自动生成 pl. dtbi 添加的协处理器设备节点。协处理器驱动加载成功，打印信息如图 8 所示，根据设备树下的节点在/dev/下生成驱动对应的设备名称供 APP 调用。比特流文件和驱动加载完成后通过网口添加相应测试 APP 软件。

5.2　实验结果

实验测试过程中严格按照载荷实际码速率测试，通

```
root@imageubuntu3711:/proc/device-tree# cd amba
root@imageubuntu3711:/proc/device-tree/amba# ls
#address-cells                  dma@a0004000                    dma@ffae0000
#size-cells                     dma@fd4c0000                    dma@ffaf0000
CTRL_50_REG@a0001000            dma@fd500000                    dma_proxy1024@0
PERIPHERAL@ff380000             dma@fd510000                    dma_proxy512@0
PERIPHERAL@ff990000             dma@fd520000                    dma_proxyin@0
afi0                            dma@fd530000                    dma_proxyout@0
ahci@fd0c0000                   dma@fd540000                    ethernet@ff0b0000
ams@ffa50000                    dma@fd550000                    ethernet@ff0c0000
can@ff060000                    dma@fd560000                    ethernet@ff0d0000
can@ff070000                    dma@fd570000                    ethernet@ff0e0000
cci@fd6e0000                    dma@ffa80000                    gpio@ff0a0000
compatible                      dma@ffa90000                    gpu@fd4b0000
ddr4@400000000                  dma@ffaa0000                    i2c@ff020000
dma@a0000000                    dma@ffab0000                    i2c@ff030000
dma@a0002000                    dma@ffac0000                    memory-controller@fd070000
dma@a0003000                    dma@ffad0000                    memory-controller@ff960000
root@imageubuntu3711:/proc/device-tree/amba#
```

图 7　设备树和比特流文件动态加载成功信息打印

```
root@imageubuntu3711:/dev# ls
autofs                  ptyd1
block                   ptyd2
btrfs-control           ptyd3
char                    ptyd4
console                 ptyd5
cpu_dma_latency         ptyd6
dma_proxy_rxin          ptyd7
dma_proxy_txin          ptyd8
fd                      ptyd9
```

