一种基于ARM微处理器的嵌入式代码语义属性分析方法

刘铁铭 蒋烈辉 井 靖 李继中
(信息工程大学信息工程学院 郑州 450002)

摘 要 通过深入研究ARM指令系统的特点及其编译后的代码特征,构建了基于ARM微处理器的二进制嵌入式代码解析模型,讨论了基于ARM体系结构的嵌入式代码语义分析方法。从指令和指令序列两种粒度级别分别讨论了代码语义属性的抽取方法,并分析了基于该解析模型的指令和指令序列的实例。结果表明,此方法极大地提高了代码解析的准确性和可读性。

关键词 ARM微处理器,语义属性,嵌入式代码,编译特征,指令序列
中图法分类号 TP314 **文献标识码** A

Analysis Method of Embedded Code's Semantic Properties Based on ARM Microprocessor

LIU Tie-ming JIANG Lie-hui JING Jing LI Ji-zhong
(Information Engineering College of Information Engineering University, Zhengzhou 450002, China)

Abstract By taking deep research on the characteristic of the ARM instruction system and its compiled codes, we built a binary embedded code analytical model based on ARM microprocessor and discussed the method of embedded code's semantic analysis based on the ARM architecture. We discussed the method of extracting the code semantic properties in terms of the instruction and its sequences respectively, and analysed its instances based on the analytical model. The result shows that this method greatly improves the accuracy and readability of the code analysis.

Keywords ARM microprocessor, Semantic property, Enbedded codes, Compilation characteristics, Instruction sequences

1 引言

当人们无法得到想要的知识、思想和设计理念时,通常采用逆向工程的方法来获取。随着软件业的发展,逆向工程也被引入软件工程领域。1990年,Chikofisky对软件逆向给出了如下定义[1]:软件逆向工程是分析目标系统,认定系统的组件及其交互关系,并且通过高层抽象或其他的形式来展现目标系统的过程。

在嵌入式设备的设计与使用中,ARM微处理器凭借其体积小、低功耗、低成本、高性能等特点,被广泛地运用在工业控制、无线通信、网络应用、消费类电子产品等领域。在分析嵌入式代码的过程中,代码的语义提取及解析是整个分析过程中至关重要的一步,因此研究基于ARM微处理器的嵌入式代码的语义属性分析方法对整个嵌入式代码逆向分析技术来说具有重要的应用价值和意义。

2 ARM源代码编译后的特征

代码语义描述的是程序的行为,同一段程序编译为不同处理器平台下的目标代码具有语义等价性,即这些不同的目标代码具有相同的功能。高级语言源程序编译为ARM二进制代码之后,由于体系结构的差异,虽然代码语义在语法表现形式上有所不同,但一些表现其功能语义的代码片段具有其特定的语义模式。并且这些语义模式不因编译模式而发生改变,这些代码片段构成了编译后的代码特征,这些特征与指令语义分析有着密切的关系。

通过观察大量的代码实例,发现源程序编译为ARM体系结构下的二进制代码后具有以下一些代码特征。

(1)数据类型编译特征

表1 C语言基本数据类型编译映射

C语言数据类型	ARM编译实现	LDR/STR指令后缀
char	unsigned 8-bit 字节	B
short	signed 16-bit 半字	SH
int	signed 32-bit 字	默认无后缀
long	signed 32-bit 字	默认无后缀
long long	signed 64-bit 双字	D

x86体系结构中数据类型通常编译为不同长度的操作数(如EAX/AX/AH/AL),而ARM采用RISC存储/加载结构,算术逻辑指令不能直接作用于存储器变量,变量的类型由LDR/STR指令体现,LDR/ST R指令以不同后缀"B"、"SB"、"H"、"SH"、"D"来区分数据类型[2]。高级语言中变量的基本数据类型与ARM代码中的类型敏感(Type Sensitive)指令存

到稿日期:2011-01-05 返修日期:2011-04-15 本文受国家高技术研究发展计划(863计划)项目(2006AA01Z408,2006AA01Z409),国家高技术研究发展计划(863计划)项目(2007AA01Z483),河南省高新领域重点攻关基金资助项目(082102210011)资助。

刘铁铭(1977-),男,博士生,讲师,主要研究方向为软件逆向工程、信息安全等,E-mail:fxliutm@163.com;蒋烈辉(1967-),男,教授,博士生导师,主要研究方向为逆向工程技术、高性能体系结构等;井 靖 女,博士生,主要研究方向为软件代码反编译等;李继中 男,博士生,主要研究方向为密码算法识别等。

