文章编号:1001 - 9081(2006)05 - 1237 - 04

# 动态可重构环境下循环计算的位宽优化

王 伟,李仁发,吴 强
(湖南大学 计算机与通信学院,湖南 长沙 410082)
(hnuwangwei@yahoo.com.cn)

**摘 要**:动态可重构技术允许根据计算的运行时情况对硬件处理单元进行重构,使其位宽适合计算的需要。而且,对代表计算密集型任务的循环计算进行位宽的动态优化可达到提高处理性能,减少所消耗的芯片资源和功耗的目的。文中构造了一个处理框架对循环计算的位宽进行动态的优化,包括对循环计算的位宽变化情况进行理论和运行时的分析,以及构造 1 个位宽管理算法选择重构的时机和对配置文件进行调度。通过对实验结果的分析,证明了该方案具有较好的性能。

**关键词**:动态可重构; 循环计算; 位宽; 优化
**中图分类号**: TP332 **文献标识码**:A

# Bitwidth optimization for loop computations on dynamical reconfigurable architectures

WANG Wei, LI Ren-fa, WU Qiang
(*School of Computer and Communications, Hunan University, Changsha Hunan* 410082, *China*)

**Abstract:** Dynamical reconfigurability had the potential to change the processing unit's bitwidth according to the runtime's condition of the computation. Further more, the dynamic bitwidth optimization for looping computations, which are the one of the most computing intensive tasks, might achieve better performance, small size and less power. In this paper, a management framework to optimize the bitwidth of the loop computations dynamically was developed, including the theoretical and runtime analysis of the loop computations' bitwidth, and developing a bitwidth management algorithm to select the schedule of reconfiguration and scheduling the configuration files. The experiment result indicates that our scheme has better performance.

**Key words:** dynamic reconfigurability; loop computation; bitwidth; optimization

## 0 引言

动态可重构系统是指系统的硬件结构在运行时可随计算任务的要求动态的改变,系统所具有的动态可重构特征为任务的加速提供了潜在的机会。在大多数主从结构的可重构系统中,作为从设备的可重构硬件一般用来处理计算密集型任务,主要表现为循环计算的形式,因此,对循环计算的优化可明显提高整个任务的性能。在微处理器及 ASIC 计算领域针对循环计算已经提出了大量的优化方法,包括循环并行技术、软件流水线技术以及利用 Cache 技术开发指令和数据的局部性等,这些传统的优化方法都可移植到可重构环境中。而且,在动态可重构环境下,可根据其特性引入有别于微处理器和 ASIC 环境下的优化方法,动态位宽优化即属于这一类。在循环计算过程中,根据计算任务的要求,对处理单元的位宽进行动态调整。通过对硬件结构的动态重构,使处理单元的位宽尽可能匹配计算任务的要求,而达到提高性能,减少芯片所需逻辑资源和功耗的目的。

近年来,针对可重构领域的应用映射问题研究比较广泛,定制可重构硬件使其适合计算任务的要求被认为是这类结构最重要的优势之一。使用可重构技术进行计算的位宽优化可分为两类:

*静态* 处理单元的位宽在编译时设定,可做到与在微处理器上使用的标准位宽(如 8 位、16 位、32 位等)不同,如文献[1~3]。但这类方法没有利用可重构硬件所具有的随计算任务的要求而动态调整位宽的特性。静态方法在编译时所确定的最大可能位宽在计算处理过程中仍然会带来不必要的资源开销。

*动态* 其在运行时可根据算法的要求动态的改变计算单元的位宽,如文献[4]。在文献[4]中提出了一个动态位宽管理框架,先通过对循环计算进行理论分析及运行时分析,得到计算过程中位宽的变化情况,然后,提出了基于遍历搜索的动态位宽管理算法,得到一个循环计算过程中配置文件的调度序列,使之在循环计算过程中使用这个调度序列执行开销最小。作者由于其模拟平台的限制,未能使用动态不放重构技术。

## 1 循环计算的位宽优化模型

模型参数:

$C$:可用的配置文件集合。

$C_{ij}$:位宽为 $i$ 编号为 $j$ 的配置文件,$C_{ij} \in C$。实现同一位宽的运算可有多种不同的实现算法和方式,对应有不同的配置文件,它们在执行时间、所占用 FPGA 资源和功耗方面也有所不同。

$t_{C_{ij}}$:在配置文件 $C_{ij}$ 下执行一次循环所需的时间。

$L_{C_{ij}}$:配置文件 $C_{ij}$ 所占用的 FPGA 的逻辑门数。

$A_{C_{ij}}$:在配置文件 $C_{ij}$ 下执行一次循环所需的功耗。

$Q_i$:位宽为 $i$ 时运行的循环次数。

$R_{i-1i}$:使用位宽为 $i$ 的配置文件替换位宽为 $i-1$ 的配置文

**收稿日期**:2005 - 11 - 02;**修订日期**:2006 - 01 - 13
**基金项目**:国家发改委重大项目(计高技 2034);湖南省科学技术厅制造业信息化示范工程项目(HNMIE-A-026)
**作者简介**:王伟(1981 - ),男,硕士研究生,主要研究方向:可重构计算、嵌入式计算系统; 李仁发(1957 - ),男,教授,博士生导师,主要研究方向:嵌入式计算系统、数字媒体、分布式计算、网络; 吴强(1974 - ),男,博士,主要研究方向:电子设计自动化.

