**大连理工大学本科毕业设计（论文）**

**基于Zynq的演化硬件验证平台设计**

**Design of Zynq-Based Evolvable Hardware Verification Platform**

学 院（系）： 软件学院

专 业： 软件工程（日语强化）

学 生 姓 名： 李婷

学 号： 201293060

指 导 教 师： 王洁

评 阅 教 师： 侯刚

完 成 日 期： 2016. 05. 30

大连理工大学

Dalian University of Technology

# 摘 要

近年来，随着嵌入式系统复杂程度急剧上升，如何设计和管理这些复杂系统，特别是当出现故障时如何对系统进行维护成了亟须解决的问题。对于维护代价高昂的系统，比如深海探测器，人造卫星或太空飞船等，一旦出现故障维护的代价非常大，故障损坏系统到一定程度会使之变得毫无用处。为了解决这种问题，可以自主地，动态地改变自身结构和行为的演化硬件技术的概念被提出。演化硬件等于演化算法加可编程逻辑器件，随着现场可编程门阵列技术的发展，演化硬件系统的实现成为了可能。

本文基于Xilinx公司的Zynq系列开发板，充分利用Zynq系列以处理系统为中心，可编程逻辑和处理系统协同的架构，按照软硬件协同设计的方法，设计了演化硬件系统的验证平台。此演化平台的架构是在可编程逻辑部分实现虚拟可重构电路和适应度计算的加速，在处理系统部分实现遗传算法，完成群体初始化，选择，交叉变异等操作，并且使可编程逻辑部分和处理系统部分通过AXI从总线进行通信。本文基于此演化平台选取了3/8位奇偶校验器，2位乘法器作为实验案例进行验证。

本文设计的演化硬件平台，有效地结合软件设计的灵活性，可扩展性和硬件设计的实时性。基于本文演化硬件平台设计案例的测试结果显示3位奇偶校验器在1000代内能演化出最优解的概率是99%，8位奇偶校验器在5000代内能演化出最优解的概率是92%，2位乘法器在10000代内能演化出最优解的概率是60%。

关键词**：**Zynq；演化硬件；可编程逻辑器件；遗传算法；软硬件协同设计

**Design of Zynq-Based Evolvable Hardware Verification Platform**

# Abstract

In recent years, with the rapid rise in the complexity of the embedded system, how to design and manage these complex system, especially how to maintain the system when a fault occurs becomes an urgent problem to solve. For maintenance-costly system, such as deep-sea explorer, artificial satellite, space ship et. Once the malfunction occurs, the cost of maintenance is very large. The system would become useless when the fault damage the system to a certain extent.

To solve this problem, the concept of evolvable hardware which can customize and dynamically change their structure and behavior have been proposed. Evolvable hardware equals evolutionary Algorithm plus programmable logic device. With the development of FPGAs, it is possible to implement the evolvable hardware system.

This paper based on Xilinx’s Zynq development board, take full advantage of the architecture that the processing system is the center and programmable logic device collaborative work with processing system. This paper used the hardware and software co-design method, implement the verification platform of evolvable hardware system. In this paper, we implement the virtual reconfigurable circuits and accelerate the fitness calculation in the programmable logic part, the implement of the generic algorithm, the population initialization, crossover and mutation operation in the processing system part. The programmable logic part communicate with processing system part by AXI master bus. This paper also designed the 3/8 bits parity-checker, 2bits multiplier as experimental case based on this evolution platform.

The evolvable hardware system designed in this paper, effectively combined the flexibility and extensibility of the software design with real-time of the hardware design. We also summary the run results of the 3/8 bits parity-checker and 2bits multiplier. The results of experimental case based on this evolution platform show that the possibility of 3 bits parity-checker can evolvable optimal solution in 1000 generations is 99%, the possibility of 8 bits parity-checker can evolvable optimal solution in 5000 generations is 92%, the possibility of 2 bits multiplier can evolvable optimal solution in 10000 generations is 60%.

**Key Words：**Zynq；Evolvable Hardware；Programmable Logic Device；Generic Algorithm；Software and Hardware Co-design

目 录

[摘 要 I](#_Toc453441599)

[Abstract II](#_Toc453441600)

[引 言 1](#_Toc453441601)

[1 课题概述 2](#_Toc453441602)

[1.1 国外研究现状 2](#_Toc453441603)

[1.2 国内研究现状 2](#_Toc453441604)

[1.3 本文组织结构 3](#_Toc453441605)

[2 系统开发平台 4](#_Toc453441606)

[2.1 软件开发工具及环境 4](#_Toc453441607)

[2.1.1 开发环境介绍 4](#_Toc453441608)

[2.1.2 Xilinx设计工具介绍 5](#_Toc453441609)

[2.2 硬件平台介绍 7](#_Toc453441610)

[2.2.1 Zynq-7000系列介绍 7](#_Toc453441611)

[2.2.2 Zedboard开发板介绍 8](#_Toc453441612)

[3 演化硬件技术分析 11](#_Toc453441613)

[3.1 演化硬件概述 11](#_Toc453441614)

[3.1.1 演化硬件的原理 11](#_Toc453441615)

[3.1.2 演化硬件的特点 11](#_Toc453441616)

[3.1.3 演化硬件的分类 12](#_Toc453441617)

[3.2 虚拟可重构电路 12](#_Toc453441618)

[3.3 笛卡尔遗传编程 14](#_Toc453441619)

[3.4 遗传算法 16](#_Toc453441620)

[3.4.1 遗传算法概念及流程 16](#_Toc453441621)

[3.4.2 选择策略 17](#_Toc453441622)

[3.4.3 交叉策略 18](#_Toc453441623)

[3.4.4 变异策略 20](#_Toc453441624)

[4 基于Zynq的演化硬件平台架构设计 21](#_Toc453441625)

[4.1 可编程逻辑部分设计 22](#_Toc453441626)

[4.1.1 虚拟可重构电路设计 22](#_Toc453441627)

[4.1.2 适应度计算模块设计 24](#_Toc453441628)

[4.2 处理系统部分设计 26](#_Toc453441629)

[4.2.1 选择部分设计 27](#_Toc453441630)

[4.2.2 交叉部分设计 28](#_Toc453441631)

[4.2.3 变异部分设计 29](#_Toc453441632)

[4.3 通信部分设计 29](#_Toc453441633)

[4.3.1 用户逻辑部分建立和配置 30](#_Toc453441634)

[4.3.2 IP核添加和设置 33](#_Toc453441635)

[4.3.3 处理系统部分调用 34](#_Toc453441636)

[5 基于演化平台的实验案例验证 35](#_Toc453441637)

[5.1 硬件部分配置 35](#_Toc453441638)

[5.2 软件部分配置 35](#_Toc453441639)

[5.3 实验结果分析 36](#_Toc453441640)

[5.3.1 硬件仿真结果 36](#_Toc453441641)

[5.3.2 下载至开发板运行结果 38](#_Toc453441642)

[5.3.3 实验结果总结 38](#_Toc453441643)

[结 论 42](#_Toc453441644)

[参 考 文 献 43](#_Toc453441645)

[致 谢 45](#_Toc453441646)

# 引 言

早在上个世纪五十年代，计算机之父约翰尼·冯·诺依曼就提出了研制具有自繁殖和自修复能力机器的设想，但由于当时技术条件的限制使这一设想未能实现[1]。直到演化计算和大规模可编程芯片的出现，这种设想才成为可能。

随着FPGA（现场可编程门阵列）技术的成熟，演化硬件越来越受到复杂电路设计者以及自适应系统研究者们的青睐。传统硬件设计一旦实现后结构和功能是固定的，演化硬件实际上是一种特殊的硬件，它可以像生物一样具有自适应，自组织，自修复特性，从而可以根据环境的改变而改变生存结构以适应环境变化[2]，演化硬件等于演化计算加上可编程逻辑器件。1996年，Adrian ThompSom首次在XC6216 FPGA上进行了演化电路设计的实验，这极大地鼓励了研究者在演化硬件领域的工作[3]。在上个世纪九十年代末，由于早期的Virtex系列芯片对部分重构的支持不足，研究者引入了用于演化设计的虚拟可重构电路。当前的Virtex系列FPGA提供了对动态重构的合理支持。在2011年，Xilinx公司引入了一个新的被称为Zynq-700 All Programmable Soc的可重构片上系统。它整合了可编程逻辑（Programmable Logic，简称PL）和处理系统（Processing System，简称PS），这种以PS为核心的结构具有更大的灵活性，可以使片上ARM更快速简单地控制重构逻辑。

本文的目的是充分利用Zynq-7000系统的特点，在开发板的可编程逻辑部分实现虚拟可重构电路的构造，和适应度计算部分的加速，在处理系统部分实现演化算法的群体初始化部分，选择，交叉变异部分。其中的难点部分是PL和PS部分的通信问题和染色体的存储表示转换问题，以及选择策略等。本文先介绍了演化硬件的定义和特点，Zynq-7000系统的特点，笛卡尔遗传编程的概念和虚拟可重构电路的定义，这是PL部分要实现的内容。由于Zynq-7000提供了ARM处理器，遗传算法的群体初始化，选择交叉变异的操作可以在处理器部分实现，不需要用FPGA部分电路的实现，大大提高了灵活性和可扩展性。比如选择的轮盘赌策略，如果没有ARM处理器，直接在FPGA上实现，需要大量的乘法器，除法器和加法器，进行迭代操作，非常复杂。在ARM处理器上可以直接在SDK上用C语言实现，变得很简单。再比如变异位置的选择需要产生伪随机数，如果用可编程逻辑则需要用线性负反馈实现伪随机数发生器，很复杂而且可扩展性低，相反在ARM部分实现只需要在SDK上用C语言实现伪随机数的产生。

# 1 课题概述

随着电子系统复杂程度的提高，整个设计由人来完全承担已经不可能，演化硬件一方面可以大大减轻设计人员的负担，另一方面可以满足了环境对硬件要求的适应性[4]。演化硬件虽然是一个新兴领域，然而已经在许多领域得到了应用，主要的领域有电路设计，控制和机器人，模式识别，容错系统，超大规模集成电路（Very Large Scale Integrated Circuits，简称VLSI）的设计。2010年，Xilinx公司推出行业第一个可扩展处理平台，它是将双核的ARM-Cortex A9处理器作为主系统，可编程部分是基于Xilinx 28nm 7系列的FPGA，Zynq系列的这种结构为软硬件协同设计提供了可能。

## 1.1 国外研究现状

21世纪以来，以美国Xilinx公司为代表的商业芯片公司在不断地开发适合演化硬件的可编程逻辑器件。利用演化技术，北美，欧洲和日本的一些研究机构已经设计实现了包括数字电路，模拟电路，数字信号处理系统在内的多种功能电路[5]。下面介绍几种国外演化硬件的典型研究：

（1）英国人Delon Levi在制造高性能演化芯片上取得了突出的成绩，并且研发出支持芯片演化的支撑软件[6]。

（2）英国Sussex大学的Adrian Thompson等人在内部演化技术上获得了突破性进展[7]，之后他们又对极小规模电子电路的演化展开了研究，这种极小规模电子电路甚至只包含一个单一的晶体管，这种针对电路物理特性的研究对研究演化硬件的内部机理有着深远的影响[8]。

（3）美国NASA Ames研究中心在过去的几年里，把演化算法成功地应用到多项NASA应用中，包括航天器的天线设计，可编程逻辑芯片的容错，模拟电路设计等[9]。

## 1.2 国内研究现状

近些年，国内研究人员在演化硬件的发展方面也贡献了力量，下面是演化硬件领域国内团队的典型研究。

（1）南京航空航天自动化学院的王友仁教授带领的演化硬件研究小组，在数字电路的函数级在线自适应演化系统[10]，演化硬件软件平台的可行性[11]，N模异构冗余容错系统[12]方面取得了显著地成绩，他们的工作为进一步开展研究积累了坚实的理论基础和宝贵的实践经验。

