

FPGA 运行时重构的延迟隐藏机制研究与实现

刘 伟¹, 柴 镇¹, 周浩杰², 吴 东², 柴志雷^{1,2}

(1 江南大学 物联网工程学院, 江苏 无锡 214122;

2 数学工程与先进计算国家重点实验室, 江苏 无锡 214125)

摘 要: 设计并实现了一种 FPGA 运行时重构的延迟隐藏机制, 通过在 FPGA 中进行页面管理、缓存架构设计、缓存策略研究, 在多个任务运行时调度中, 可以对用户隐藏 BIT 文件配置的延迟, 提高了总体运行效率. 从而使 FPGA 具备了和 GPU 一样的用户模式, 为 FPGA 更广泛地用于计算领域提供了技术思路.

关键词: FPGA; 运行时重构; 延迟; 开发方式

中图分类号: TP302.1

文献标识码: A

文章编号: 1000-7180(2016)08-0040-05

Research and Implementation of Delay Hidden Mechanism for FPGA Runtime Reconfiguration

LIU Wei¹, CHAI Zhen¹, ZHOU Hao-jie², WU Dong², CHAI Zhi-lei^{1,2}

(1 School of Internet of Things, Jiangnan University, Wuxi 214122, China;

2 State Key Laboratory of Mathematical Engineering and Advanced Computing, Wuxi 214125, China)

Abstract: This paper designs and realizes a delay hidden mechanism for FPGA runtime reconfiguration. Through the pages management of FPGA, the cache architecture design and caching policy research, the mechanism can hid the configuration delay of BIT file, improve the overall efficiency in the multiple tasks scheduling. It gives FPGA the same user mode with GPU, provides the idea of using FPGA in computing technology more widely.

Key words: field programmable gate arrays (FPGA); runtime reconfiguration; delay; development mode

1 引言

当前,随着大数据及人工智能的快速发展,信息呈爆炸式增长. 巨大的信息量对处理器的计算性能和低功耗有了更高的需求. FPGA (Field Programmable Gate Array) 支持细粒度的并行方式和流水操作,可以高效处理大批量的数据,是常用的硬件加速器^[1].

传统的 FPGA 开发主要采用硬件描述语言 HDL (Hardware Description Language) 进行硬件设计,所有的通信、外设都需要开发人员自行设计. 这样的开发方式开发和调试难度大.

目前 Xilinx 推出的 HLS (High-Level Synthesis)^[2] 工具和 Altera 推出的 OpenCL (Open Com-

puting Language)^[3] 提升了编程抽象层次,很大程度上简化了 FPGA 的编程设计.

但是,新的开发工具仍然没有解决 FPGA 设计成果的跨平台性和可扩展性比较差的问题. 特别是,当使用的资源超出芯片的资源时,往往只能通过更换更大的芯片或者重新修改设计来保证功能的实现. 这些问题很大程度上制约了 FPGA 的应用和发展.

FPGA 上的运行时可重构技术,即 PR (Partial Reconfiguration)^[4] 技术允许 FPGA 在运行过程中切换逻辑,运行过程中的逻辑调度可以弥补 FPGA 资源难以扩展的劣势. 然而 PR 技术支持的逻辑切换的延迟远远大于 GPU (Graphics Processing Unit)^[5] 替换内核的时间,以 Zynq 7030 为例,切换一

次逻辑耗时在 100 ms 以上. 切换的延迟影响了运行时的重构.

本文设计和实现了基于 PR 技术的延迟隐藏机制,解决逻辑切换时延迟过大的问题,最大程度发挥 PR 技术对 FPGA 逻辑扩展的能力,让 FPGA 在编程上具备和 GPU 一样的优势.

2 PR 机制

PR 技术为 FPGA 提供了一块特殊区域,允许 FPGA 在运行过程中替换这个区域的逻辑而不影响其他逻辑的正常运行. 使用 PR 技术主要有以下三大优点:通过分时复用的方式节约硬件资源;保持成熟的设计不变,只修改部分逻辑,加快开发效率;只替换需要的逻辑部分,运行过程中加快了逻辑功能的切换速度^[6].

图 1 所示是 PR 机制的示意图. 在 FPGA 上划分出一个可重构的块 A. 运行过程中可以使用多个部分的 BIT 文件,如 A1. bit、A2. bit 等对块 A 进行配置. A1. bit、A2. bit、A3. bit 和 A4. bit 分别包含不同的算法,通过对 BIT 文件的配置完成算法的切换. 在配置 BIT 文件的过程中,除了可重构块 A 以外其他的 FPGA 资源都保持不变,可持续运行而不受配置影响.

