《FPGA与图像处理应用》课程

设计报告

<姓名>

北京至芯科技FPGA培训课程

2016年7月2日星期六

目录

[一、长整加速器设计 3](#_Toc455233921)

[1.1 设计概要 3](#_Toc455233922)

[1.2 顶层架构设计 4](#_Toc455233923)

[1.3 地址映射表 5](#_Toc455233924)

[1.4 加速器从机寄存器设计 5](#_Toc455233925)

[1.4.1 源数据首地址寄存器ASS 5](#_Toc455233926)

[1.4.2 目标数据首地址寄存器AST 5](#_Toc455233927)

[1.4.3 状态寄存器S\_REG 6](#_Toc455233928)

[1.5 CPU的状态转移图 6](#_Toc455233929)

[1.6 仲裁器Arbiter的状态转移图 7](#_Toc455233930)

[1.7 加速器ACC的状态转移图 7](#_Toc455233931)

[二、阅读和欣赏（加速器+图像处理+算法） 8](#_Toc455233932)

[三、流水作业方案的sobel图像滤波器 8](#_Toc455233933)

[四、练习和竞赛 8](#_Toc455233934)

# 一、长整加速器设计

## 1.1 设计概要

设计需求：

1. 能够执行64比特的长整加速运算（非指令实现，用特定的硬件逻辑实现），原则上执行加减乘除，最基本目标能够执行64位无符号长整加法运算。
2. 具有现代加速器架构的基本体系特征：CPU能够访问存储器资源；CPU能够访问ACC；ACC能够访问存储器；
3. 该系统的数据总线宽度为8bit
4. 该设计中的CPU可以是一个虚拟CPU（不必写指令），该CPU在置位后，仅执行对ACC的命令发布，以及等待ACC的中断。
5. 加速器ACC中，必须设置能够接受CPU访问的三个寄存器：源数据首地址寄存器，目标数据首地址寄存器以及ACC的状态寄存器（其中有启动标志位）
6. 加速器与CPU的通信，采用中断方式：ACC完成当前的长整计算任务后，发出上升沿有效的中断，以通知CPU。
7. 无论加减乘除运算，全部忽略溢出（不考虑进位）
8. 存储器地址为16位（65536个字，字宽度为1字节）
9. 仲裁器采用简单的唯一申请者获得仲裁的策略

## 1.2 顶层架构设计

cpu\_data\_in[7:0]

cpu\_data\_out[7:0]

acc\_int

cpu\_addr\_out[15:0]

CPU

（Master）

arb\_res

cpu\_write

cpu\_req

mem\_data\_in[7:0]

MEM

（Slave）

mem\_data\_out[7:0]

mem\_addr\_in

mem\_write

acc\_data\_in[7:0]

ACC

（Master/Slave）

acc\_data\_out[7:0]

acc\_req

acc\_addr\_out[15:0]

acc\_write\_out

acc\_addr\_in

arb\_res

acc\_write\_in

acc\_int

dmux[1:0]

data\_bus

addr\_bus

amux

write\_bus

wmux

amux

awmux\_encoder

cpu\_req

wmux

arb\_res

acc\_req

arbiter

addr\_bus

dmux

acc\_sel

dmux\_encoder

cal\_1604

acc\_sel

多路器选择信号编码器的真值表：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **访问方式** | **arb\_res** | **cpu\_write** | **acc\_write** | **acc\_sel** | **dmux** | **amux** | **wmux** |
| CPU读MEM | `ARB\_CPU | 0 | X | 0 | `DMUX\_MEM | `AMUX\_CPU | `WMUX\_CPU |
| CPU写MEM | `ARB\_CPU | 1 | X | 0 | `DMUX\_CPU | `AMUX\_CPU | `WMUX\_CPU |
| CPU读ACC | `ARB\_CPU | 0 | X | 1 | `DMUX\_ACC | `AMUX\_CPU | `WMUX\_CPU |
| CPU写ACC | `ARB\_CPU | 1 | X | 1 | `DMUX\_CPU | `AMUX\_CPU | `WMUX\_CPU |
| ACC读MEM | `ARB\_ACC | X | 0 | 0 | `DMUX\_MEM | `AMUX\_ACC | `WMUX\_ACC |
| ACC写MEM | `ARB\_ACC | X | 1 | 0 | `DMUX\_ACC | `AMUX\_ACC | `WMUX\_ACC |

## 1.3 地址映射表

|  |  |  |  |
| --- | --- | --- | --- |
| 地址 | 寄存器(名) | 深度（byte） | 说明 |
| 8020H~802FH | S\_REG | 16 | ACC的状态寄存器 |
| 8010H~801FH | ACC\_SLAVE\_TARGET(AST) | 16 | ACC的目标数据首地址寄存器 |
| 8000H~800FH | ACC\_SLAVE\_SOURCE(ASS) | 16 | ACC的源数据首地址寄存器 |
| 0000H~7FFFH |  | 32768 | 存储器地址（地址范围） |
| 0110H~0117H | TARGET\_L，TARGET\_H |  | 验证时目标数据地址 |
| 0100H~0107H | SOURCE\_L，SOURCE\_H |  | 验证时源数据地址 |