（2）武汉大学软件工程国家重点实验室的康立山教授，何巍教授和陈毓屏教授带

领的团队，从演化硬件的结构出发，提出了函数型（树型结构函数）的演化硬件结构，

提高了编码效率，加快了演化速度[13]。

（3）西安电子科技大学电子工程学院的赵曙光教授研究了基于最小项表达式的染色体编码新方法，并且证明了在设计输入端个数较少的电路时效果较好[14]。

（4）军械工程学院的王国庆等人应用演化硬件技术对三模冗余技术进行了研究[15]，该研究中有3个相同模块实现相同的功能，利用少数服从多数的原则，若一个模块出现故障，另外两个模块仍然正常运行，则可以直接替换出错的模块，若两个以上模块出现故障则用演化算法演化出最优电路。

## 1.3 本文组织结构

本文以现有的软硬件平台为基础，利用软硬件协同设计的方法，基于ZedBoard开发板，在可编程逻辑部分实现了虚拟可重构电路，适应度计算的加速，在处理器端实现遗传算法，两者交互演化最优解。接下来的章节将介绍本文的技术原理，基于Zynq的演化硬件平台架构设计，架构上运行的实验案例及结果。

第二章详细介绍了系统平台，包括软件开发工具，开发环境以及硬件平台架构和板载资源。

第三章详细介绍了演化硬件技术，包括演化硬件的原理，特点和分类，里面用到的关键技术和算法，比如虚拟可重构电路的思想，笛卡尔遗传编程的染色体编码方法，以及经典的遗传算法策略等。

第四章详细介绍了本文平台架构的设计，主要包括可编程逻辑部分设计，处理系统部分设计和通信部分设计。

第五章详细介绍了基于此平台架构设计的实验验证案例，主要包括软硬件部分的配合，以及最后的测试结果。

第六章对本文所做的工作进行总结，并提出了下一步需要改善的地方。

# 2 系统开发平台

## 2.1 软件开发工具及环境

### 2.1.1 开发环境介绍

本项目使用的交叉开发平台如图2.1所示。



图2.1 交叉开发平台示意图

图2.1是交叉开发平台示意图，从图中可以看出，开发板的UART串口（J14）和PROG下载口（J17）通过HUB连接到了电脑上。并且启动模式设置成JTAG启动。J17下载口是往FPGA部分下载bit流文件的接口，J14是USB转UART口，采用了Cypress公司的USB-UART控制器。CY7C64225是一种简洁高效的USB串口解决方案，它在片内集成了UART收发器、USB2.0全速控制器、晶振以及EEPROM等各种功能，而只采用了28脚的SSOP封装。在使用UART串口的时候需要去Xilinx的官网下载驱动程序进行安装，否则电脑无法识别UART口，并且必须设置波特率是115200，如果设置其它的波特率，可能会导致控制台的输出都是乱码的情况，其它配置按照默认即可。

演化硬件平台的设计开发流程主要包括以下步骤：在软件部分实现遗传算法，先编写C代码，用arm gcc编译器进行编译生成可执行文件；在硬件部分实现虚拟可重构电

路和适应度计算的设计，先编写Verilog HDL代码，然后进行综合实现仿真。最后在软硬件交互部分进行配置，实现软硬件部分的交互。设计流程图如图2.2所示。

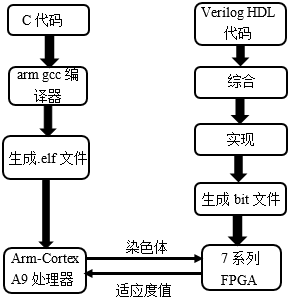


图2.2 系统设计流程示意图

从图2.2中可以看出，本项目的开发环境主要由两部分组成，其中一部分是ARM-Cortex A9处理器，另一部分是7系列的FPGA。ARM-Cortex A9处理器上主要实现遗传算法，每次将染色体传给7系列的FPGA进行下载配置，7系列的FPGA将计算出的适应度传给处理器部分进行选择交叉变异等操作。

### 2.1.2 Xilinx设计工具介绍

本文使用到的Xilinx设计工具的版本是14.7，ISE 14.7是赛灵思公司于2013年10月发布的开发工具，是Xilinx设计最新的工具，并且从当年10月起，ISE已移植到其它产品的开发中，不再有计划地发布新的设计套件，其它产品的发布主要是vivado开发工具和SDx工具等。ISE开发套件主要由PlanAhead工具，ISE工具，EDK（XPS，XSDK）工具，Chipscope工具等组成。

本文使用到的设计工具及完成的工作如下：

（1）PlanAhead工具是本文实现演化硬件的可编程部分开发工具，主要是导入硬件源文件，进行约束文件设计和综合仿真生成bit流，导出比特流到SDK的工具。

（2）ISE开发工具主要是IP核的虚拟可重构电路和计算适应度部分的源码设计，

仿真验证。

（3）Xilinx Platform Studio工具主要是MIO块的选择和配置，IP核的添加和配置，以及IP核地址的生成。

（4）Xilinx Software Development Kit工具的主要作用是编写软件部分的实现，主要是遗传算法的群体初始化，选择策略，交叉变异策略，染色体编码转换的工具，并且最后通过SDK工具下载PlanAhead生成的比特流文件，运行ARM部分的文件，并通过开发板的串口将运行结果传送到SDK的控制台。

PlanAhead是在设计FPGA设备的时候对软件产品设计和分析的。它为整个FPGA的实现过程提供了一个集成的和直观的开发环境。通过PlanAhead，你可以通过分析设计的RTL级的资源，综合的网表和实现的结果来提高电路的性能。Platform Studio是整合了从设计到调试认证所有过程的图形接口。用户可以用Platform Studio去设计嵌入式系统任务。

ISE的主要特点如下：

（1） 完成FPGA/CPLD开发的集成开发环境；

（2） 开发界面简单易学；

（3） 提供在线帮助信息，可以方便解决遇到的各种问题；

（4） 提供很多辅助设计工具。

Xilinx

PlanAhead

ISE Design Suite

Xilinx Platform Studio

Xilinx

SDK

修改IP核（主要是增加VRC部分和适应度计算部分）

在xmp中添加外设

新建并添加IP核

给IP核分配地址

新建xmp源文件

添加ucf文件

生成.bit文件

实现遗传算法

生成.elf文件

导出.bit文件到SDK

修改后的IP核

空的XMP

源文件

添加了外设的XMP

源文件

图2.3 开发工具实现功能图

图2.3是开发工具实现功能图，其中主要用到四个开发工具，Xilinx PlanAhead主要用于建立嵌入式系统源文件即xmp文件，空的xmp文件需要添加外设和功能项，就需要用Xilinx Platform Studio添加外设和用户自定义IP核，用户自定义IP核的功能需要

在ISE Design Suite里进行完善添加。ISE主要就是用户自定义IP核功能的实现，实现

的主要是虚拟可重构电路和适应度计算的功能。当IP核功能完善之后，在XPS里添加，然后生成IP核的地址，xmp文件完善完成，再添加用户约束文件，最后生成比特流文件，并将比特流文件导入到Xilinx SDK里，Xilinx SDK里用C语言实现遗传算法，并生成elf文件，可以与FPGA部分交互运行。

## 2.2 硬件平台介绍

### 2.2.1 Zynq-7000系列介绍

Zynq-7000系列开发板主要有Zedboard和Zybo。它们使用的都是Zynq AP SOC架构。Zynq-7000平台结构如图2.4所示。

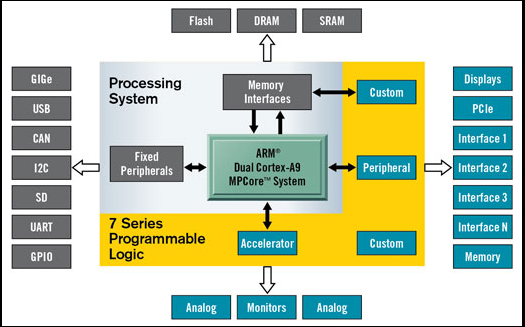


图2.4 Zynq-7000平台结构图

图2.4是Zynq-7000系列平台的结构图，从图中可以看出PS部分的核心是双核ARM Cortex-A9处理系统，同时还有固定的外设，内存接口与处理器进行交互。内存接口与DRAM，SRAM和Flash进行交互。固定外设和CAN，I2C，GPIO等进行交互。7系列的可编程逻辑（PL）部分主要有加速器，外设和用户自定义部分。加速器主要是由模拟器和监控器等组成。外设部分主要是和接口，PCIe等交互。同时，PS的内存接口也可以和PL部分的用户自定义部分交互。从结构图中可以看出，Zynq-7000系列和之前赛灵思公司的其它系统FPGA不同，本系列是以ARM处理器为核心，可编程逻辑类似于

围绕在ARM处理器周边的外设。用户可以在PL部分自定义外设，然后和处理器部分交互实现软硬件协同设计。此种架构的FPGA，使得软硬件协同设计成为可能，设计更加方便，可扩展性高，并且引入了ARM处理器，使处理更加高效。

### 2.2.2 Zedboard开发板介绍

本项目使用的开发板是Xilinx 大学项目的Zedboard开发板，它是基于Zynq-7000扩展处理平台的一款评估开发板，包含了双核ARM-Cortex A9处理器和可编程逻辑FPGA，是一款以处理系统为核心的开发板。它具有丰富的外围设备和强大的扩展能力，不论对于初学者还是经验丰富的设计者，它都是理想的开发平台。

本文选用的开发板是ZedBoard不是Zybo，它们都属于Zynq-7000系列，Zybo是一款超低价格的Zedboard替代品，它们都兼顾了Zynq AP SOC架构，但是由于Zybo相对于ZedBoard外设接口少，本文需要将适应度通过LED显示，Zybo的LED何SWITCH只有4个，如果利用Zybo则对于8位奇偶校验器，则不能通过LED显示其适应度，对于2位乘法器，最大适应度是16，Zybo也表示不了，所以本文使用ZedBoard它更直观。图2.5是开发板实物图。

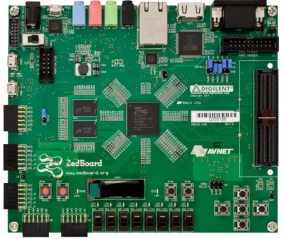


图2.5 Zedboard实物图

图2.6是Zedboard开发板片上资源示意图，从图中可以看出处理系统（PS）部分的特点如下：

（1）基于双核ARM-Cortex A9的应用处理单元：每个CPU 2.5DMIPS/MHZ，超过1GHZ；具有双精度和单精度的向量浮点单元（Vector Floating Point Unit，简称VFPU）；具有三个看门狗定时器，一个全局时钟。

（2）缓存：每个CPU有4路组相连的一级缓存32KB，CPU之间共享的8路组相连二级缓存512KB；支持字节奇偶校验。

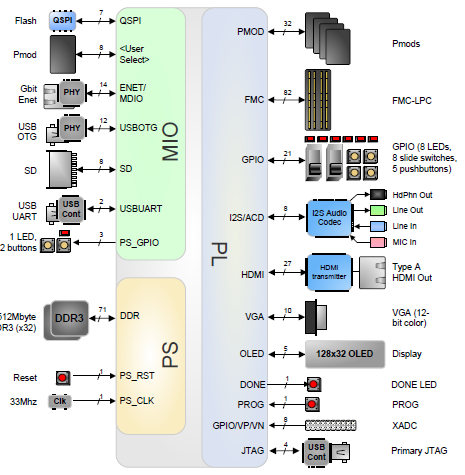


图2.6 Zedboard开发板片上资源示意图

（3）存储器：包含512KB DDR3和256Mb QSPI Flash。

（4）外围接口：包括两个10/100/1G的以太网口；两个USB2.0接口；两个CAN2.0总线接口；SD卡接口；2个告诉UARTs接口；2个主从I2C总线等。

（5）互连：包含基于AXI的ARM AMBA的高速带宽，用于可编程逻辑和处理系统之间的通信。

可编程逻辑（PL）部分的特点如下：

（1）可配置逻辑块：包括查找表，触发器和级联加法器。

（2）36Kb的块状RAM：包括真正的双端口RAM并且可以被配置成2个18Kb的块状RAM。

（3）可编程IO块：支持通用的IO标准，包括LVCMOS，LVDS和SSTL。

Zynq系列拥有处理器配置访问接口（Processor Configuring Access Interface，简称PCAP），使得它可以不需要PL内的任何实例。它以PS为中心的架构非常利于演化硬件设计，首先，演化系统可以通过ADC来得到环境的返回，从而改善自己来适应环境的变化；然后，它可以不用PL的支持来在本地执行应用，也可以通过PL来加速应用的演化。应用程序既可以运行在操作系统上，也可以运行在裸机上[3]。