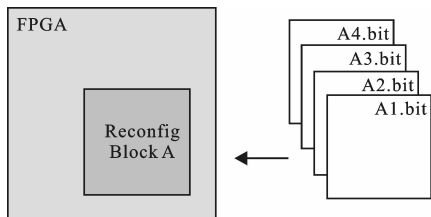


图 1 PR 机制示意

PR 机制为 FPGA 上的算法调度提供了可能,使 FPGA 可以像 GPU 一样通过替换内核完成整体算法的实现,很大程度上扩展了 FPGA 的资源 and 灵活性. 然而, BIT 文件配置的时间比较长,远大于 GPU 替换内核的时间. 这个问题影响了 FPGA 运行时的重构.

3 延迟隐藏机制的设计与实现

3.1 硬件设计

传统 PR 技术的使用方式是在 FPGA 上划分一个专用的可重构区域 A 用来完成算法的切换. 这种方式决定了算法切换的过程只能串行实现,切换过程中产生的配置 BIT 文件的时间将对运行效率产

生很大的影响.

如图 2 所示,为了隐藏配置 BIT 文件的延迟,考虑在 FPGA 上划分两个可重构区域 A、B,两个可重构区域交替完成配置 BIT 文件和执行算法的过程,以乒乓操作的形式完成 FPGA 上算法的切换. A、B 两个区资源十分相近,每个区都作为一个独立的逻辑“页面”.

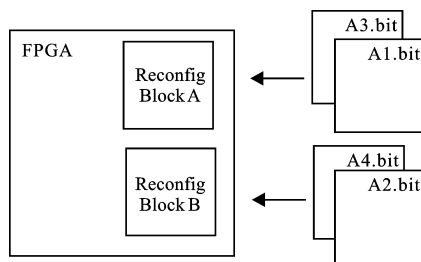


图 2 延迟隐藏机制下的 PR 机制示意

3.2 时序设计

图 3 所示是采用延迟隐藏机制和传统机制下的 FPGA 算法切换过程对比. 上下两图分别表示传统机制和延迟隐藏机制下分时段 FPGA 上的操作. 两种机制都需要配置一个区的 BIT 文件作为初始操作. 传统模式采用配置和执行串行的方式,每执行一次算法切换就需要一个配置的延迟;而延迟隐藏机制下两个分区的执行和配置交替进行,运行时间只包含算法执行时间而不存在配置的延迟.

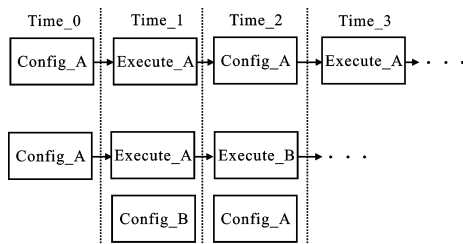


图 3 延迟隐藏机制对比传统机制

延迟隐藏机制下,同一时间需要完成 A、B 两个区域的配置和执行. 为了实现延迟隐藏机制,提高整体运行效率,需要协调算法执行和 FPGA 配置的时间. 由于绝大部分算法执行时间远小于配置时间,延迟隐藏机制中设计了数据缓存机制以保证算法执行缓存的批量数据的时间和配置时间协调.

对于视频处理,设摄像头采集模块采集的两帧间隔为 T_{camera} ;VGA 显示的两帧间隔为 T_{VGA} ;算法整体处理批量数据的时间为 $T_{\text{com_total}}$;算法各部分处理批量数据的时间分别为 $T_{\text{com_1}}, T_{\text{com_2}} \dots T_{\text{com_m}}$;配置 FPGA 的时间是 T_{config} ;缓存帧数为 N_{frames} ;算法分

段数为 $M_{\text{algorithm}}$.