在一一对应的映射关系，从而能够反映出指令的数据类型语义。表 1 给出 C 语言基本数据类型与 ARM 编译的类型映射表。

(2)寄存器分配编译特征

x86 体系结构中程序计数器 PC 不出现在指令中，返回地址存储于堆栈中，其控制流语义是隐表达的。ARM 体系结构中程序编译时寄存器的分配具有自身的特点，理论上 R0－R15 都可用作通用寄存器存储中间数值。但实际代码中，为了尽可能减少溢出变量(spilled variables)，一些寄存器是具有一定用途的。ARM 编译后的代码中，寄存器分配一般遵循其 ABI 中的 ATPCS[3,4] (ARM-Thumb Procedure Call Standard)，例如 LR 分配作为保存返回地址，SP 作为过程栈指针，R0－R4 用于传递过程前 4 个参数。

(3)函数调用编译特征

ARM 微处理器中不存在"call"和"ret"指令，函数调用和调用返回是通过具有特定操作语义的指令或指令序列完成的。函数调用遵循 4 寄存器规则(four-register rule)，即编译器通过寄存器 R0－R3 传递函数的前 4 个参数。调用过程和被调用该过程从栈中访问余下的参数(如果存在的话)，函数返回值则通过寄存器 R0/R1 传递。函数调用和返回的编译特征示例如图 1 所示。函数 f2 通过指令 BL 实现调用 f1 时要求的语义操作：LR[31:1] ＝返回地址、PC 调用过程的地址，LDMFD 指令实现函数 f1 返回到 next 处继续执行，参数的传递方式如图中阴影部分所示。

```
Source Code:                    Assembly Code:
int f1(int a,int b,int c,       函数 f1:
int d,int e)                      MOV R12,SP
{                                 STMFD SP!,{R11,R12,LR,PC};保存
  ……,                          next 到 LR
   return result;//re-            ……
turn                              LDR R3,[R11,#K2]
}                                 MOV R0,R3;返回值传递
void f2( )                        LDMFD SP,{R11,SP,PC};返回到 next
{                               函数 f2:
  ……                              ……
   f1(1,2,3,4,5);                 MOV R3,#5
   …… //next                     STR R3,[SP,#K1]
}                                 MOV R0,#1;参数
                                  MOV R1,#2;参数
                                  MOV R2,#3;参数
                                  MOV R3,#4;参数
                                BL sub_8030;f1(1,2,3,4,5)
                                  ……
```

注：K_1 和 K_2 是栈空间存储单元的整数偏移值

图 1　函数调用编译特征示例

此外，ARM 指令集中不存在除法指令，编译器会将除法运算编译为库函数调用。ARM 微处理器本身不支持硬件浮点运算，编译器会将浮点运算转换为使用定点整数模拟浮点运算子函数调用。ARM 体系结构下的这些程序编译后的代码特征影响二进制嵌入式代码的语义属性分析。

3　ARM 微处理器二进制嵌入式代码解析模型

嵌入式代码逆向分析从机器码出发，首先遇到的问题就是二进制嵌入式代码的反汇编，同时分析汇编表示中的代码语义属性，以便服务于更高层次上的分析，实现更高级别的代码语义提升。

因此借鉴已有的二进制代码解析思想，本文构建了一种基于代码语义属性分析的二进制代码解析模型，利用已有的反汇编工具可以快速有效地实现 ARM 二进制嵌入式代码的解析，从而完成机器级到汇编级的代码语义提升。ARM 二进制嵌入式代码解析模型如图 2 所示。二进制解析由反汇编和汇编代码语义属性分析组成，通过汇编代码的语义属性分析可以无缝连接到反汇编器，分析提取相关的代码语义信息，从而完成二进制解析，并支持后续的代码逆向分析。