件的重构时间开销。

$P_{i-1i}$:使用位宽为 $i$ 的配置文件替换位宽为 $i-1$ 的配置文件的重构功耗开销。

评价指标:

$E$:总的时间开销,包括循环执行时间和重构时间。

$S$:平均每个循环所消耗的芯片逻辑门数。

$Y$:总的功耗开销。

$$E = \sum_{j=v}^{w}\left[(Q_{j+1} - Q_j) \times t_{c_{jk}} + R_{j-1j}\right]$$

$$S = \left\{\sum_{j=v}^{w}\left[(Q_{j+1} - Q_j) \times \min_{k}(L_{c_{ij}})\right]\right\} / \sum_{j=v}^{w} Q_j$$

$$Y = \sum_{j=v}^{w}\left[(Q_{j+1} - Q_j) \times A_{c_{jk}} + P_{j-1j}\right]$$

从不同的角度出发,我们可对循环计算的位宽执行不同的优化。从执行时间的角度出发,优化的目的是使 $E$ 最小。类似地,针对资源或功耗敏感型应用,优化的目标是使 $S$ 或 $Y$ 最小。或综合考虑几个方面的因素达到一个综合的性能。本文由于平台和工具的限制,仅关注总的执行时间开销的优化。

## 2 优化算法的实现

由所给出的循环计算,经理论分析和运行时分析得到计算的位宽变化曲线,找到位宽变化点(即重构点)的集合。由于重构开销的存在,频繁的重构可能会完全抵消优化所带来的性能提高或使性能更低,因此,需要在操作单元的位宽精度和重构开销之间做出折中。以重构点集,配置文件集和重构开销为输入,经动态位宽管理算法处理,得到最优的配置文件调度序列。最后,在循环执行过程中按序将配置文件下载到 FPGA 上进行重构。框架的总体结构如图 1 所示。

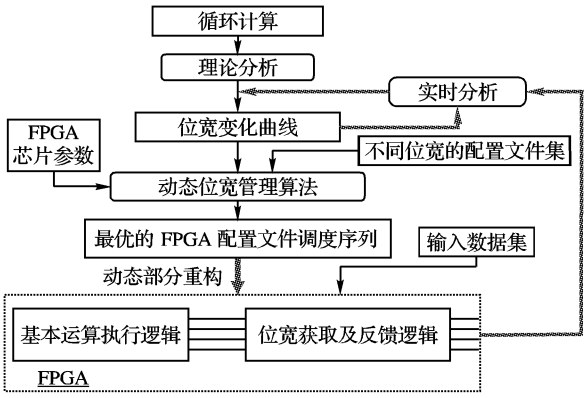


图 1 系统框架的总体结构

### 2.1 循环计算的位宽变化情况分析

在循环计算中,计算结果的位宽不仅与计算的类型有关,而且会随着迭代的进行而有规律的变化。本节将对循环计算的位宽进行理论和运行时分析,得出其位宽变化曲线。参考文献[4]对位宽变化曲线的定义,对于给定的循环计算,可用序列 $< L_i, P_i >$(其中,$1 \leqslant i \leqslant u+1$,$L_{u+1} = N-1$,$N$ 为最大循环迭代数)表示位宽变化曲线。对于 $1 \leqslant i \leqslant u$,$P_i$ 是迭代 $L_i \cdots L_{i+1} - 1$ 中操作数所要求的最小位宽。

#### 2.1.1 位宽变化的理论分析

循环过程中变量的位宽变化可由迭代前变量的位宽、迭代次数和变量所执行的操作来确定。对每种类型的算术操作,计算结果的最大可能位宽都可由这 3 个参数来确定。如对于 $X = X + C$ 的循环运算,在第 $i$ 次迭代时 $X$ 的理论位宽上界 $\Pr(X) \leqslant \Pr(C) + \log(i+1)$,其中 $\Pr(C)$ 表示常量 $C$ 的位宽。对于 $X = X * C$ 型的循环运算,在第 $i$ 次迭代时,$X$ 的理论位宽上界 $\Pr(x) \leqslant i * \Pr(C)$[4],类似的,对于减或除的迭代运算,也可得出其输出的理论位宽上界。而对于循环中较复杂的递归表达式

$$X_i = c_1 * X_{j_1} + c_2 * X_{j_2} + \cdots + c_k * X_{j_k} = \sum_{l=1}^{l=k} c_l * X_{j_l}$$