摄像头采集模块采集速度和 VGA 显示速度一致,算法处理时间不能大于摄像头采集时间,因此:

$$T_{\text{com_total}} < N_{\text{frames}} * T_{\text{camera}} = N_{\text{frames}} * T_{\text{VGA}} \quad (1)$$

从算法部分分析,同一个时段内 A 区和 B 区分别进行 FPGA 配置和算法的执行,时间由二者中较长的决定. 因此有:

$$T_{\text{com_total}} = \max(T_{\text{com_1}}, T_{\text{config}}) + \max(T_{\text{com_2}}, T_{\text{config}}) + \dots + \max(T_{\text{com_m}}, T_{\text{config}}) \quad (2)$$

将(2) 式带入(1) 式,有:

$$N_{\text{frames}} * T_{\text{camera}} > \max(T_{\text{com_1}}, T_{\text{config}}) + \max(T_{\text{com_2}}, T_{\text{config}}) + \dots + \max(T_{\text{com_m}}, T_{\text{config}}) \quad (3)$$

假设算法处理足够快,与配置时间相比是很小的值,即 $T_{\text{com_max}} \ll T_{\text{config}}$, 有下列关系:

$$N_{\text{frames}} * T_{\text{camera}} > T_{\text{config}} * M_{\text{algorithm}} (T_{\text{com_max}} \ll T_{\text{config}}) \quad (4)$$

由于配置时间固定,而算法处理速度很快,因此为了保证不丢帧和高速处理,只需增加缓存的帧数即可.

而当算法处理时间较长,远大于配置时间,即 $T_{\text{com_min}} \gg T_{\text{config}}$, 则需要调整帧频来配合算法的运行时间. 帧频和计算时间需要满足以下关系:

$$N_{\text{frames}} * T_{\text{camera}} > T_{\text{com_1}} + T_{\text{com_2}} \dots + T_{\text{com_m}} \quad (5)$$

由于算法运行很慢,只能通过降低摄像头帧频或者丢帧来满足上述关系. 而对于静态数据的处理,只需保证算法执行一个批量的数据的时间接近配置时间即可.

3.3 硬件实现

如图 4 所示,以 Xilinx xc7z030fbg484-3 芯片为例,基于延迟隐藏机制在 FPGA 上划分出两个可重构区域 A 和 B. 两个区域内的物理资源可以在运行过程中进行算法功能切换,用以实现算法不同部分的功能. 其余部分资源完成了摄像头采集、VGA 显示、总线协议以及划分区域内的 IP 核接口等.

表 1 列出了静态逻辑部分使用的资源以及 A、B 两个可重构区域内主要的逻辑资源数量和在总资源中所占比重. 在进行算法 IP 核设计时,IP 核划分出的多个功能模块所用资源必须在每个分区提供的资源以内. 当资源超出时,应考虑将算法划分为更多部分.

延迟隐藏机制下,算法 IP 核在进行切换时需要和系统部分以及存储器之间进行数据交换. 图 5 所示为 IP 核与外围的通信结构. 算法 IP 核通过 FIFO 与 AXI42AXIS 和 AXIS2AXI4 两个 IP 核进行交

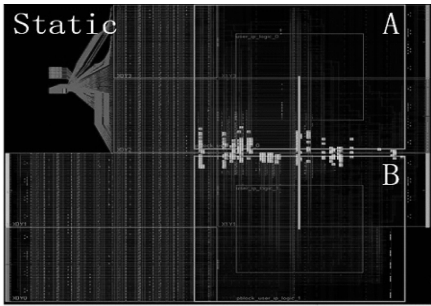


图 4 可重构区域划分

表 1 可重构区域资源

	Static	A	B
FF	17%	29%(46 000)	32%(50 000)
LUT	25%	35%(27 600)	35%(28 000)
BRAM	8%	39%(100)	32%(85)
DSP	8%	30%(120)	20%(80)

互. AXI42AXIS 核以 AXI4 协议^[7]从缓存 Buffer 中获取数据,然后利用 AXI4-stream 协议将数据运输至算法核中;AXIS2AXI4 核利用 AXI4-stream 协议获取算法核中处理完的数据,再用 AXI4 协议转存进 Buffer. A、B 两个分区的 AXI42AXIS 和 AXIS2AXI4 模块通过 FIFO 进行互联,支持两个分区的算法直接进行数据交换.