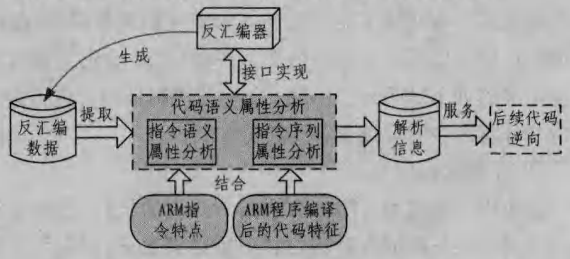


图 2　ARM 二进制嵌入式代码解析模型

在 ARM 二进制嵌入式代码解析模型中，反汇编器首先识别和加载二进制文件，并完成机器指令到对应的汇编指令的映射。这一阶段实现代码中指令和数据的分离、过程(函数)的划分等。二进制文件反汇编后，嵌入式代码解析需要分析代码语义属性，主要包括指令语义属性和指令序列属性。这两类语义属性的分析依赖于 ARM 指令的组成格式和指令特点。此外，源程序编译为 ARM 目标码后的一些共有特征也是代码语义属性分析需要考虑的。结合 ARM 指令特点和编译后的代码特征，通过反汇编器的扩展接口能够生成解析信息，并服务后续的代码解析工作。

4　ARM 微处理器嵌入式代码语义属性分析

为了进一步变换代码以提升代码语义，需要分析嵌入式代码在汇编级的语义属性，并抽取这些语义属性作为下一步分析的基础，这就是二进制嵌入式代码解析中的代码语义属性分析。本节结合 ARM 体系结构的指令特点和程序编译后的代码特征阐述汇编级别上的代码语义属性分析方法，从不同粒度级别上抽取代码语义属性，在呈现汇编级语义的同时，为下一步的代码分析提供准确充分的数据源。

4.1　指令属性分析

指令一般由操作符、条件词、操作数 3 部分组成。操作符对应于机器指令码中的操作码，条件词和操作数对应于操作码执行操作的相关数字序列。ARM 微处理器支持 32 位的 ARM 指令集和 16 位的 Thumb 指令集，Thumb 指令集在语义功能上是 ARM 指令集的一个压缩子集。这里以 ARM 指令集为例，阐述指令分析的方法也适用于 Thumb 指令集。

ARM 指令的一般格式可以表示为如下形式：

〈opcode〉{condition}{S}{SB/SH/B/H/D}

〈operand0〉{!},〈operand1〉{,〈operand2〉}

指令格式中〈〉符号内的项是必需的，{}符号内的项是可选的，/ 符号表示选其中之一。其中，opcode 表示指令操作符部分，后缀 condition，S，SB/SH/B/H/D 及！构成了指令的条件词，operand0，operand1，operand2 为操作数部分。指令分析就是将指令的各组成部分解析出来，构成指令属性[5]。

(1)操作符属性

操作符在指令中是固定不变的，唯一对应机器指令码中的操作码。反汇编能够确定指令操作符和机器指令操作码之

间的对应关系，将机器指令操作码转换为汇编语言指令中的以字符串表示的操作符，因此本质上不同的操作符就是一个能够唯一区分的数字序列编码。

反汇编器在反汇编的同时，其内部表示必然具有操作符及其内部表示之间的对应关系，以方便内部处理和文本结果的输出。反汇编器对操作符的内部表示屏蔽了表示操作码的间隔的位序列，采用唯一标识该处理器指令操作的编码表示操作符。基于本文提出的二进制嵌入式代码解析模型，操作符属性可以延用反汇编器的内部编码值(这个编码值在可扩展反汇编器如 IDA Pro、OllyDbg 等中是可提取的)，这样指令中的操作符属性就可以很容易解析出来。

(2)条件词属性

指令中一些附加后缀也影响指令语义的执行，这些后缀称为条件词。条件词本身不归属于操作符部分，但它们与机器码指令中的特定位有一一对应的关系，是指令语义必不可少的组成部分。ARM 指令中的条件词具有特定的含义：条件词“condition”指示当前指令是有条件执行的，反映到语义上就是根据状态寄存器 CPSR 中条件标志位是否满足而选择执行；条件词“S”表示指令是否影响更新状态寄存器 CPSR 中的相应标志位；条件词“SB/SH/B/H/D”表示指令所涉及的存储器数据类型；条件词“!”使用在含有地址表达式的指令中，紧跟在地址表达式后面，指示指令执行后是否更新基址寄存器。这些条件词指示了指令语义的副作用(side effect)。

条件词在指令中通常表现为固定的字符(串)或机器码指令中固定的连续若干位，于是条件词属性的解析等价于简单的串匹配或位匹配问题。ARM 指令条件码总是位于指令的最高 4 位[31:28]，Thumb 指令只有条件分支指令 Bxx 包含条件码并且位于[11:8]，ARM 指令和 Thumb 指令的位匹配掩码 condition _ mask 分别为 0xF0000000 和 0x0F00Bcondition的解析使用如下的方法，其中 Match()为条件码匹配函数，ins_data 为指令的机器码：

```
Match(ins_data & condition_mask,4bit_condcode);
```