# 3 演化硬件技术分析

## 3.1 演化硬件概述

### 3.1.1 演化硬件的原理

演化硬件（Evolvable Hardware，简称EHW）是指通过与环境的相互作用自适应改变其硬件结构和行为的硬件。狭义概念上，演化硬件是通过演化算法实现电子系统自身的重构；广义概念上，演化硬件包括各种形式的硬件，从传感器到能够适应自身的环境，并且可在运行期间增强性能的整个演化系统[5]。本文所讨论的演化硬件指狭义概念上的演化硬件，就是将演化算法和可编程逻辑器件相结合产生的新的硬件。演化硬件一般可以采取如下的公式定义：EHW=EAs + PLDs。演化硬件的基本工作原理如图3.1所示。

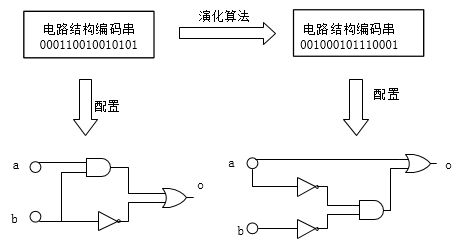


图3.1 演化硬件的基本工作原理

演化硬件的基本工作原理如图3.1所示，图中一条电路结构编码串表示一个电路结构图，演化前的电路结构编码是000110010010101，下载到可编程逻辑器件部分就会生成功能如C = a & b | ~b的电路结构图。经过演化算法的处理，电路结构的编码串就会变成如右图所示的001000101110001，这条编码串下载到可编程器件部分会生成如C = ~a & ~b | a的电路图。

### 3.1.2 演化硬件的特点

通过研究演化硬件的特点，我们可以更精确地认识到什么是演化硬件，什么不是演化硬件[16]。演化硬件通常包括发展的循环，循环的终止条件可能是达到最大适应度或者

演化了指定大小的代数，选择策略和变异操作等。以下是演化硬件系统设计必须要考虑的几个特点：

（1）演化过程的最终结果必须是硬件的或者是至少能在硬件上实现的。

（2）演化出来的系统表现出一定程度的容错性，这在传统方法设计的系统中是不可能的。

（3）它们被设计实现是为了适应变化的环境，可能不是最优的结果，但是可以满足需求，这种适应度取决于什么时候循环停止。并不是所有的系统都必须绝对的正确[16]。

（4）演化设计最终的结果必须得解决一个有用的问题或者说明一个方法的可用性。比如演化一个n位的多路选择器就没有什么意义[16]。

### 3.1.3 演化硬件的分类

演化硬件根据目标电路，染色体编码，演化过程等大概可以分为以下几类：

（1）根据演化目标电路的不同可以分为演化数字电路，演化模拟电路和演化混合电路[17]。本文演化的例子是组合电路。

（2）根据染色体编码方式的不同可以分为直接编码方式和间接编码方式。直接编码方式是直接将FPGA的配置位串作为染色体进行编码，当FPGA的配置位串特别长的时候，演化所需要的时间和存储规模也增大，直接限制了演化的速度和电路规模。间接编码是将树或者特殊的语法规则作为染色体，在评估电路配置的时候需要先进行翻译解码生成配置位串，然后再对配置位串进行评估。本文使用的是笛卡尔遗传编程的方式，属于间接编码。

（3）根据演化操作基本单元的不同可以分为门级演化硬件和函数级演化硬件。函数级演化硬件演化的基本电路不是门而是函数，有利于演化更复杂的功能电路，又不增加演化困难。由于本文演化的就是基本的奇偶校验器和乘法器比较简单，所以就是简单的门级演化。

（4）根据适应度评估方式的不同分为外部演化和内部演化。外部演化是在软件部分进行适应度评估，每一代只把最优的染色体下载到硬件部分，也就是硬件部分只配置一次。内部演化是在硬件部分计算适应度，每一条染色体都会被下载到硬件部分进行配置。本文采用的是内部演化的方式。

## 3.2 虚拟可重构电路

虚拟可重构电路（Virtual Reconfigurable Circuit，简称VRC）是在FPGA之上的第二重构层，目的是为了减少配置比特流的长度和获得更快的内部重构[18]。虚拟可重构电路是由功能元素（Function element，简称FE）矩阵组成，每个FE的功能和FE之间的连接项取决于染色体的配置比特流，而染色体是由演化算法生成的[19]。虚拟可重构电路的每个功能元素是具有一定逻辑功能的电路，逻辑功能既可以是简单的与或非门，也可以是复杂的函数，根据目标系统的复杂度确定。每个FE由两个连接项和一个功能项组成，连接项表示的是FE的输入是来自哪个编号的FE的输出，功能项表示的是对两个连接项进行何种运算。8位奇偶校验器的虚拟可重构电路示意图如图3.2所示。

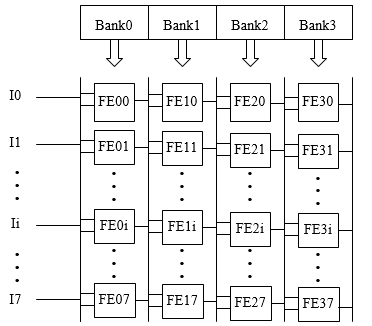


图3.2 8位奇偶校验器的虚拟可重构电路示意图

图3.2是8位奇偶校验器的虚拟可重构电路示意图，它是8\*4的FE矩阵，其中FE00，FE01，…FE0i，…FE07是第一列FE，它们存储Bank0里，依次类推，FE30，FE31，…FE3i，…FE37是最后一列FE，它们存储在Bank1里。每一个FE用两个输入项和一个功能项进行编码，其中每一列FE的输入来自前λ列FE的输出，本文中λ取值为1，也就是每一列FE的输入来自前1列FE的输出。第一列FE的输入来自整个电路系统的输入，对于8位奇偶校验器来说就是8位输入，整个系统的输出来自最后一列FE的输出。

整个虚拟可重构电路是用一条染色体来表示的，整个电路最终实现什么样的功能，取决于染色体的值，也就是电路是可以用染色体进行重构的。整个染色体的编码利用笛卡尔遗传编程的思想。如图3.2的8位奇偶校验器的虚拟可重构电路，由于输入都是从I0，I1，…I7或者前一列FE的输出中选择，所以用3位二进制串来表示一个FE的输入。对于FE的功能项来说，要根据具体演化目标的复杂程度来确定功能项种类的个数。如果对于本文中的例子8位奇偶校验器，那么功能项用3位表示就行，表示8种功能，具体的每种功能在本例中用与，或，非，异或等表示就可以演化出本文例子的功能。那么一个FE有2个输入项，一个功能项，总共需要2\*3+3=9位二进制串表示就行。对于一个8\*4的FE矩阵，FE矩阵需要8\*4\*9=288位表示，而对于8输入1输出的8位奇偶校验器来说，输出从最后一列FE的输出中选择，用3位表示就行，所以总共需要的染色体的长度是8\*4\*9+3=292位。

虚拟可重构电路中FE矩阵的大小依据具体的经验值选择，没有固定的要求。比如8位的奇偶校验器是有8个输入，如果选择3\*4的FE矩阵，在λ值为1的情况下，无论演化多少代都不能演化出最优解。而如果选择4\*4的FE矩阵，则第一列FE的输入完全覆盖输入的概率是8!/88，大约等于0.0024，如果不考虑功能模块和后几列的随机性，就只是第一列完全覆盖8个输入概率就很小。所以我们把8位奇偶校验器的FE阵列选为8\*4，经编码测试，也证明了8\*4的FE矩阵更容易演化出最优解。如果矩阵太大，会使染色体串的位数变长，很难演化出最优解，矩阵太小，染色体短，逻辑电路少，可能会出现局部最优的情况。对于FE模块的功能项，太少太简单不能演化出最优解，太大会导致染色体变长，可选的范围变大也不容易演化出最优解。比如对于演化三位奇偶校验器，如果只是使用简单的与，或，非门，3位奇偶校验器的逻辑表达式O=(~a) & (~b) & c | (~a) & b & (~c) | a & (~b) & (~c) | a & b & c为则至少需要8个，而如果使用异或门，逻辑表达式为O = a ^ b ^ c则2个异或门就可以，所以说FE的功能项要根据具体实现目标电路的复杂程度决定。



## 3.3 笛卡尔遗传编程

笛卡尔遗传编程（Cartesian Genetic Programming，简称CGP）是一种越来越被人们广泛使用的遗传编程的方法。1999年，在基于数字电路演化方面丰富的工作经验的基础上，Julian Miller首次在GECCO上提出了笛卡尔遗传编程的概念，这种想法是受到数字电路芯片FPGA架构的激励而产生的。

笛卡尔遗传编程思想是把连接项的数据表示数据来源的地址，功能项的数据是基于一个查找表来确定的，这种表示方法可以不用考虑输入输出的值，只需要表示基因的输入来自于哪一个模块，功能项表示什么功能即可，在实现数字电路的演化方面相当的简单，灵活，可扩展性强，可以解决很多问题。如图3.3是简单的笛卡尔遗传编程的例子。

图3.3是简单笛卡尔遗传编程的示意图，图中的电路是3输入，1输出，所表示的染色体的编码是010 121 341 422 560 643 8，其中下划线表示的是功能项。从输入到基因矩阵依次编码为0，1，2，…，8，其中第一个基因表示的含义是输入来自编号0的模块和编号为1的模块，也就是系统的两个输入，本基因的编号为3，编号为5的基因可以

从本基因处获得本基因的输出作为输入项，编号为3的基因表示的功能是将来自编号0模块的数据和编号1模块的数据相与。依次类推，编号为8的模块表示的是将编号为6模块的输出和编号为4模块的输出作为输入数据相与非。系统的输出来自模块7的输出。功能表表示基因模块的功能，本例子中能实现4个功能，分别是与，或，异或，与非的功能。

0

1

0

1

2

1

4

0

0

1

1

2

4

3

2

5

6

6

4

3

3

4

5

6

7

8

7

功能表

0：a&b

1：a|b

2：a^ b

3：~(a&b)

2

010

121

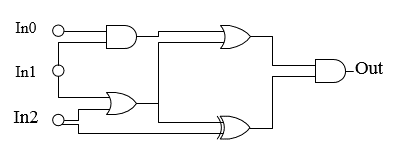
341

422

560

643

7



解码

CGP编码矩阵

解码后电路

染色体串

图3.3 笛卡尔遗传编程原理示意图

最上部分是染色体编码串，表示的是矩阵和输出项，如果有2\*3的矩阵，并且有一个系统输出，则染色体的长度是6\*3+1=19。染色体表示的电路经过CGP解码后的电路结构如图3.3所示。

本例中一个基因的输入项可以来自前任何列基因的输出项，但这样电路复杂，通常使用的笛卡尔遗传编程都是本列的基因只可以来自上一列的基因的输出，并且本列基因的输入不能来自本列基因的输出。上述都是简单笛卡尔编程的概念，随着演化硬件染色体编码技术的进步，自适应笛卡尔遗传编程技术，改变基因的输入输出个数等各项也相