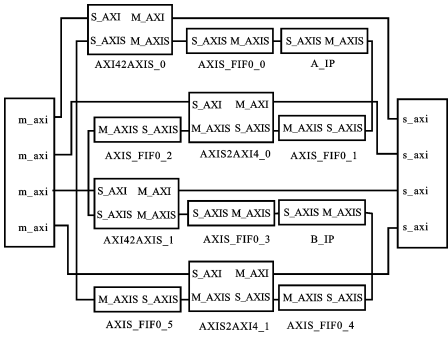


图 5 IP 核外围通信硬件结构

算法 IP 核的 HLS^[8] 接口设计如下:

```
# pragma HLS INTERFACE axis register depth=2048
port=in_pix_stream

# pragma HLS INTERFACE axis register depth=2048
port=out_pix_stream

# pragma HLS INTERFACES_axilite register port =
return offset=0x00 bundle=Ctrl_Bus

# pragma HLS INTERFACES_axilite register port =
rows offset=0x14 bundle=Ctrl_Bus

# pragma HLS INTERFACES_axilite register port =
cols offset=0x1c bundle=Ctrl_Bus

# pragma HLS INTERFACES_axilite register port =
stride offset=0x24 bundle=Ctrl_Bus
```

```
#pragma HLS INTERFACES_axilite register port =  
addr_reserved offset=0xFF0 bundle=Ctrl_Bus
```

3.4 软件实现

软件编程时主要完成 FPGA 的配置,实现软件和硬件的协同. FPGA 上的各个 IP 核通过 AXI4-Lite 协议与 ARM 端进行交互,通过软件编程完成与各个 IP 核的数据及控制信号的传递.

如图 6 所示,静态逻辑的配置和初始化以及块 A 的配置和初始化是软件实现的初始操作,完成初始操作后,块 A 和块 B 的配置、初始化、启动和停止等操作以固定顺序循环执行,完成整体机制.

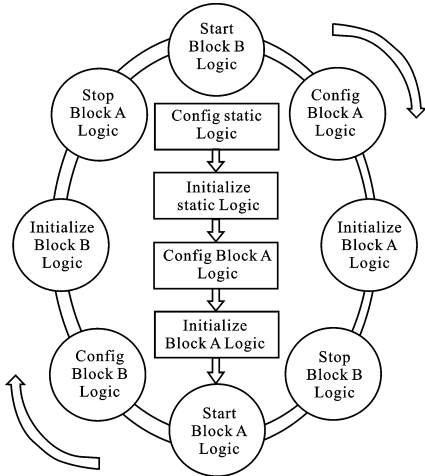


图 6 软件架构及流程

FPGA 的配置使用 load_bitstream 函数,该函数可以完成全 BIT 文件和部分 BIT 文件的 FPGA 配置. write_gpio 函数用于配置参数,可以指定缓存帧数等. Buffer2DDR_initialize 用于配置缓存数据搬运模块,包括数据搬运地址和行列等信息. HW_initialize 用于初始化 IP 核,传递行列等信息. 函数中的参数 ipid 表示块编号. IPcore_Start_Ex 和 IPcore_Stop_addr 两个函数用于控制算法 IP 核的启动和停止.

```
参照图 6 的软件流程,编程如下:  
/* Config Static logic */  
load_bitstream("Static_logic. bin", full_bit);  
/* Initialize Static logic */  
write_gpio (GPIO_addr, buffer_num);  
Buffer2DDR_initialize(PHY1_addr, Camera_buff, buff_in, rows, cols, buffer_num);  
Buffer2DDR_initialize (PHY2_addr, buff_out, VGA_buff, rows, cols, buffer_num);  
/* Config Block A logic */  
load_bitstream("A_partial. bin", partial_bit);
```

```
/* Initialize Block A logic */  
HW_initialize(A, rows, cols, buffer_num);  
  
While(1){  
/* Start Block A logic */  
IPcore_Start_Ex(A);  
/* Config Block B logic */  
load_bitstream("B_partial. bin", partial_bit);  
/* Initialize Block B logic */  
HW_initialize(B, rows, cols, buffer_num);  
/* Stop Block A logic */  
IPcore_Stop_addr(A);  
/* Start Block B logic */  
IPcore_Start_Ex(B);  
/* Config Block A logic */  
load_bitstream("A_partial. bin", partial_bit);  
/* Initialize Block A logic */  
HW_initialize(A, rows, cols, buffer_num);  
/* Stop Block B logic */  
IPcore_Stop_addr(B);}
```

4 实验结果与分析

4.1 用户编程模式

延迟隐藏机制的编程基于 Xilinx Vivado 环境,算法设计使用 HLS 工具^[9],使用芯片是 Xilinx Zynq 系列 SOC,芯片包括 ARM 硬核和 FPGA 逻辑两个部分. 编程模式如图 7 所示. 图 7 中左侧为用户需要编写的程序,中间是 EDA 工具完成的工作,右侧是延迟隐藏机制中完成的工作. 延迟隐藏机制完成了用户 IP 核的接口管理和通信协议;提供除用户算法以外的所有 FPGA 上的逻辑设计;开放了软件开发所需的函数库. 用户基于延迟隐藏机制进行 FPGA 开发时,只需使用 C/C++ 或 SystemC 完成算法和测试文件的设计,完成 ARM 上 API 的编写. 而在不同的设计阶段,延迟隐藏机制都有相应的函数库以供使用,最大程度上简化了 FPGA 的开发,使 FPGA 在编程上具备和 GPU 一样的优势.