条件词“S”、“SB/SH/B/H/D”及“!”在指令汇编表示都采用固定字符(串)。同样，反汇编器采用 IDA Pro，其解析方法如下，其中 generate_disasm_line( )函数为 IDA Pro SDK 接口函数[5,6]，用于获取指令的文本字符串：

```
generate_disasm_line(ins_addr,buf,sizeof(buf)-1);//buf 存放指令文本的空间
FindPostfix(buf,postfix);//postfix 为后缀字符
```

(3)操作数属性

操作数是指令中操作符操作的对象，是蕴含低级指令语义的关键因素。操作数属性包含操作数长度、操作数寻址方式、操作数对象 3 个部分。操作数长度表达的是高级语言中的类型信息，操作数对象表达的是操作数内容信息，而操作数寻址方式表达的是如何取得参与运算的内容。操作数寻址方式和操作数对象共同决定了指令运行时信息，这 3 个部分共同组成了操作数属性的语义。

·操作数长度解析

一般情况下，操作数长度就是处理器的寄存器宽度。但对于寄存器和内存交换数据的指令，存在存储器类型操作数，其操作数的长度与类型敏感指令相关。在 ARM 微处理器体系结构下，LDR/STR 指令中的条件词"SB/SH/B/H/D"影响操作数长度。操作数长度一般为 dt_byte(8 位)，dt_hword(16 位)，dt_word(32 位)及 dt_dword(64 位)。ARM 体系结构下，即使内存和寄存器传送的是 8 位或者 16 位数据，地址总线和数据总线也总是以 32 位宽度工作，也就是说操作数寻址方式在语义上与操作数长度无关。操作数长度和操作数寻址方式从不同的方面描述了操作数属性的语义。

·操作数寻址方式分析

ARM 体系结构下操作数寻址方式比较复杂。ARM 参考手册中依据 ARM 指令类型将操作数寻址方式划分为 5 类[7]，每一类又可以有多种格式的表示，例如数据处理操作数(Data-processing operands)类寻址方式就有 11 种格式。综合研究分析这 5 类寻址方式中出现的操作数寻址格式，结合通用二进制嵌入式代码解析模型反汇编采用的通用寻址方式，本文扩展描述了 ARM 操作数寻址方式类别。

表 2 给出了操作数解析中 ARM 寻址方式的扩展描述。其中 o_mem 类型操作数所在的指令常常表现为一条 LDR 伪指令，表示将一个常数或者地址传递到寄存器中，而这个常数或者地址位于数据缓冲区中。在某些特定值的情况下，表 2 中操作数寻址的扩展分类存在重叠，例如[Rn，+/－Rm]与[Rn，+/－Rm，LSL，#0]、Rm 与 Rm，LSL，#0，但上述特定值表示的操作数语义是等价的，故可以将其归入任何一类，不会影响到操作数的解析。

表 2　ARM 操作数解析中寻址方式扩展描述

编码	寻址方式	操作数表示	含义
1	o_reg	Rm	寄存器
2	o_mem	[address] 或者 label	直接存储器寻址
3	o_imm	#imm	立即数寻址
4	o_addr	address	相对 PC 的寻址
5	o_shift_imm	Rm，shift #imm	移位位数为立即数
6	o_shift_reg	Rm，shift Rs	移位位数为寄存器
7	o_RRX	Rm，RRX	带扩展循环右移
8	o_displ0	[Rn]	立即数偏移为 0
9	o_displ_imm	[Rn，#imm] / [Rn，#imm]! / [Rn]，#imm	立即数偏移
10	o_regoff	[Rn，+/－Rm] / [Rn，+/－Rm]! / [Rn]，+/－Rm	寄存器偏移
11	o_scalregoff	[Rn，+/－Rm，shift #shift_imm] [Rn，+/－Rm，shift #shift_imm]! [Rn]，+/－Rm，shift #shift_imm	寄存器移位偏移
12	o_RRXregoff	[Rn，+/－Rm，RRX][Rn，+/－Rm，RRX]! [Rn]，+/－Rm，RRX	RRX 移位偏移
13	o_reglist	{reg，reg，…}	多寄存器寻址