继被提出，其中以自适应笛卡尔编程技术最为典型，它是简单笛卡尔编程的扩展，它的特点是包含自适应功能，基因型随着时间改变，并且可以关闭掉本身的自适应功能。

## 3.4 遗传算法

### 3.4.1 遗传算法概念及流程

遗传算法（Genetic Algorithm，简称GA）是受到达尔文生物进化思想的启发，模拟生物进化过程提出的一种启发式搜索算法。遗传算法的过程是先进行群体的初始化，然后进行选择，交叉变异，适应度计算等操作，直到得到最优解或者进化了指定的代数后停止。遗传算法分为简单遗传算法和量子遗传算法等，已经有很多研究者为遗传算法的改进做出了贡献，本文是在简单遗传算法上进行改进的，简单遗传算法的流程图如图3.4所示。

开始

群体初始化

选择

交叉

变异

适应度计算

适应度达

到最大

进化了指

定代数

否

是

结束

否

是

图3.4 简单遗传算法的数据流程图

图3.4是简单遗传算法的流程图，从图中可以看出先进行群体的初始化，初始化群

体的个数没有明确的规定，根据具体情况具体分析。如果初始群体的个数过少，会使得群体多样性变小，进化代数明显增加。整个流程终止的条件是适应度达到最大或者进化了指定的代数，适应度达到最大就是得到了最优解，可能因为策略问题或者编码问题使得得不到最优解，最后陷入局部最优可以设置终止条件是进化了最够多代。

### 3.4.2 选择策略

选择策略有轮盘赌策略，锦标赛策略，精英保留策略等。轮盘赌算法的主要思想是适应度越大的个体被选择的概率越大，假设初始群体是*P0*，*P1*，…，*Pi*，…，*Pn*，将群体中每个个体的适应度依次累加，求出适应度和*Sum*，再对每个个体*Pi*，从标号为0的个体，到标号为*i*的个体位置，依次将适应度累加得到累加和*PartSi*，依次类推，第*i-1*个个体的累积和是*PartSi-1*，随机生成一个0到*Sum*的随机数，如果随机数的范围是在*PartSi*到*PartSi-1*之间，那么选中第*i*条染色体。轮盘赌策略的一般步骤如下：

（1）群体（*P0*，*P1*，…，*Pi*，…，*Pn*）中每个个体的适应度值相累加，即整体累加和如公式（3.1）所示。

** （3.1）

（2）求从第0个个体到第*i*个个体适应度值的累计值，即部分累加和如公式（3.2）所示。

**  （3.2）

（3）随机生成一个整数*RandNum*，范围在0到*Sum*之间，依次搜索，如果*RandNum*的范围在*PartSi-1*和*PartSi*之间，则把第*i*条染色体*Pi*放到下一代的群体里。

从轮盘赌策略的步骤上可以看出，轮盘赌策略的时间复杂度是O(*n2*)，当群体的个数特别大的时候，运行时间呈指数增长，所以轮盘赌算法的时间开销比较大，但是性能稳定，不容易陷入局部最优值。

锦标赛策略的一般步骤如下：

（1）设定每次选择的个体的数量*T*，*T*的个数不宜太大，不宜超过群体个数的一半。

（2）从种群（*P0*，*P1*，…，*Pi*，…，*Pn*）中随机选择*T*个个体，每个个体被选择的概率是相同的。

（3）计算*T*个个体的适应度，找出其中适应度最大的个体*Pi*放到下一代群体里。

从锦标赛策略的步骤上可以看出，锦标赛策略的时间复杂度是O（*n*），在时间开销上比轮盘赌算法更优，但是明显锦标赛策略的稳定性不如轮盘赌算法。首先个体数量*T*的选择，如果*T*太大会导致整个个体的最优值总会在这*T*个个体中，每次选择的染色体都是一样的。但是锦标赛策略的实现更为简单，不需要累加和迭代，并且适应度最大的个体被选到下一代里的概率明显大于轮盘赌算法。

精英保留策略的思想非常简单，就是最优个体不参与交叉，变异和替换，除去最优

个体在剩下的个体中随机选择个体作为进行交叉变异操作的父体。此种选择策略虽然全局搜索能力不强，容易陷入局部最优[20]，但是最优的个体总是能保存下来，并且最优的个体不会被劣质个体替代，在某些领域，还是很有用处的，本文选择的就是精英保留策略和轮盘赌策略想结合的方式。

锦标赛策略和轮盘赌策略各有各的优势，也各有各的不足，不同的应用和不同的平台适用不同的策略。比如用可编程逻辑（PL）部分实现遗传算法，那么用Verilog HDL语言，电路级别的表示很难用轮盘赌算法，因为对每个个体都需要累加，还需要产生一个随机数，随机数的范围在0到适应度累加和之间，而累加和随着群体适应度的改变也随之改变，纯粹用电路来实现随机数的生成，目前比较成熟的是基于线性反馈移位寄存器的方法，要想实现一个从0到适应度累加和的随机数几乎不可能。所以纯硬件的遗传算法不能用轮盘赌策略，而锦标赛策略只需要生成0到群体个数之间的随机数，群体个数是固定的，所以随机数的生成是可行的，并且锦标赛策略只需要求出最大值即可不需要累加，所以锦标赛策略在纯可编程逻辑上是可行的。庆幸的是，Xilinx推出的Zynq系列是ARM和FPGA相结合的架构，本文中使用的也是软硬件协同的设计方法，可以把选择部分在ARM 的PS部分实现，而且通过Xilinx的SDK工具，我们可以用C语言实现轮盘赌的策略和精英保留策略的结合，简单并可扩展性强，演化速度加快并且稳定性比锦标赛算法好。

### 3.4.3 交叉策略

遗传算法中的核心就是基因的重组（交叉）或者新基因的产生（变异），基因重组的实现方法就是通过交叉操作，所谓交叉操作就是将父代个体的部分结构通过替换重组形成新的个体。重组交叉策略有单点交叉，多点交叉，均匀交叉，洗牌交叉，缩小代理交叉等交叉的方法。图3.5和图3.6分别是单点交叉和多点交叉的示意图。

单点交叉的示意图如图3.5所示，单点交叉即对交叉位置之后的基因片段进行重组，单点交叉是最简单也是应用最广的交叉方法。对于演化硬件的编码方式来说，单点交叉后不会使编码违背笛卡尔遗传编程的规则。如果交叉点的基因片段是FE矩阵第i列的连接项，则两条染色体都是FE矩阵第i列的连接项，满足λ为1时，输入连接项只能

来自前一列的要求，如果是功能项，则两条染色体都是功能项，也都是在功能项的范围变化，符合笛卡尔遗传编码规则，适用于演化硬件。

图3.6是双点交叉的示意图，双点交叉的意思就是有两个交叉点*PTa*和*PTb*，将交叉点之间的基因片段进行重组。双点交叉也适用于演化硬件，交叉后仍然符合笛卡尔遗传编程规则。双点交叉比单点交叉要复杂一点，适用的范围不同，当演化的初级阶段，适应度增长较快，单点交叉可以适用，当演化接近最优解的时候，适应度增长变得异常缓慢，此时应该及时改变策略，使用双点交叉可能会有效果一点。

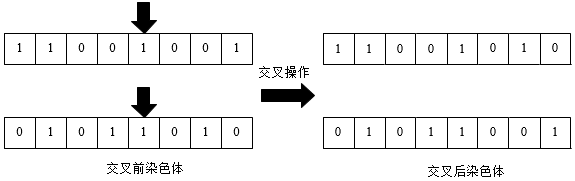


图3.5 单点交叉示意图

均匀交叉和双点交叉一样是有两个交叉点*PTa*，*PTb*把染色体分成3段，只是均匀交叉从0，1，2，中随生成一个随机数，当随机数是0时，将父代染色体的第一段进行交叉，也就是0到*PTa*段，当随机数是1时，将父代染色体的第二段进行交叉，也就是*PTa*到*PTb*段，当随机数是2时，将父代染色体的第三段进行交叉，也就是*PTb*到末端那段。均匀交叉同样适合演化硬件，符合笛卡尔遗传编码的规则。洗牌交叉等会打乱染色体基因组的顺序，使同一列的FE模块可能不是来自前一列FE矩阵，不符合笛卡尔遗传编程规则，不适合演化硬件。

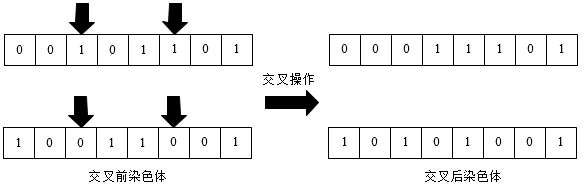


图3.6 双点交叉示意

对于应不应该进行交叉操作，前人也进行了不少的探索研究，Miller在1999年提出了重组好像并不能提高群体的适应度；Walker和Miller在2006提出当多个染色体有独立适应度评估的时候交叉会起到很大的作用；Clegg，Walker，Miller通过实验证明，在

用笛卡尔遗传编程的浮点表示时交叉是有用的[21]。

虽然对于某些情况下交叉并不能提高群体的平均适应度，但是可以显著提高群体最优个体的适应度。在本文中，使用交叉操作能有效提高演化效率。如果没有交叉，只有选择和变异，则最优染色体的演化只能依靠选择变异操作，本项目中，染色体长度最大达到300位，情况众多，一次变异对染色体的影响很小，不能最大程度发挥优秀个体的优良基因。交叉对染色体的改变幅度很大，能显著提高演化速度。

### 3.4.4 变异策略

遗传算法的变异是对染色体基因某位的改变，分为单点变异，多点变异等。单点变异的原理如图3.7所示。

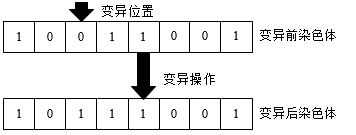


图3.7 单点变异示意图

如图3.7是单点变异的示意图，就是将变异位置的值翻转。多点变异的原理同单点变异，只是变异位置为多个。单点变异是使用最多的变异策略，单点变异和多点变异在进化的不同时刻各有各的作用，比如在进化的开始时刻，适应度增加的很快的时候，单点变异完全能够满足要求，在进化在最后时刻，也就是最大适应度将要接近最优的时候，适应度值增加很慢，这时候如果把变异策略换为多点变异可能会加快进化进程，能够加速演化出最优解。

# 4 基于Zynq的演化硬件平台架构设计

本文设计的演化硬件平台目的是利用软硬件协同的设计方法，探索适用于Zynq-7000系列开发板的平台架构，使得使用此平台能够演化出目标电路。此架构的原理图如图4.1所示。

PL部分

虚拟可重构电路

真值表

Zynq系列

开发板

AXI总线

染色体

适应度值

比较器

PS部分

适应度值

比较结果

适应度值

存储模块

群体存

储模块

选择模块

交叉模块

变异模块

更新

群体

读取

群体

读取

群体

更新

群体

染色体

更新

群体

读取

群体

更新群体适应度

最大适应度

最大适应度

更新群体适应度

更新群体适应度

最大适应度

图4.1 演化硬件平台架构图

图4.1是演化硬件平台架构图，从图中可以看到此平台主要是由PL部分和PS部分组成的，PL部分即可编程逻辑部分，实现的难点是虚拟可重构电路模块，将染色体通过虚拟可重构电路的进行解码输出与真值表的输出循环对比，计算出适应度值。PS部分即处理器部分，主要完成的是遗传算法的过程。遗传算法的过程如下：

（1）判断适应度存储模块内最大适应度的值是否达到最优或者演化了指定代数。如果是，则执行（6），否则执行（2）。

（2）进行选择操作，选择操作的输入是群体数组和适应度数组，使用选择策略后的输出是更新后的群体和适应度。

（3）交叉模块从最新的群体里逐个按照交叉概率进行交叉，交叉后更新群体更新适应度存储模块。

（4）变异模块从最新的群体里逐个按照变异概率进行变异，变异后更新群体更新适应存储模块。

（5）计算新群体的适应度，更新适应度存储模块，执行（1）。

（6）如果适应度达到了最大，则读出最优染色体。如果演化了N代还不能达到最大，则读出此时群体的最大适应度。

图中每次都要更新适应度，并且都要读取最大适应度，因为对于简单遗传算法不需要这样，但是简单遗传算法的效率很低，使用简单遗传算法很容易将适应度大的个体经过交叉变异等操作变得越来越小，所以框架中每次都保留群体最新的最大适应度，将适应度最大的个体保留下来。从图中还可以看到PL和PS之间是通过AXI总线进行交互的。PL部分传给PS部分适应度值，PS部分传给PL部分染色体。架构中每个模块的详细设计将在以下章节内结合本文设计的实验案例（3/8位奇偶校验器，2位乘法器）进行介绍。