## 1.4 加速器从机寄存器设计

### 1.4.1 源数据首地址寄存器ASS

ASSH：8000H

ASSL：8001H

B2

B3

B0

B5

B6

B7

B4

B9

B10

B11

B8

B12

B13

B14

B15

B1

ASS:

ASSH:

B12

B13

B14

B15

B8

B9

B10

B11

ASSL:

B7

B0

B1

B2

B3

B4

B5

B6

### 1.4.2 目标数据首地址寄存器AST

ASTH：8010H

ASTL：8011H

B2

B3

B0

B5

B6

B7

B4

B9

B10

B11

B8

B12

B13

B14

B15

B1

AST:

ASTH:

B12

B13

B14

B15

B8

B9

B10

B11

ASTL:

B7

B0

B1

B2

B3

B4

B5

B6

### 1.4.3 状态寄存器S\_REG

S\_REG: 8020H

B0

OP0

B1

OP1

B2

B3

B4

B5

B6

B7

s\_bit

S\_REG:

说明：

1. ACC的启动标志位S（仅写），CPU写入1则ACC启动，S标志由ACC内部的clear信号同步清除。
2. 操作码(无符号，忽略溢出，忽略进位，除操作忽略小数部分)

|  |  |  |
| --- | --- | --- |
| OP1 | OP0 | 说明 |
| 0 | 0 | 加操作：Target+ Source=>Target |
| 0 | 1 | 减操作：Target- Source =>Target |
| 1 | 0 | 乘操作：Target\* Source =>Target |
| 1 | 1 | 除操作：Target/ Source =>Target |

## 1.5 CPU的状态转移图

说明：

1. `ASSH=8000H; `ASSL=8001H; `ASTH=8010H; `ASTH=8011H; `S\_REG=8020H
2. 验证时目标数据首地址：`TARGET=0110H; `TARGET\_L=10H; `TARGET\_H=01H
3. 1验证时源数据首地址：`SOURCE=0100H; `SOURCE\_L=00H; `SOURCE\_H=01H

s3

cpu\_addr\_out=`ASTH

cpu\_data\_out=` TARGET\_H

cpu\_write=1

s2

cpu\_addr\_out=`ASSL

cpu\_data\_out=` SOURCE\_L

cpu\_write=1

arb\_res!= `ARB\_CPU

cpu\_req=1

s1

cpu\_addr\_out=`ASSH

cpu\_data\_out=` SOURCE\_H

cpu\_write=1

!acc\_int

s4

cpu\_addr\_out=`ASTL

cpu\_data\_out=` TARGET\_L

cpu\_write=1

rst

cpu\_data\_out=0

cpu\_addr\_out=0

cpu\_write=0

cpu\_req=0

s6

cpu\_write=0

cpu\_req=0

s5

cpu\_addr\_out=`S\_REG

cpu\_data\_out=8’b1000000

cpu\_write=1

## 1.6 仲裁器Arbiter的状态转移图

当前仲裁策略：唯一申请者或者仲裁

acc\_req\*!cpu\_req

arb\_res=`ARB\_ACC

rst

arb\_res=`ARB\_CPU

!acc\_req\*cpu\_req

arb\_res=`ARB\_CPU

## 1.7 加速器ACC的状态转移图

s4

acc\_addr\_out= +1

acc\_write\_out=0

source64[7:0]=acc\_data\_in

rst

acc\_data\_out=0

acc\_addr\_out=0

acc\_write\_out=0

acc\_req=0

acc\_int=0

clear=0

source64=0

target64=0

s3

acc\_addr\_out= +1

acc\_write\_out=0

arb\_res!=`ARB\_ACC

acc\_req=1

!s\_bit

s2

acc\_addr\_out=ASS

acc\_write\_out=0

s5

acc\_addr\_out= +1

acc\_write\_out=0

source64[15:8]=acc\_data\_in

s20

target64=target64 OP source64

s22

acc\_addr\_out=+1

acc\_write\_out=1

acc\_data\_out=target64[15:8]

s21

acc\_addr\_out=AST

acc\_write\_out=1

acc\_data\_out=target64[7:0]

s28

acc\_addr\_out=+1

acc\_write\_out=1

acc\_data\_out=target64[63:56]

s30

acc\_int=0

clear=0

s29

acc\_write\_out=0

acc\_int=1

acc\_req=0

clear=1

# 二、阅读和欣赏（加速器+图像处理+算法）

# 三、流水作业方案的sobel图像滤波器

# 四、练习和竞赛