·操作数对象分析

操作数对象一般可以分为寄存器、立即数、存储器单元，其中存储器单元一般是由寄存器、立即数以及有限的几种运算组合构成存储单元表达式。ARM 体系结构下，由于桶形移位器的使用，操作数寻址中出现寄存器移位类型，表示参与运算的操作数为经过移位运算之后的内容。但其本身并不改变参与移位的寄存器的内容，并且存储器类型操作数中的变址部分也可以是寄存器移位类型，这就使操作数对象的分析和提取变得更加困难。

立即数和直接存储器类型，其数值有两种情况：一是直接存储于指令中；二是部分存储于指令中。但反汇编后，能体现

在汇编指令中。比如指令“MOV R0，#0x3FC”中，第二个操作数“#0x3FC”为立即数类型，其在对应的机器码中只在低 8 位存储 0xff，由反汇编器循环左移 2 位恢复，因此这两种类型的操作数对象本质上就是数值。

寄存器是一个命名的处理器存储空间，处理器对支持的每个寄存器都具有固定的唯一编码。基于这种思想，操作数属性分析中，定义映射函数 f1：R→Z，其中 R 为 ARM 微处理器的寄存器集合，Z 为整数集。这样每个寄存器语义上等价于一个整数值。经过大量的测试分析，反汇编器内部数据表示中维护着一个整数编号唯一标识汇编指令中的寄存器，寄存器操作数对象映射函数的建立可以基于二进制嵌入式代码解析模型中采用的反汇编器的内部表示。

立即数和寄存器操作数属于简单操作数对象。而操作数类型为存储器单元或者寄存器移位类型时，操作数对象是寄存器、数值、运算符的算术运算表达式，称之为复杂操作数对象。这类操作数对象需要分析操作数表达式的各组成部分以及存储地址计算模式。这里的存储器地址计算模式针对的是 LDR/STR 指令中 operand1 的变址方式和 LDM/STM 指令中多寄存器的传送模式。ARM 体系结构下存储器地址计算模式影响存储地址的计算，也影响指令的执行动作，比如基址寄存器的更新。因此复杂操作数对象的分析需考虑这两类指令中的存储器地址计算模式。

4.2 指令序列属性分析

指令属性反映的是指令内部的语义信息，对于一段嵌入式代码来讲，单条指令通常没有多大价值。代码的语义一般是以指令序列体现的，并且某些指令序列与体系结构相关。ARM 体系结构下，过程调用开始和结束时典型的指令序列如图 3 所示，可以发现这些指令序列具有如下特点：

- 这些指令序列是连续且顺序执行的，其中必定不会出现分支跳转；
- 指令序列共同完成某项功能，比如图 3(a)最主要就是保存 SP 和 LR、分配栈空间，(b)指令序列主要是恢复 SP 和执行“PC=LR”、回收栈空间。

```
MOV R12,SP
STMFD SP!,{R11,R12,…,LR,PC}
SUB R11,R12,#4
SUB SP,SP,#K1
```

(a)过程开始处指令序列

```
……
SUB SP,R11,#K2
LDMFD SP,{R! 1,…,SP,PC}
```

(b)过程结束处指令序列

注：K1 和 K2 是过程栈帧空间分配与回收的常数

图 3 过程调用典型的指令序列示例

代码语义属性分析中需要分析一定级别的指令序列间的关系，才能在后续的高层次的分析中进一步分析程序语义，实现代码语义提升。从程序代码层级结构分析，可以看出代码逆向分析中主要关注两类指令序列：一类是组成基本块的指令序列，一类是组成过程的指令序列。这里阐述的指令序列针对的就是这两类指令序列，即指令序列间的语义属性包含反映基本块间关系的过程内控制流图和反映过程间关系的过程调用图[6]。

(1)控制流图

控制流图呈现的是基本块级粒度的指令序列间的关系，一般来说控制流图与控制转移指令(Control Transfer Instruction)有关，但 ARM 体系结构下控制流图分析具有自身的一些特点。

ARM 微处理器中非 CTI 类指令可以附带 condition 条件词，当 condition 不为"AL"时，表示当前指令有选择执行，本质上是一种双分支选择结构。非 CTI 类指令的选择执行特性虽然“不改变”程序的执行流(下一条指令总会得到执行)，但会影响代码中数据的流向。在语义上，反映到高级语言上就是 if-then 结构，即该指令也引起控制流的改变。