## 4.1 可编程逻辑部分设计

可编程逻辑部分的设计主要是虚拟可重构电路的实现和适应度计算模块的加速。适应度计算模块调用虚拟可重构模块进行染色体的解码。由于适应度计算模块是最费时间的一项，对于n位输入的电路来说输入的可能性有2n种，也就是输入的情况是指数增长。如果每个输入组合的输出值都要通过总线传输到软件部分，由软件部分来计算适应度，对于输入组合少的电路来说，放到可编程逻辑部分和处理系统部分并没有明显的时间开销上的差异，但是对于输入情况多的电路，每一种虚拟可重构电路的输出都要通过总线送到处理器部分进行处理，时间开销增大。本文使用的是内部演化，内部演化就是需要把所有的染色体都送到硬件部分进行配置，所以本文的设计把计算适应度模块在硬件里实现。

### 4.1.1 虚拟可重构电路设计

虚拟可重构电路部分的设计的主要思路是由一个主模块构成VRC和若干个从模块构成，对于组合电路的演化，主模块的主要输入是染色体的编码和此电路的输入可能情况，然后模块根据染色体表示的电路的功能输出对应的输出值，可能输出的结果并不与该组合电路应该输出的结果相等，这就需要进行适应度计算和演化。

虚拟可重构电路部分设计的难点是FE矩阵的选择，具体选择多大规模的FE矩阵取决于目标电路的输入输出规模和FE的功能列表。如果FE的功能列表太简单，则需要的FE矩阵规模比较大，如果FE的功能列表较复杂，则FE矩阵的规模适当即可。比如对于3位奇偶校验器，Out = a ^ b ^ c，如果功能列表中有异或，则只需要2个异或门即可，如果没有异或门，Out = (~a) & (~b) & c | (~a) & b & (~c) | a & (~b) & (~c) | a & b & c，只是普通的与或门，则至少需要8个门，也就是FE的个数至少是8。

3位奇偶校验器为例说明的FE矩阵和功能列表如图4.2所示。

A

B

C

FE00

FE01

FE10

FE11

FE20

FE21

FE30

FE31

O

Bank0

Bank1

Bank2

Bank3

图4.2 3位奇偶校验器的FE矩阵

表4.1 FE的功能表

|  |  |
| --- | --- |
| 编号 | 功能 |
| 0x00 | A & B |
| 0x01 | A | B |
| 0x02 | A ^ B |
| 0x03 | ~A |
| 0x04 | ~B |
| 0x05 | ~(A & B) |
| 0x06 | ~(A | B) |
| 0x07 | ~(A ^ B) |

图4.2是3位奇偶校验器的FE矩阵，本文设计FE矩阵是2\*4的，其中每一列的数据存储在一个Bank里。每个FE的输入用2位二进制表示即可，功能用3位二进制表示，功能的列表如表4.1所示，那么一个FE是用7位二进制表示，整个FE矩阵长度为2\*4\*7=56位，输出用2位表示，整条染色体的长度为56+2=58位。当VRC使能开始，则将染色体的值存于4个Bank里，每个Bank是7\*2=14位，然后先读取第一个Bank的值，将第一个Bank的值交于2个FE处理，2个FE的输入来自整个VRC的输入，比如对于第一个FE来说输入除了时钟使能外还有i\_data[2:0]，bank0Data[1:0]，bank0Data[3:2]，bankData0[6:4]，其中i\_data[2:0]是3位奇偶校验器的输入，其它三个分别表示FE的输入和功能项，当bank0Data[1:0]=0x00，bank1Data[3:2]=0x02，bank0Data[6:4]=0x07，则表示的功能是column0Data[0]=~(i\_data[0] ^ i\_data[2])。当第一列FE计算完成，使能第2列FE，将第一列FE的输出作为第二列的输入，比如对于第二列的第一个FE来说除了时钟信号外，使能信号是column0Enable，也就是只有当第一列FE计算完成，第二列FE才能开始，输入项还有column0Data，bank1Data[1:0]，bank1Data[3:2]，bank1Data[6:4]，当bank1Data[1:0]=0x00，bank1Data[3:2]=0x01，bank1Data[6:4]=0x03，则表示的功能是column1Data[0]=~column0Data[1]。依次类推，系统输出的值来自第四列FE的输出，比如chromn[57:56]=0x00，则系统的输出结果o\_data=column3Data[0]。

本文中用例的设计采用的是并行的设计方法，也就是在VRC模块里调用了8个FE模块，虽然每列FE之间需要使能，是串行地执行，但是这是功能的需要，同一列FE之间完全是并行的方法，如果使用串行的方法，需要第一列第一个FE使能完成再使能第一列的第二个FE，串行的方法需要严格地控制时钟，复杂且时间开销大，并行的方法编码简单，容易理解，是使用牺牲空间换时间的思想。

### 4.1.2 适应度计算模块设计

适应度计算模块本文采用的是串行和并行相结合的方法，对于输入组合少的目标电路，采用串行的适应度计算方法，对于输入组合多的目标电路，采用并行的适应度计算方法。假设FE矩阵是*m*\**n*，目标电路的输入个数是*c*，则对于串行适应度计算方式流程图如4.3所示，并行适应度计算方式流程图如4.4所示。

图4.3是串行适应度计算部分的流程图，从图中可以看出，一次性使能2*c*个VRC模块，这就是模块的并行，2*c*个VRC模块的输入数据i\_data分别是从0x00，到2*c*-1，然后是串行比较，一个时钟周期比较VRC模块的输出和真值表模块的输出是否相等，如果相等则适应度寄存器值加1，否则进入下一个循环。比如第一个VRC模块的输入i\_data是0x00，输出是1，真值表的输出是0，不相等的情况下，直接计数器累加1，进入下一个循环，比如当第2*c*个VRC模块的输入i\_data是2*c*-1，输出是1，真值表输出是1，则适应度值加1，累加器加1，累加器值正好得2*c*-1，则跳出循环，读出适应度的值，然后结束。对于输入组合少的目标电路，并行执行只是循环比较VRC模块的输出和真值表的输出，时间开销少。

图4.4是并行适应度计算流程图，它是运用串行的设计方法，当计数器值小于2*c*的时候，返回到使能VRC的部分，不是返回到判断和真值表输出值是否相等的部分。模块首先使能一个VRC此时输入数据i\_data的值是0x00，VRC的输出与真值表的输出值对比，如果相等则适应度值累加，计数器和输入值累加，如果不相等则直接跳转到计数器和输

入值累加，然后跳到下一个循环，一直这样直到输入数据i\_data的值是2*c*-1时，使能VRC，对比真值表的输出，循环完最后一次读出适应度值然后结束。虚拟可重构值的计算需要的时间很长，串行的方法对于*c*输入的目标电路需要循环2*c*次，真值表和虚拟可重构电路的输出对比也要进行2*c*-1次，其中虚拟重构电路串行是最耗费时间的，但是如果不串行，并行的话会导致内存开销急速增大，所以本文是使用串行的方法。

开始

使能2c个VRC模块

初始化计数器值

和适应度值

计数器值小于2c

计数器值累加

VRC

的输出和真值表的输出

相等

适应度值累加

读出适应度值

结束

是

否

是

否

图4.3 串行适应度计算流程图

除了适应度计算的串并行问题，还有很重要的就是适应度计算策略问题。对于3/8位奇偶检验器，由于输出只有1位，所以采取的适应度计算策略是对于输入的*N*种情况，如果输出值与真值表值相等则加1。而对于2位乘法器，输出是4位，如果还是输出值与真值

表输出一样则相加这种策略，则演化效率很低，6000代以内能演化出最优解的概率14%左右。本文对于2位乘法器采取的策略是对于4位的输出结果逐位比较，有几位相等则适应度加几，对于4位输出的乘法器，最大适应度是64。采用这种策略，使得2位乘法器在6000代内能演化出最优解概率达到38%，10000代内能演化出最优解概率达到60%。

开始

初始化输入数据，

计数器和适应度值

使能1个VRC

计数器的值小于

VRC

的输出和真值表的

输出相等

适应度值累加

计数器和输入值累加

读出适应度值

结束

是

是

否

否

图4.4 并行适应度计算流程图

## 4.2 处理系统部分设计

处理系统部分的设计主要是遗传算法部分的实现，主要的难点是根据演化硬件的特

点，选择适合演化硬件的选择，交叉变异策略。本小节主要是详细介绍本文采取的选择，交叉变异的策略。

### 4.2.1 选择部分设计

本文选择部分采取的策略是精英保留策略和轮盘赌策略相结合的方式，由于轮盘赌策略可能会丢弃演化很久才演化出的一个局部最优解，虽然这条染色体并不是最优，但是属于当前群体里适应度最大的个体，而精英保留策略容易陷入局部最优，所以本文将两者结合起来进行演化。选择部分的输入变量是种群染色体数*ChromArray*，种群适应度数组*FitnessArray*，当前种群最大适应度下标*CurMaxIndex*。输出变量是新种群最大适应度下标*NewMaxIndex*，伪代码如下所示：

**function** SELECT (*ChromArray*; *FitnessArray*; *CurMaxIndex*)

*NewMaxIndex* ← 0

*i* ← 0

**while** *i* < POPSIZE **do**

**if** *i*! = *CurMaxIndex* **then**

*index* ← ROULETTE(*FitnessArray*)

*chrom\_temp[i]* ← *ChromArray[index]*

*FitnessArray\_temp[i]* ← *FitnessArray[index]*

**else**

*chrom\_temp[i]* ← *ChromArray[CurMaxIndex]*

*FitnessArray\_temp[i]* ← *FitnessArray[CurMaxIndex]*

**end if**

*i* + +

**end while**

*ChromArray* ← *chrom\_temp*

*FitnessArray* ← *FitnessArray\_temp*

*NewMaxIndex* ← MAX(*FitnessArray*)

**return** *NewMaxIndex*

**end function**

**function** ROULETTE (*FitnessArray*)

*partsum* ← PARTADD(*FitnessArray*; POPSIZE – 1)

*i* ← 0

*result* ← 0

**if** *partsum* > 0 **and** *partsum* < *FitnessArray[0]* **then**

*result* ← 0

**else**

**while** *i* < POPSIZE **do**

*temp0* ← PARTADD (*FitnessArray*; *i* - 1)

*temp1* ← PARTADD (*FitnessArray*; *i*)

**if** *partsum* > *temp0* **and** *partsum* < *temp1* **then**

*result* ← *i*

**end if**

*i* + +

**end while**

**end if**

**return** *result*

**end** f**unction**

从伪代码中可以看出选择策略是轮盘赌策略ROULETTE与精英保留策略相结合。每一次先将上一代群体的最优个体传入下一代，然后再进行轮盘赌选择POPSIZE-1个个体组成新的种群。轮盘赌策略的大抵思路是随机生成的数在群体中第i和第i-1个个体的累加和之间，则选择第i个个体。

### 4.2.2 交叉部分设计

交叉部分本文不采用简单遗传算法的群体里的每个个体按照概率进行交叉，因为这样会导致优秀的个体交叉后适应度变得小。本文采用的是把最优的染色体不参与交叉，其它染色体进行交叉，交叉后更新群体和适应度存储模块，再选出适应度最大个体，留给下一代。交叉部分的输入变量是种群染色体数*ChromArray*，种群适应度数组*FitnessArray*，当前种群最大适应度下标*CurMaxIndex*。输出变量是新种群最大适应度下标*NewMaxIndex*，伪代码如下所示：