图 8 协处理器驱动加载成功信息打印

过网口将高低轨载荷源码数据传入混合异构处理板卡内。遥感数据预处理结果采用 matlab 对结果数据进行分析，分析结果如图 9～10 所示。

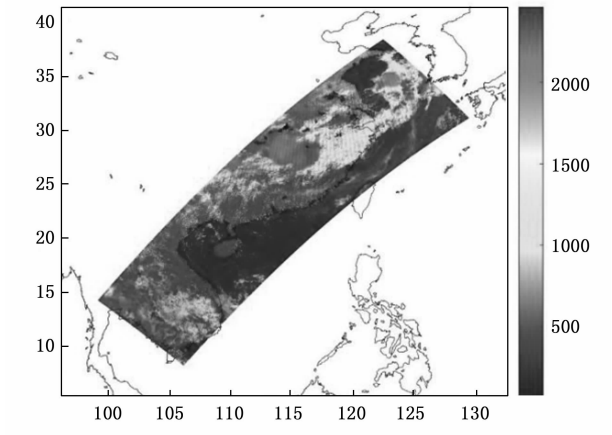

图 9 极轨卫星混合异构处理结果

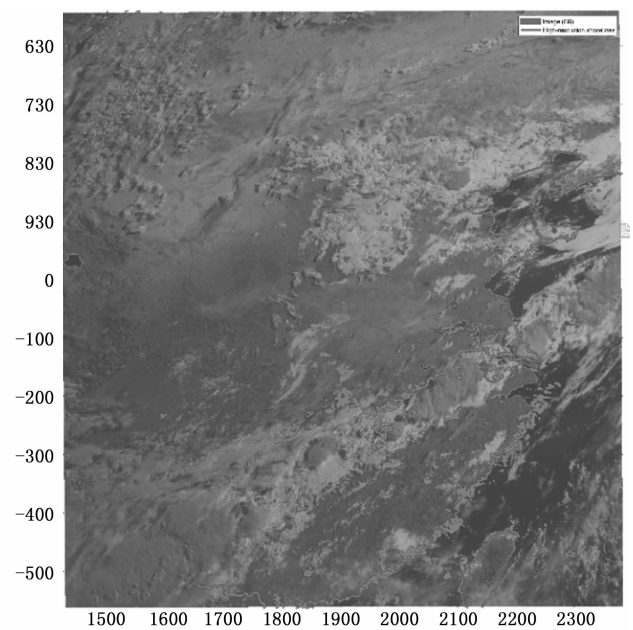

图 10 高轨卫星混合异构处理结果

上图结果显示混合异构在轨编程能够正常运行，极轨卫星和高轨卫星均实现了在轨实时定位定标预处理算法。所设置混合异构在轨编程方式能够满足不同卫星多种载荷多种数据处理算法的在轨编程需要，为后续星载边缘异构计算提供在轨编程服务。

6 结束语

星载边缘计算平台由单纯的 CPU 计算向主处理器加协处理器混合异构模式发展，在轨数据处理需求也由单一 CPU 变成了混合异构处理需求。混合异构在轨编程能力是考核星载边缘计算的关键指标。本文依托风云卫星高低轨协同观测背景，使用相同混合异构硬件平台通过在轨重构方式，实现 FY-3 卫星和 FY-4 卫星在轨数据预处理功能。验证混合异构在轨 APP 和协处理器数字电路同时重构方式。极大提升混合异构处理器在轨运算效能，充分利用星载边缘有限算力。

参考文献：

[1] 钱方亮，林荣锋，周 宇，等. 一种基于微小卫星系统软件在轨编程功能的设计方法 [J]. 计算机应用与软件，2018，35 (12)：16－20.

[2] 李兴伟，白 博，周 军. 基于 FPGA 的立方星可重构星载处理系统研究 [J]. 计算机测量与控制，2018，26 (8)：172－176.

[3] 陆 峥，金 光，杨天社，等. 基于可重构度的在轨卫星多级健康评估方法 [J]. 系统工程与电子技术，2018，40 (8)：1769－1776.

[4] 李丹丹，马金全，杨平平. 信号处理平台中可重构组件的设计与实现 [J]. 计算机工程，2017，44 (5)：33－39.

[5] 汪俊锋，叶函函，易维宁，等. 在轨超高分辨率傅里叶光谱仪仪器线型函数更新方法研究 [J]. 红外与毫米波学报，2018，37 (5)：613－620.

[6] 刘基余. 四大系统的卫星导航电文概论：GNSS 导航信号的收发问题之五 [J]. 数字通信世界，2014 (s1)：1－10.

[7] 汪宏浩，王慧泉，金仲和. 基于增量链接的可回滚星载软件在轨更新方法 [J]. 浙江大学学报 (工学版)，2015，49 (4)：724－731.

[8] 温连强. 卫星光通信系统软件在轨更新策略 [D]. 哈尔滨：哈尔滨工业大学，2012.

[9] 韦涌泉，董振辉，张红军. 一种基于文件的嵌入式星载软件在轨升级方法 [J]. 单片机与嵌入式系统应用，2018，18 (5)：32－35.

[10] 史毅龙，薛长斌. 基于“龙芯”的 VxWorks 系统函数在轨更新研究 [J]. 电子设计工程，2015，23 (21)：106－109.

[11] 王战强，翟盛华. 星载处理设备软件在轨重构技术研究 [J]. 空间电子技术，2013，10 (1)：7－13，43.

[12] 张 衡，顾泽凌，杨明远，等基于 1553B 的 DSP 在轨编程设计与实现 [J]. 制导与引信，2020，12 (4)：33－37

[13] 王元乐，杨玉辰，方火能，等. 一种可重构的星载高性能智能异构计算系统 [J]. 空间控制技术与应用，2022，48 (5)：125－132.

(下转第 360 页)