4.2 实验结果与分析

本文所使用的实验平台是数学工程与先进计算国家重点实验室开发的搭建了 Xilinx Zynq7030 的蚁群计算平台,其中 PS 部分使用的处理器是 ARM Cortex-A9 双核处理器,异构芯片是 xc7z030fbg484-3. 测试选取的算法是边缘提取算法 Sobel^[10]. Sobel 算子对噪声具有平滑作用,可以提供较为精确的边缘方向信息,是一种较为常用的边缘检测方法.

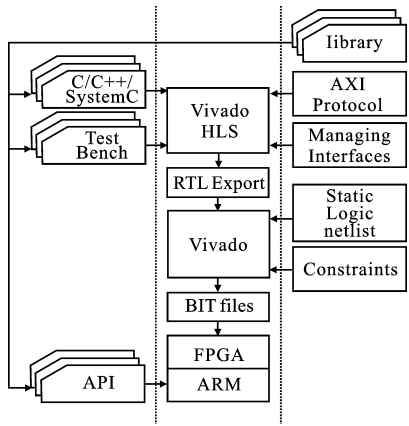


图 7 用户编程模式及开发流程

表 2 所示为 Sobel 算法使用的资源分别在 A、B 分区内所占比例以及在芯片总资源中占的比例,算法工作频率是 200 MHz,可以处理的图片最大列数为 1 920,算法资源使用须参考表 1,使用的资源不能超出 A、B 区内的资源。

表 2 Sobel 算法资源使用

	Total/%	A/%	B/%
flip-flop(FF)	1.09	9.84	9.45
LUT	2.89	6.02	5.43
BRAM	0.56	1.50	1.76
DSP	3.50	15.00	15.00

本文以 Sobel 运算为例,对传统的 PR 切换机制和延迟隐藏机制下的运算性能进行了测试,缓存帧数由单次配置 BIT 文件的时间和算法处理单帧图片的时间决定,由式(3)~(5)可得不同图片尺度下的实验参数如表 3 所示。

表 3 实验主要参数

	480 p	720 p	1 080 p
单次配置时间/s	0.1	0.1	0.1
单帧处理时间/s	0.01	0.03	0.05
缓存帧数	12	4	2
处理总帧数	24	8	4

图 8 所示是不同模式下的处理完所有帧数的总时间。传统模式的总时间为配置时间和计算时间的总和。

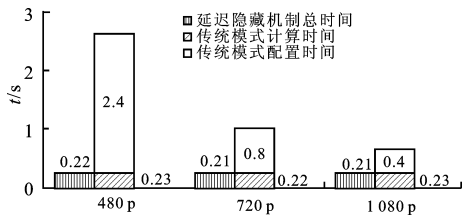


图 8 不同模式下的性能对比

传统模式中,配置的延迟占据了运行时间的绝大部分,降低了整体的运行效率。延迟隐藏模式的总

体运行效率相比传统模式有很大的提升,且运行时间接近传统模式的加算时间。由此可见,延迟隐藏模式可以有效地屏蔽 FPGA 运行中重构产生的延迟,提升整体的运行效率。

5 结束语

本文设计并实现了用于 FPGA 运行时重构的延迟隐藏机制,可以屏蔽 FPGA 动态可重构时配置 BIT 文件的时间,提升了 PR 机制的运行效率。同时,本文提出的机制很大程度上简化了 FPGA 上的编程开发,使 FPGA 在编程上具备和 GPU 一样的优势。

为了进一步提升 FPGA 编程和实现的优越性,未来工作的主要内容除了进一步优化延迟隐藏机制外,参照 OpenCL 标准对 API 进行整合和优化,实现延迟隐藏机制对 OpenCL 的支持,进一步简化 FPGA 的编程,提升 FPGA 的易用性和可移植性。

参考文献:

[1] 丁帅帅,柴志雷. 基于 HLS 的 SURF 特征提取硬件加速单元设计与实现[J]. 微电子学与计算机,2015,32(9):133-137.

[2] Martin G, Smith G. High-level synthesis: past, present, and future[J]. IEEE Design and Test of Computers,2009,26(4):18-25.