**function** CROSSOVER (*ChromArray*; *FitnessArray*; *CurMaxIndex*)

*NewMaxIndex* ← 0

*i* ← 0

**while** *i* < POPSIZE **do**

**if** *i*! = *CurMaxIndex* **then**

*j* ← *i* + 1

*index* is intersect position

**if** *j* < POPSIZE && *j* != *CurMaxIndex* **then**

SPINTERSECT(*ChromArray[i]*; *ChromArray[j]*; *index*; *ChromArray[i]*; *ChromArray[j]*)

*FitnessArray[i]* ← *CALCUFITNESS(ChromArray[i])*

*FitnessArray[j]* ← *CALCUFITNESS* (*ChromArray[j]*)

*i* ← *i* + 2

**else**

*i* ← *i* + 1

**end if**

**else**

*i* + +

**end if**

**end while**

*NewMaxIndex* ← MAX(*FitnessArray*)

**return** *NewMaxIndex*

**end function**

本文采用的是精英保留的单点交叉策略。即当前群体的最优染色体被保留下来不参与交叉操作，单点交叉的主要函数是SPINTERSECT (unsigned char \*src0, unsigned char\*src1, int index, unsigned char\*dst0, unsigned char\* dst1)，其中src0和src1是交叉前的母染色体，index是交叉的位置，dsr0和dst1是交叉后的子染色体。遍历一遍染色体，当下标i小于index时，dst0[i]=src0[i]，dst1[i]=src1[i]。当i大于等于index时，dst0[i]=src1[i]，dst1[i]=src0[i]。

### 4.2.3 变异部分设计

变异部分和交叉部分类似，也不能采用简单遗传算法的变异策略，为了防止目前群体中最优染色体变异成适应度较小的染色体，本文采用的是将最优染色体保留，其它染色体进行单点变异的策略。变异部分的输入变量是种群染色体数*ChromArray*，种群适应度数组*FitnessArray*，当前种群最大适应度下标*CurMaxIndex*。输出变量是新种群最大适应度下标*NewMaxIndex*，伪代码如下所示：

**function** MUTATION (*ChromArray*; *FitnessArray*; *CurMaxIndex*)

*NewMaxIndex* ← 0

*i* ← 0

**while** *i* < POPSIZE **do**

**if** *i*! = *CurMaxIndex* **then**

*index* is mutation position

SPMUTATION(*ChromArray[i]*; *index*; *ChromArray[i]*)

*FitnessArray[i]* ← *CALCUFITNESS(ChromArray[i])*

*i* ++

**else**

*i* + +

**end if**

**end while**

*NewMaxIndex* ← MAX(*FitnessArray*)

**return** *NewMaxIndex*

**end function**

变异部分采取的是精英保留的单点变异策略，即当前群体放入最优染色体被保留下来，不参与变异操作，变异操作的主要函数SPMUTATION (unsigned char\*src, int index, unsigned char\*dst)，其中src是变异前染色体，index是变异的位置，dst是变异后染色体。对于变异后的值一定要与变异前的值不相同，如果相同，则一直生成随机数，直到生成随机数的值和变异前的值不同为止。

## 4.3 通信部分设计

软硬件通信部分主要是使用AXI总线，需要使用的配置工具是PlanAhead和Xilinx Platform Studio。软硬件通信部分主要的流程是按照软硬件协同设计的思路用PlanAhead工具建立嵌入式工程，把硬件部分设计完成的文件导入工程里，在开发工具Xilinx Platform Studio里选择要使用的开发资源，然后在嵌入式工程里把硬件部分进行综合实现生成比特流，再把生成的文件导入到软件开发套件里，系统就自动生成了驱动文件，软件部分通过驱动文件和硬件部分进行交互。下文将详细介绍通信部分设计的流程和配置的具体步骤。

### 4.3.1 用户逻辑部分建立和配置

首先用PlanAhead建立嵌入式源文件，也就是xmp文件，然后用Xilinx Platform Studio工具打开新建的源文件。在XPS里点击Hardware->Create and Import Peripheral Wizard新建IP核。选择生成新的外设模板，定义新IP核的名字是ga\_io，然后就是选择连接的总线，可连接的总线图4.5所示。

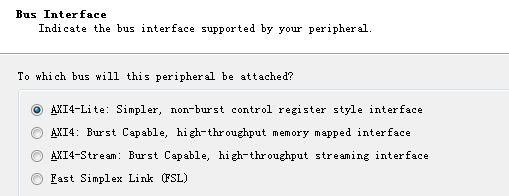


图4.5 用户自定义IP核总线图

图4.5是PL部分用户自定义IP和PS部分通信的时候可以使用的总线图，从图中可以看出主要有4种，分别是AXI4-Lite总线，AXI4总线，AXI4-Stream总线和快速单项链路。其中AXI4-Lite总线简单，无突发控制寄存器类型的接口。AXI4总线有高吞吐量的内存映射接口。AXI4-Stream是有高吞吐量的流接口。

本文将新建的外设连在AXI4-Lite总线上，AXI总线接口服务示意图如图4.6所示。

图4.6是AXI总线接口服务示意图，从图中可以看出AXI总线有AXI主模块和AXI从模块，其中主模块主要是针对复杂的外设比如PCI和Ethernet，AXI的从模块主要是用于大部分的外设，比如逻辑控制，数据缓存，内存访问等，本文不适用主模块的支持，只使用从模块。

图4.7是用户逻辑块结构示意图，从图中可以看出用户逻辑模块里需要定义软件可访问寄存器的数量，这里的寄存器是为软件程序提供的用于控制和监视用户逻辑状态的，可以是一个字节，一个字，半字或者双字大小，寄存器通过Bus2IP\_RdCE信号来读寄存器，通过Bus2IP\_WrCE信号来写寄存器，本文中主要用寄存器来存储染色体和控制信号。对于3位奇偶校验器来说，如果FE矩阵是2\*4阵列的，则1个32位的寄存器可以存储4个FE，2个32位的寄存器就可以存储8个FE外加一个2bit的输出，然后需要1个寄存器来存储适应度结果，还需要一个寄存器来使能。这样一个3位的奇偶校验器就需要4个寄存器，其中slv\_reg0用于读出适应度计算的结果，slv\_reg1用于使能，slv\_reg2和slv\_reg3用于存储FE矩阵和输出。对于8位的奇偶校验器，如果FE矩阵是8\*4，则一个FE 3\*2+3=9位，1个寄存器只能存储3个FE。32个FE和1个3位的输出需要11个寄存器，其中最后一个寄存器存储的是最后2个FE和1个3位的输出，还需要1个使能寄存器和一个存储适应度的寄存器，总共需要13个寄存器。其中slv\_reg0是只读寄存器，用于读出适应度，slv\_reg1是使能寄存器，slv\_reg2到slv\_reg12是只写寄存器，用于存储染色体。

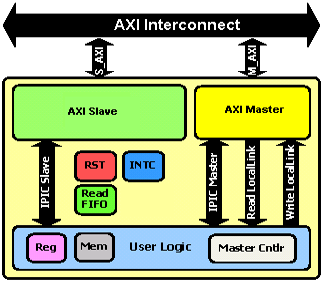


图4.6 AXI总线接口服务示意图

通过上述步骤，系统自动生成了user\_logic.v文件，还有ga\_io.vhdl文件。其中user\_logic.v文件就是上述用户逻辑，ga\_io.vhdl需要实例化user\_logic模块。在user\_logic模块中，进行的操作如下：

（1）定义输出信号LED[7:0]，这个LED用于演示演化过程中的适应度，这个LED的输出可以有也可以没有，毕竟软件部分取得适应度是通过slv\_reg0，本项目中用LED显示适应度可以更直观地表示适应度的变化。

（2）在源码中实例化calculateFitness模块，此模块在硬件设计中已经实现仿真完成，

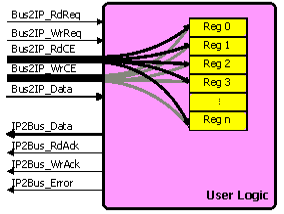


图4.7 用户逻辑块结构示意图

将calculateFitness模块的clk信号实例化城Bus2IP\_Clk，nrst信号实例化成Bus2IP\_Resetn，使能信号是slv\_reg1延迟2个时钟周期后的信号sw，因为对于3位奇偶校验器，2\*4的FE矩阵，染色体用2个寄存器就能存储，所以使能信号要延迟2个时钟信号等待写寄存器完成。染色体是{slv\_reg3[29:0], slv\_reg2[27:0]}，其中slv\_reg3存储的是后4个FE和输出总共30位，slv\_reg2存储的是前4个FE总共28位。

（3）计算适应度模块有两个输出fitness和valid，当valid等于1时，将fitness的值赋给slv\_reg0，并且用assign LED=slv\_reg0，将slv\_reg0的值赋给LED用以在开发板中显示适应度。

（4）由于slv\_reg0是只读寄存器，所以在写寄存器片选循环中将slv\_reg0的赋值隐掉。4个寄存器都是可以读的，所以在读片选里不需要隐掉寄存器部分。

源文件按照上述步骤实例化完成后，开始进行配置文件的修改。在介绍配置文件修改之前，先介绍IP核源文件的结构。本文定义的IP核名字是ga\_io，系统自动生成的文件夹是ga\_io\_v1\_00\_a，在此文件夹下有三个子文件夹，分别是data，devl，hdl，其中data文件夹里是配置文件，devl文件夹里是工程文件，hdl文件夹里是源文件，需要把硬件部分设计的源文件拷贝到此文件夹里。data文件夹里有ga\_io.xst，ga\_io\_v2\_1\_0.mpd，ga\_io\_v2\_1\_0.pao文件，修改步骤如下：

（1）打开ga\_io.xst文件，把硬件设计部分的源文件，按照此种verilog ga\_io\_v1\_00\_a "../hdl/verilog/calculateFitness.v"格式添加到user\_logic.v下面。

（2）打开ga\_io\_v2\_1\_0.mpd文件，在外设选项里添加OPTION RUN\_NGCBUILD = TRUE和OPTION STYLE = MIX，并且在端口一栏里添加PORT LED="", DIR = O, VEC = [7:0]。

（3）打开ga\_io\_v2\_1\_0.mpd文件，在lib ga\_io\_v1\_00\_a user\_logic verilog下一行添加硬件设计部分源文件的路径，按照lib ga\_io\_v1\_00\_a XXX verilog的格式。

（4）在data文件夹里新建ga\_io\_v2\_1\_0.bbd文件，在文件中添加指向IP核的语句，格式是FILES ../hdl/verilog/ipcore\_dir/ram.ngc因为在源文件里使用了ram的IP核，IP核的配置方法和普通源文件配置方法不一样，需要添加bbd文件，还需要在源码里实例化ram的前面添加(\* box\_type = "user\_black\_box" \*)，比如硬件设计部分cfgBank实例化ram，就需要在cfgBank模块里调用ram模块的前面添加这一句话。

### 4.3.2 IP核添加和设置

上述配置完成之后，在ISE开发环境部分进行综合实现，综合实现的结果完全正确的情况下，打开XPS，选择工程里的扫描用户存储，会在IP Catalog里显示自定义的IP核，然后添加IP核。

图4.8是IP核端口示意图，其中LED是8位的输出端口。IP核地址分配如图4.9所示。

勾选出右边部分，ga\_io\_0的命名方式表示是第一个名字为ga\_io的IP核，从图中可以看出此IP核的地址是从0x6FC00000到0x6FC0FFFF，分配的空间大小是64KB，使用的总线是S\_AXI也就是AXI的从模块。最后点击生成地址，就会生成IP核的地址。