构建 ARM 中控制流图需要考虑附加条件词 condition 非 CTI 指令的语义，附加条件词 condition 非 CTI 指令 Ins 可以分为两部分：条件 cond 和执行操作 operation。根据这种指令语义等价变换思想，控制流图的分析可以统一于基于 CTI 指令目标地址。

(2)过程调用图

过程调用图呈现的是过程级粒度的指令序列间的关系。一般来说过程调用图分析是基于过程的划分和调用指令的识别，过程调用图的构建依赖于识别调用指令及其目标地址。

ARM 指令系统中，不存在"call"和"re""指令，ARM 汇编代码中使用具有"PC=procedure address"和"PC= LR"语义的指令(序列)作为过程调用与返回的特征。编译器由于时间和空间因素会采取优化方式编译源程序，ARM 二进制代码使用了大量的桩函数(thunk function)来实现外部动态链接库函数的调用。桩函数的函数体中只包含一条跳转到某函数开始的指令，函数真正的功能执行是由跳转到的函数来完成的。

过程调用图的构建需要分析哪些指令(序列)产生"call"调用语义。通过研究大量编译后的程序代码发现，如下指令会产生"call"调用操作的语义：

1)跳转类指令：BL proc_addr；proc_addr 为过程体起始地址。

BLX proc_addr

2)存储/加载类指令：LDR PC，=__imp__procname；__imp__procname 函数标号。

LDR PC，proc_addr

3)算术类指令：MOV PC，proc_addr_imm；proc_addr_imm 表示过程地址立即数值。

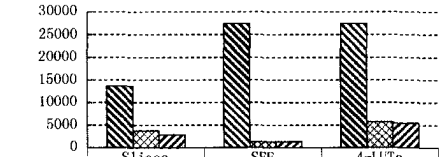
由上面的分析可知，产生"call"调用语义的指令(序列)特征：跳转类指令其跳转的目标地址必定是某函数起始地址；非跳转类指令目的操作数为寄存器 PC，源操作数为被调用函数的标号或地址立即数。过程调用图是过程间关系的抽象表示，是代码语义属性的一部分。构建过程调用图的基本思想是首先划分代码中的过程，生成过程集合，然后扫描过程中每条指令，根据过程调用特征构建过程调用图的边。

结束语 本文讨论的是基于 ARM 体系结构的二进制嵌入式代码解析中的代码语义属性分析问题。构建了基于 ARM 微处理器的二进制嵌入式代码解析模型，并结合 ARM 微处理器体系结构的特征，从指令属性和指令序列间的关系两方面来分析代码语义属性。通过该方法的实现，不仅能在语义解析过程中保证代码变换的等价性，同时也极大地提高了后续代码解析的准确性和可读性。对于二进制嵌入式代码解析模型中采用不同的反汇编器，本文的代码语义属性分析

(下转第 292 页)

的 614 个时钟周期相比都有了很大的改进。

使用 Verilog 实现本文前面设计的 H. 264 去块滤波器，在 Mentor Graphics 公司提供的 ModelSim SE PLUS 6. 1f 上进行 RTL 级模拟仿真，然后在 Xilinx 公司提供的 ISE 9. 1 平台上进行综合，配置选择 Virtex2P 系列的 XC2VP30，开发包为 FF896，速度选择 6 级，得到其资源对 FPGA 资源的使用数据。与改进前(见图 9 中 DF_top0)相比，使用了更少的 FPGA 资源，节约了约 25%的 FPGA 硬件资源单元 Slices。综合表明，过滤处理的频率最大为 72. 396MHz，能够满足高清视频的要求。



	Slices	SFF	4-LUTs
xc2vp30	13696	27392	27392
DF_top0	3771	1322	5680
DF_top1	2827	1320	5430

图 9　去块滤波器硬件资源使用

最后将生成网表在 XC2VP30 FPGA VIRTEX-II PRO 开发板上配置布线。将输入输出设置在最坏 PVT 情况下，可以以 72. 396MHz 的频率配置布线。在处理 CIF 视频文件时速度可达 689frame/s，平均每帧花费时钟周期 105074，每一宏块花费的时钟周期为 265，与理论 252 相差 13 个时钟周期，但是与文献[6,7]相比较还是有很大的改进，速度得到提高。处理 1920×1080 的高清视频速度可达 33. 5frame/s，处理 1080 HD@30 frame/s 图像大概需要 64. 872MHz 即可，能够满足高清视频的需要。