[3] Altera. Implementing FPGA design with the openCL standard [EB/OL]. [2015-11-15]. http://www.altera.com/en_US/pdfs/literature/wp/wp-01173-opencl.pdf.

[4] Conger C, George A D, Gordon Ross A. Design framework for partial run-Time FPGA reconfiguration [C]// 2008 International Conference on Engineering of Reconfigurable Systems & Algorithms. Las Vegas, Nevada,USA,2008.

[5] Nvidia What is gpu computing[EB/OL]. [2015-11-19]. <http://www.nvidia.com/object/what-is-gpu-computing.html>.

[6] Xilinx. Vivado design suite user guide: partial reconfiguration [EB/OL]. [2015-11-27]. http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_3/ug909-vivado-partial-reconfiguration.pdf.

[7] ARM, AMBA. AXI™ and ACE™ protocol specification [EB/OL]. [2015-11-19]. http://www.gstitt.ece.ufl.edu/courses/eel4720_5721/labs/refs/AXI4_specification.pdf.

- IEEE 802.11s: the WLAN mesh standard. [J]. IEEE Wireless Commun., 2010, 17(1):104-111.
- [2] 张牧,严军荣. 802.11s 无线 mesh 网络研究进展与挑战[J]. 计算机工程与应用,2010,46(22):75-79.
- [3] 杨孟珂,杨亚涛,白中英. 基于 HWMP 协议的无线 Mesh 网络多网关路由协议研究[J]. 微电子学与计算机,2009,26(12):4-8.
- [4] 姬文江,马建峰,马卓,等. 无线网格网中一种基于树的先验式路由协议(英文)[J]. 中国通信,2012,8(1):25-33.
- [5] Hu Yun, He Weiqing, Yang Shoubao, et al. Multi-gateway multi-path routing protocol for 802.11s WMN [C]// proceedings of IEEE 6th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob'10). Ontario, Canada, 2010:308-315.
- [6] 李茜,刘经纬,周雅楠,等. 基于 IEEE 802.11s 的无线 Mesh 网络路由协议研究[J]. 现代电子技术,2013,36(23):36-40.
- [7] NS-3 developers. NS-3 Tutorial 2015[EB/OL]. [2015-11-15]. <https://www.nsnam.org/documentation/>. 2015.
- [8] IEEE 802.11 INT 1-2012, IEEE Std 802.11-2012 用 IEEE 标准解释文件. 信息技术用 IEEE. 系统间通信和信息交换. 局域网和城域网. 专门要求. 第 11 部分:无线 WAN 媒体访问控制(MAC)和物理层(PHY)规范[S]. 美国,加利福尼亚,电气和电子工程师协会,2012.

作者简介:

潘琢金 男,(1962-),教授,硕士生导师. 研究方向为嵌入式系统、计算机检测与控制.

吴昊(通讯作者) 男,(1988-),硕士研究生. 研究方向为嵌入式系统. E-mail:whimsyduke@163.com.

罗振 男,(1981-),博士,讲师. 研究方向为以太网路由技术、嵌入式系统.

杨华 男,(1974-),博士,副教授. 研究方向为计算机系统结构、智能人机交互、仿人机器人、机载计算和智能控制.

(上接第 44 页)

- [8] 何宾. Xilinx FPGA 设计权威指南[M]. 北京:清华大学出版社,2014.
- [9] Xilinx. Vivado design suite user guide: high-levelSynthesis [EB/OL]. [2015-11-10]. http://www.xilinx.com/support/documentation/sw_manuals/xilinx2012_2/ug902-vivado-high-level-synthesis.pdf.
- [10] 袁春兰,熊宗龙,周雪花,等. 基于 Sobel 算子的图像边缘检测研究[J]. 激光与红外,2009,39(1):85-87.
- 作者简介:**
- 刘伟** 男,(1990-),硕士研究生. 研究方向为嵌入式系统设计与集成. E-mail:313165240@qq.com.
- 柴镇** 男,(1992-),硕士研究生. 研究方向为嵌入式软件.
- 周浩杰** 男,(1981-),博士,高级工程师. 研究方向为计算机系统结构、大数据处理.
- 吴东** 男,(1971-),博士,高级工程师. 研究方向为 FPGA 高效能计算机.
- 柴志雷** 男,(1975-),博士,副教授. 研究方向为嵌入式系统设计技术、高性能视觉系统、基于 FPGA 的可重构计算和 FPGA 操作系统等.