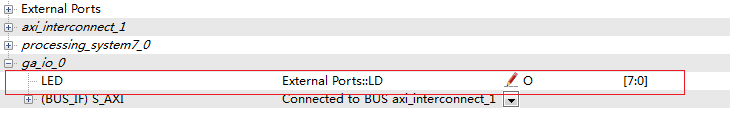


图4.8 IP核端口示意图

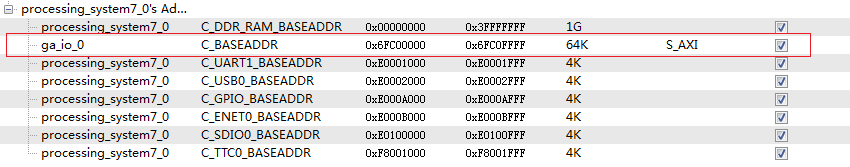


图4.9 IP核地址示意图

IP核地址生成完毕后，关闭XPS，在PlanAhead里面选中GA.xmp，生成顶层文件，系统自动生成GA\_stub.v，然后添加约束文件，综合仿真实现生成比特流文件。

### 4.3.3 处理系统部分调用

在PlanAhead工程里生成bit流文件后，选择导出硬件到SDK，就会自动生成驱动文件和硬件平台文件。驱动文件在工程源文件/edk/GA/drivers目录下，ga\_io\_v1\_00\_a/src目录下有ga\_io.c，ga\_io.h，ga\_io\_selftest.c三个文件，软件部分通过调用ga\_io.h里面的函数实现和硬件部分的通信。硬件平台文件在sdk的GA\_hw\_platform文件夹，里面包含比特流文件，系统初始化文件等。

在SDK环境下建立应用工程，选择硬件平台是GA\_hw\_platform，处理器是ps7\_cortexa9\_0。然后在界面里单击Next选项，选择Hello World模板，系统自生成helloWorld.c源文件。在源文件目录里加进drivers/ga\_io\_v1\_00\_a/src目录下的ga\_io.c，ga\_io.h和ga\_io\_selftest.c文件，因为这些文件在源码里需要调用。

将之前设计好的软件部分的源码替换helloWorld模板部分内容，并且加上calFitness.c文件，在计算适应度文件部分，需要先通过转换函数将染色体转换成uint类型数，然后将数据写进寄存器。对于2\*4FE阵列的3位奇偶校验器来说，就是写寄存器slv\_reg2和slv\_reg3。然后忘寄存器slv\_reg1里写进1，最后读出寄存器slv\_reg0的值就是适应度。写寄存器函数在ga\_io.h，按照这个文件提供的函数调用形式GA\_IO\_mWriteSlaveReg2 (XPAR\_GA\_IO\_0\_BASEADDR, 0, reg2\_data)进行函数调用;其中XPAR\_GA\_IO\_0\_ BASEADDR是基地址，0是地址偏移，reg2\_data是要写进去的数据。读寄存器的格式是GA\_IO\_mReadSlaveReg0(XPAR\_GA\_IO\_0\_BASEADDR,0);其中第一个参数是基地址，第二个是地址偏移。

需要注意的是赛灵思的开发工具还有很多缺陷不完善，需要手动修改。比如自动生成的文件ga\_io\_selftest.c里面，#include”xio.h”不能识别需要改成#include”xil\_io.h”。并且需要在常数定义部分加上#define GA\_IO\_USER\_NUM\_REG TEST\_AXI\_LITE\_USER\_ NUM\_REG。

# 5 基于演化平台的实验案例验证

根据上一章设计的演化硬件平台架构，本章设计了基于该平台架构的实验案例来验证此平台架构的性能。主要的实验案例是3/8位奇偶校验器，2位乘法器。实验案例的设计出了上述详细的架构外，还需要配置个部分的资源，本章将从硬件部分配置，软件部分配置和实验结果分析部分进行详细说明。

## 5.1 硬件部分配置

硬件部分的配置如图5.1所示。

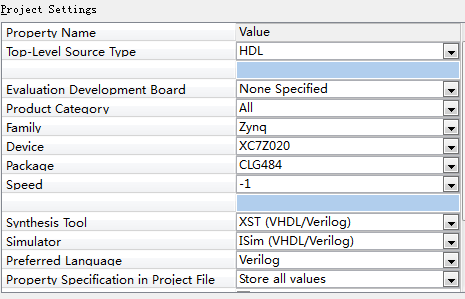


图5.1 硬件部分配置图

图5.1是硬件部分的配置图，从图中可以看出实现的语言是Verilog HDL，实现的工具是Xilinx的ISE Design Tool，仿真软件是ISim，综合工具是XST，设备是XC7Z020 clg484。

## 5.2 软件部分配置

软件部分实现的工具是Xilinx Software Development Kit，软件部分配置首先需要解决的是头文件的添加和设置，软件部分主要有三个文件夹组成，分别是ga，ga\_bsp和GA\_hw\_platform。这些文件夹的详细介绍如下。

（1）文件夹ga是软件部分的源文件和执行文件部分，C代码源文件在ga/src文件夹

里，硬件件部分生成的驱动文件ga\_io.c等也需要放到这个文件夹下。ARM编译器编译生成的.elf文件在ga/debug目录下。

（2）文件夹ga\_bsp是板级支持包文件，里面主要是引用的库文件，工具自动生成的板级支持包缺少xbasic\_types.h，使得某些数据类似等。

（3）文件夹GA\_hw\_platform主要是硬件平台文件，其中system.bit是硬件设计部分生成的bit文件，其它文件都是在导出到SDK的过程中自动生成的。

实验设计除了架构的问题，在具体实现的过程中还有存储和转换的问题要考虑。染色体的存储格式在硬件部分就是二进制串，但是在C语言部分串的格式要根据演化操作的基本单位来决定。演化操作的基本单位是FE的连接项或者功能项。这里考虑的染色体的一个基因不是一个功能单元（FE），而是FE的基本组成部分（输入项，功能项），一个FE不能作为一个基因，因为比如对于3位奇偶校验器的变异操作，变异的如果是输入项，则需要在0到2范围内或者0到1范围内变化，如果变异的是功能项则需要在0到7范围内变化。

染色体的存储转换主要是为了把软件部分存储结构转换成硬件部分寄存器的存储格式。染色体在软件部分是用十进制表示的，在硬件部分是用二进制表示的。比如对于3位奇偶校验器的染色体表示chromn[25]={0，1，7，0，2，3，1，2，5，0，1，2，…，0，1，6，1}，染色体在硬件部分是用多个32位的寄存器存储的，1个FE是2+2+3=7位，1个寄存器可以存储4个FE，所以我们需要把4个FE也就是12个元素合并成一个数，写进寄存器里。存储4个FE需要32位类型的数据，在系统提供的Xbasic\_types.h的文件里有uint32类型，对于2\*4 FE矩阵的三位奇偶校验器，需要2个uint32类型的数据来存储，我们就定义uint32 data0来存储第一个寄存器的内容，data0转换的基本步骤如下：

（1）data0初始化为0，data0赋值chromn[11]。

（2）如果下一个数据是输入项则data0乘以2^2，也就相当于2进制的左移2位，再加上这个输入项数据。如果遇到功能项，则data0乘以2^3，也就是2进制的左移3位，再加上这个功能项数据。

（3）不停循环步骤（2）操作，直到4个FE的12个元素都累加完毕为止。

对于第二个寄存器的内容，也就是存储后4个FE和输出的寄存器内容，转换原理类似第一个寄存器。

## 5.3 实验结果分析

### 5.3.1 硬件仿真结果

3位奇偶校验器仿真结果如图5.2所示。8位奇偶校验器的仿真结果如图5.3所示。

2位乘法器的仿真结果如图5.4所示。

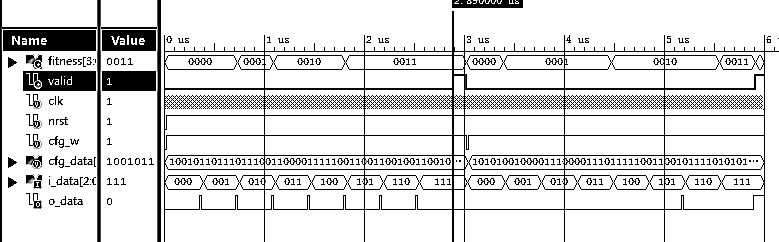


图5.2 3位奇偶校验器仿真结果

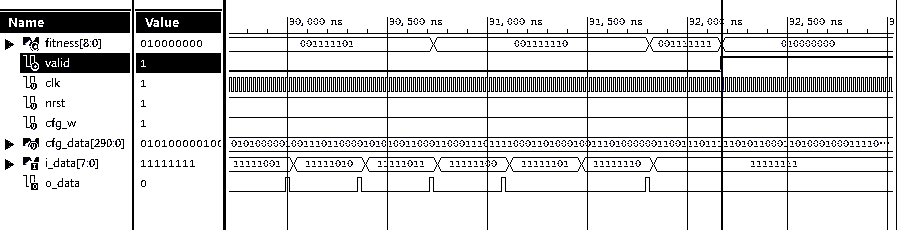


图5.3 8位奇偶校验器仿真结果

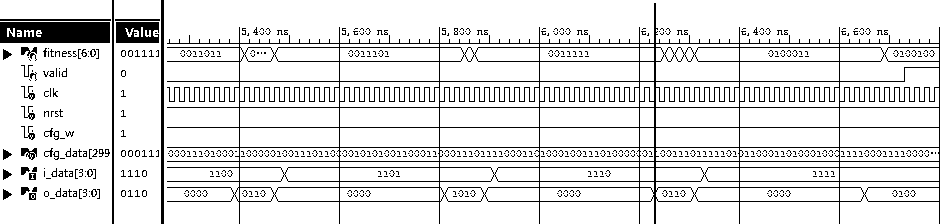


图5.4 2位乘法器仿真结果

从图中可以看出如果设置时钟周期是20ns，利用串行的方法，3位奇偶校验器可以在3us内计算出染色体的适应度，8位奇偶校验器在100us内能计算出染色体的适应度，2位乘法器在6us内能计算出适应度。当valid等于1的时候适应度计算完毕，i\_data是输入数据，对于3位奇偶校验器，输入数据就是从0到7，o\_data是输入数据经过虚拟可重构电路后的输出数据，虚拟可重构电路的输出如果是正确的，则适应度加1，如果不正确，则适应度不变。比如8位奇偶校验器中，当输入是8’b11111101时，输出是0，适应度不变，当输入是8’b11111110时，输出是1，适应度加1。对于8位奇偶校验器，输入数据从0增长到255，所以运行的时间更长。对于2位乘法器，最大的适应度。

### 5.3.2 下载至开发板运行结果

软件部分是在开发板的ARM处理器端运行，结果通过UART串口传到控制台，串口的波特率是115200。实验案例针对3位奇偶校验器，随机生成4个群体，1000代内演化出的最优染色体对应的电路图如图5.5所示。

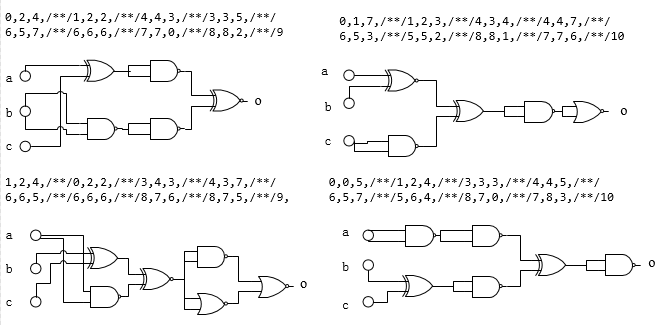


图5.5 最优染色体对应的目标电路图

图5.5是4个最优染色体对应的目标电路图，从图中可以看出需要的有效逻辑门的数量在6个左右，也就是有部分逻辑门可能是不影响输出的逻辑门。并且从图中还可以看出，虽然4个目标电路表示的都是三位奇偶校验器的逻辑功能但是结构各不相同，也就是目标电路都是异构的，这意味着此演化平台演化出的目标电路可以用于三模冗余容错系统设计，异构的电路同时出错的可能性要低于同构的电路。

图5.6是8位奇偶校验器的最优染色体对应的目标电路图，图中的电路是上方染色体代表的目标电路。从图中可以看出总共使用了12个逻辑门，8位奇偶校验器演化使用的是8\*4的FE矩阵，也就是有20个逻辑门是无效的。