结束语　根据 H. 264 视频编码标准，本文通过改进去块滤波的基本顺序，提出了一种流水线的过滤结构，从而减少了过滤处理过程中的中间数据，降低了处理过程中需要的寄存器资源，减少了 FPGA 的硬件资源。对过滤计算进行了优化，提高了过滤速度。仿真和综合表明，本文设计的 H. 264 去块滤波器实现了设计目标。但是对于整个 H. 264 编码器的优化而言，研究工作还需要继续。接下来的工作，将围绕 H. 264 视频编码标准的帧内预测、帧间预测的运动估计和预测补偿等其他数据量大、计算复杂的模块进行优化和设计。最终使用 Xilinx 公司的 VIRTEX-II 能够同时运行 H. 264 的几个计算量大的模块，使用可重构技术，实现 H. 264 解码器。

参 考 文 献

[1] Joint Video Team(JVT) of ITU-T VCEG and ISO/IEC MPEG. Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification[S]. ITU-T Rec. H. 264 and ISO/IEC 14496-10 AVC. May 2003

[2] Richardson I E G. H. 264 and MPEG-4 Video Compression[M]. Wiley & Sons, ed. England, 2003

[3] Wiegand T, Sullivan G J, Bjøntegaard G, et al. Overview of the H. 264/AVC Video Coding Standard[J]. IEEE Trans. on Circuits and Systems for Video Technology, 2003, 13(7)

[4] Keith J. Video Demystified(4th edition)[M]. Eagle R, ed. VA: LLH Technology Publishing, 2004

[5] 林亭安. 应用于数位电视之视讯双标准解码器设计与实现[D]. 台湾：国立交通大学电子工程系，2005

[6] Huan G Y W, Chen N T W, Hsieh B Y, et al. Architecture design for deblocking filter in H. 264/JVT/AVC [C]// IEEE Int'l Conf on Multimedia and Expo. IEEE, 2003: 693-696

[7] Shen G B, Gao W, Wu D. An implemented architecture of deblocking filter for H. 264/AVC [C]// IEEE Int'l Conf on Image Processing. IEEE, 2004: 665-668

(上接第 280 页)

[5] Chanda I D B, Chattopadhyay B. Enhancing effective depth-of-field by image fusion using mathematical morphology [J]. Image and Vision Computing, 2006, 24(12): 1278-1287

[6] Tao G Q, Li D P, Lu G H. On image fusion based on different fusion rules of wavelet transform[J]. Acta Photonica Sinica, 2004, 33(2): 222-227

[7] Comaniciu D, Meer P. Mean Shift: A Robust Approach Toward Feature Space Analysis[J]. IEEE Transaction on Pattern Analysis and Machine Intelligence, 2002, 24(5): 603-619

[8] 马剑英，张晓娜. 基于免疫遗传算法的图像多阈值分割[J]. 微计算机信息，2007(1-3): 309-311

[9] 宋翠家，龙建忠，罗代升. 基于遗传算法的模糊熵多阈值图像分割[J]. 仪器仪表学报，2004, 25(4): 572-573

[9] 万辉. 一种基于最小二乘支持向量机的图像法[J]. 重庆理工大学学报：自然科学版，2011, 25(6): 53-57

(上接第 287 页)

方法同样适用。

参 考 文 献

[1] Chikofsky E J, Cross J H. Reverse Engineering and Design Recovery: A Taxonomy[J]. IEEE Software, 1990, 7(1): 13-17

[2] 张绮文，王廷广. ARM9 嵌入式应用开发[M]. 北京：电子工业出版社，2009

[3] Litmited A R M. ARM Procedure Call Standard for the ARM® Architecture[EB/OL]. http://infocenter. arm. com/, 2008

[4] Steve Micallef. IDA Plug-in Writing In C/C++[EB/OL]. http://www. binarypool. com/ idapluginwriting/, 2009

[5] Eagle C. The IDA Pro Book[M]. San Francisco: No Starch Press Inc., 2008

[6] 陈龙，武成岗，谢海斌，等. 二进制翻译中解析多目标分支语句的图匹配方法[J]. 计算机研究与发展，2008, 45(10): 1789-1798

[7] ARM Litmited. ARM Architecture Reference Maual[EB/OL]. http://infocenter. arm. com/, 2005