### 5.3.3 实验结果总结

本文设计了3位奇偶校验器，还设计了8位奇偶校验器和2位乘法器来验证演化算

法的有效性。影响演化效果的因素主要是FE矩阵的规模，适应度计算策略和演化策略，

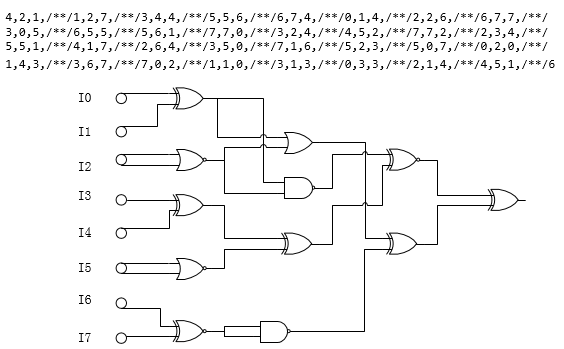


图5.6 8位奇偶校验器最优染色体对应的目标电路图

演化策略采用的是上一章平台设计部分描述的策略，而FE矩阵则要根据目标电路的规

模，进行经验判断，然后试出效果最优的FE规模，适应度计算的策略要根据目标电路的输入输出端口个数来确定。

其中三位奇偶校验器演化我们设置演化终止条件是演化达到了1000代或者最大适应度8，实验的群体个数是500，每个群体的规模是16，使用的矩阵规模是2\*4的FE矩阵，策略是精英保留的交叉变异策略，并使精英保留的轮盘赌选择和精英保留策略相对比，实验演化结果如图5.7所示。

图5.7是3位奇偶校验器的演化结果图，从图中可以看出在10代以内，100代以内，300代以内，500代以内精英保留结合轮盘赌策略能演化出最优解的概率大于单纯的精英保留策略。1000代以内虽然精英保留结合轮盘赌策略概率略小于轮盘赌策略，但是整体上能演化出最优解的概率都达到了98%以上，本文只是随机生成了500个群体，可能有随机性。从整体上看，精英保留加轮盘赌策略优于单纯的精英保留策略，并且精

英保留结合轮盘赌策略的平均适应度大于单纯的精英保留策略。本文也尝试了单纯的轮盘赌策略，由于使用单纯的轮盘赌策略，由于最优染色体和其它染色体的适应度差别不大，特别是接近最优解的时候可能只是稍微有差别，所以很有可能把最优染色体排除掉，所以效果并不好。

图5.8是8位奇偶校验器的演化结果，其中演化使用的是8\*4的FE矩阵，群体规

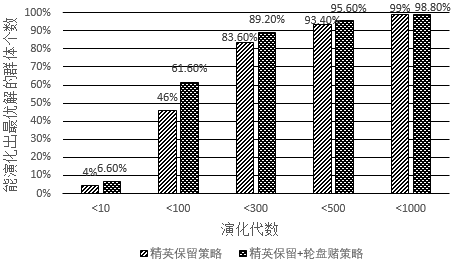


图5.7 三位奇偶校验器演化结果图

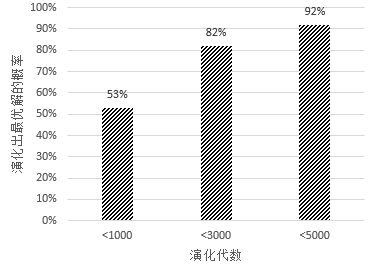


图5.8 8位奇偶校验器演化结果图

模是16，演化的终止条件是演化达到了5000代或者达到最大适应度256，实验中随机生成了100个群体。结果显示，有92个群体能在5000代内演化出最优解，有82个群体能在3000代以内演化出最优解，有53个群体能在1000代内演化出最优解。实验也尝试了4\*4的FE矩阵，在3000代以内适应度值不仅没有达到最大值，连增加的迹象也没有，5000代以内适应度值没有一次达到最大值。

图5.9是2位乘法器采用4位输出结果逐位比较，适应度逐位相加的演化结果图，从图中可以看出10000代内能演化出最优解的概率达到60%，6000代内能演化出最优解的概率达到38%，3000代内能演化出最优解的概率是10%。

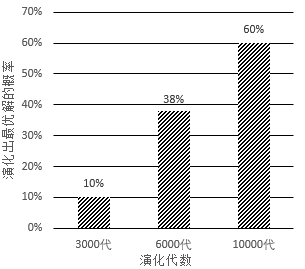


图5.9 2位乘法器演化结果图

2位乘法器的演化采用的FE矩阵是8\*4，因为采用4\*4的矩阵很难演化出最优解，甚至适应度值增加都很难。群体规模是16，实验中随机生成了50个群体。如果适应度的计算策略是4位输出与真值表输出相等则加1的方法，6000代内只有14%的群体达到了最优解，对于演化50次来说，也就是7个。3000代以内只有6%能演化出最优解，1000代以内能演化出最优解的概率是0，演化效率明显小于4位演化结果几位相等，适应度加几的演化策略。实验中发现，在演化的后期，演化陷入了停滞的状态，特别是在演化接近最优解时，群体最大适应度增长的速度特别缓慢。

# 结 论

本文主要利用软硬件协同的设计方法，基于Zynq-7000系列开发板设计了演化硬件系统的平台架构。其中在Zynq的PL部分实现了虚拟可重构电路和适应度计算的加速，在PS部分实现了遗传算法的群体初始化，选择交叉变异等操作，该设计方法充分考虑了Zynq-7000系列以处理器为中心的架构，既避免了纯硬件设计的复杂和扩展性低的特点，又避免了纯软件设计的速度慢的特点，能在特定的时间内演化出最优解。本文按照先易后难的顺序，依次实现了3位奇偶校验器，8位奇偶校验器和2位乘法器的演化设计，本文的主要工作如下：

（1）分析了演化硬件技术的特点，虚拟可重构电路的概念和笛卡尔遗传编程的规则。基于上述概念在可编程逻辑部分实现了虚拟可重构电路和适应度计算。本文的适应度计算采取的策略是对于不同的输入组合，输出与虚拟可重构电路的输出有n位相等，则适应度加n。

（2）分析了现有遗传算法的选择，交叉，变异策略等，并且探索出适用于演化硬件的精英保留结合轮盘赌的选择策略，精英保留的单点交叉变异策略，并将这些算法在处理系统部分实现。

（3）分析了Zynq平台的特点，并且利用AXI从总线，通过自定义用户逻辑块实现了可编程逻辑部分和处理系统部分的通信。

由于时间的限制，本文实现的演化硬件系统平台还有一下需要完善的地方：

（1）演化硬件系统的案例都是基本的电路，可以演化复杂的系统，比如图像滤波系统等，图像滤波系统的演化需要定义更为复杂的功能表，和更为复杂的适应度计算方法。

（2）从演化的结果可以看出，演化系统在前期适应度增加迅速，到了后期适应度增加出现了停滞的现象，我们需要采用更加灵活的选择交叉和变异策略，比如在后期采用多点交叉，多点变异等策略来加速最优染色体的演化。

# 参 考 文 献

[1] 魏巍.电子电路并行演化算法的设计与实现[D].武汉:中国地质大学,2006.

[2] 徐阳,王友仁,等.演化硬件理论与应用技术研究[J].航空电子技术,2003,34(1):43-48.

[3] R Dobai, L Sekanian. Towards Evolvable Systems Based on the Xilinx Zynq Platform [C]//IEEE International Conference on Evolvable System,Singapore,2013,7586: 89 -95.

[4] 陈毓屏,康立山,潘正君,等.一个新的研究领域-演化硬件[J].航空计算技术,1998(1):1-8.

[5] 王婷.基于演化硬件的可重构技术研究[D].郑州:中国人民解放解放军信息工程大学,2012.

[6] D.Levi, SA Guccione.Evolving stable circuits on mainstream FPGA decices[C]// Proceedings of NASA/DoD Conference on Evolvable Hardware, Washington,1999,12-17.

[7] Miguel Garvie, Adrian Thompson. Evolution of Self-Diagnosing Hardware[C]//The 5th International Conference on Evolvable System, Trondheim, Norway, 2003, 2606: 238-248.

[8] Adrian Thompson. Silicon evolution[C]//Proceedings of Genetic Programming, Stanford University,1996,444-452.

[9] Jason Lohn, G.Hornby, etc al. Evolvable Hardware for Space Applications[C]// Proceedings of AIAA 1st Intelligent Systems Technical Conference, Lausanne, Switzerland, 2004,20-22.

[10]平建军,王友仁,等.数字演化硬件的函数级在线进化技术研究[J].高技术通讯,2009,19(1):61-65.

[11]徐阳,王友仁.硬件演化原理及实现方法研究[J].计算机测量与控制,2003,11(5):385- 394.

[12]高桂军,王友仁,姚睿,等.基于演化硬件的容错系统设计技术研究[J].信息与控制2008,37(3):370-376

[13]康立山,何巍,陈毓屏.用函数型可编程器件实现演化硬件[J].计算机学报,1999,22 (7):781-784.

[14]赵曙光,杨万海.一种用于演化硬件的染色体编码新方法[J].西安电子科技大学学报, 2000,27(6):778-780.

[15]王国庆,张剑炜,等.演化硬件在容错系统中的应用[J].测控技术学报,2004,8(18):11-15.

[16]Pauline C. Haddow, Andy M. Tyrrell. Challenges of evolvable hardware，past, present[J].[Genetic Programming and Evolvable Machines](http://link.springer.com/journal/10710),2011,12(3):183-215.

[17]李杰,辛明瑞.演化硬件导论-自适应系统设计实践指南[M].西安:西安电子科技大学出版社,2013.

[18]J.Wang, Q.S.Chen, C.H.Lee. Design and implementation of a virtual reconfigurable architecture for different applications of intrinsic evolvable

hardware[J].Iet Computers & Digital Techniques,2008,2(5):386-400.

[19]OE Elnokity, II Mahmoud, MK Refai, et al. Hardware Implementation of Virtual Reconfigurable Circuit for Fault Tolerant Evolvable Hardware System on FPGA[J]. Lecture Notes in Engineering & Computer Science,2014,2(1):7-10.

[20]杨长雷,朱明程.用于进化硬件的遗传算法的选择策略初探[N].深圳大学学报:理版,2004,21(4):306-309.

[21]JF Miller. Cartesian Genetic Programming[J].[IEEE Symposium on Computers & Informatics](http://xueshu.baidu.com/s?wd=confuri%3A%28b7c9c6f2098c2f30%29%20IEEE%20Symposium%20on%20Computers%20%26%20Informatics&tn=SE_baiduxueshu_c1gjeupa&ie=utf-8&sc_f_para=sc_hilight%3Dpublish&sort=sc_cited),2010,10(2):17-34.

# 致 谢

行文至此，我的这篇论文已接近尾声；岁月如梭，我四年的大学时光也即将敲响结束的钟声。离别在即，站在人生的又一个转折点上，心中难免思绪万千，一种感恩之情油然而生。

首先，我要对我的论文指导教师王洁老师表示衷心的感谢。王洁老师做事严谨，为人和蔼可亲，认真负责，一直以来都是我学习的榜样，感谢他一年来对我的悉心指导，无论是鼓励还是批评，我都从中获益良多，他的谆谆教诲和鞭策将是我前进道路上的明灯，并且给我的论文提供了很多宝贵的意见。

感谢所有辛勤教导我的老师们，他们在我求学路上孜孜不倦传道授业解惑，同时还要感谢实验室的学长，其中要特别感谢柳继委学长，感谢柳继委学长在我迷茫的时候为我指明方向，并且耐心地给我讲解不懂得知识，使我的毕业论文能够顺利完成。最后我要向我的父母和家人表示诚挚的谢意，他们无私地帮助我，做我经济的强力支柱，使我能够在学费如此贵的地方还能安心学习，并且是我的家人在我迷茫，无助，心浮气躁的时候为我指明前进的方向，让我不再追求很多东西。

再一次感谢所有曾经给予我指导和关心的老师、家人、朋友以及同学！