在第 $i$ 次迭代时 $X_i$ 的理论位宽上界:

$$\Pr(X_i) \leqslant (i-1) * \log(C) + (i-1) * \log k + \Pr(X_1)$$

其中,$C = \max[c_1, c_2, \cdots, c_k]$[4]。对于所有由常量和变量组成的多项式迭代循环,都可经过理论分析得到其多项式结果的位宽变化曲线。这适用于大多数的信号和图像处理。

#### 2.1.2 位宽变化的运行时分析

循环计算的理论分析是以最大可能输入为基础对循环计算的位宽变化进行评价的,而在实际运行时,输入数的位宽是随机分布的。这样,每次迭代过程输入与理论估计值都会有一定偏差,而此偏差会随迭代的步进而累积。

我们在 FPGA 上实现对下面这个循环运算的动态可重构模块,对实际运行时的结果与理论分析的结果进行比较,得到如图 2 所示的理论分析和运行时的位宽变化曲线:

```
for (int i=0; i<N; i++)
  for (int j=0; j<N; j++) {
    result = result + A[i][j] * B[i][j];
  }
```

其中,A[N][N]和 B[N][N]是两个 int 型的数组,保存的数据是随机生成的 0 ~ 8bit 的整形数。我们的实时分析是通过在芯片上构造一个变量位宽的实时反馈模块,从而得到运行时变量的位宽。从图 2 可知,理论分析的位宽变化大大超前于运行时的变化,以运行时的位宽信息对理论分析得出的位宽变化曲线进行调整,以此作为对算术运算模块动态重构的依据,可进一步减少计算所消耗的芯片资源和功耗,提高处理的速度。

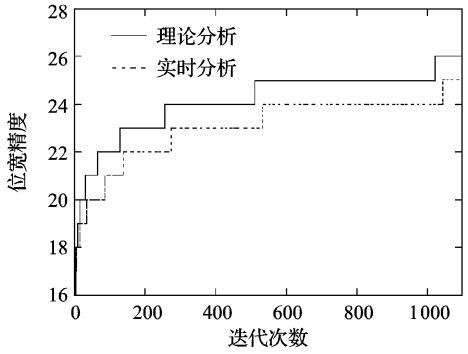


图 2 理论和实时分析的位宽变化曲线

### 2.2 运行时位宽反馈及分析

通过在 FPGA 上加入一个位宽获取及反馈逻辑,得到某次循环中变量的实际位宽。当循环过程中变量的位宽发生变化时,以当前的位宽变化曲线(第一次处理时为理论分析得出的位宽变化曲线)为基础,以位宽获取及反馈逻辑得到当前循环中变量的实时位宽,修改后续循环中位宽变化点的预测位置,即位宽变化曲线中拐点的位置。整个处理过程如图 1 中的粗连线部分所示,实时分析和位宽变化曲线之间是个反复迭代的过程,即根据获取的运行时变量位宽反复的对位宽变化曲线进行修正,使之尽可能的反映循环过程中实际的变量位宽变化情况。

### 2.3 动态位宽管理算法

问题描述:给定重构点集(由位宽变化曲线给出)$P(P_1, P_2, \cdots, P_n)$、配置文件集 $C(C_1, C_2, \cdots, C_n)$ 和重构开销矩阵 $R$(重构开销矩阵为一上半矩阵,表示从 $C_i \rightarrow C_j (i<j)$ 的重构

开销,找到一个配置文件的调度序列( $< P_{i1}, C_{j1} >$, $< P_{i2}, C_{j2} >, \cdots, < P_{im}, C_{jm} >$),($ik \leqslant jk$),使得总的开销(计算开销 + 重构开销)

$$E = \sum_{j=v}^{w} \left[ (Q_{j+1} - Q_j) \times t_{C_j} + R_{j-1j} \right]$$

最小,其中 $Q_j$ 表示重构点 $P_{i1}$ 的迭代次数,因此 $Q_{j+1} - Q_j$ 表示在 $C_j$ 上执行的循环次数,$t_{C_j}$ 表示在 $C_j$ 上执行一次循环所需时间。

调度序列需满足:

(1) $P_{i1} = P_1$:循环开始执行时需要一个配置文件对 FPGA 进行配置。

(2) $ik \leqslant jk$:配置文件的位宽需大于等于计算所要求的位宽。

(3) $i1 < i2 < \cdots < in$:不允许在同一个重构点进行多次重构。

(4) $j1 < j2 < \cdots < jn$:不允许使用同一位宽的配置文件配置多次。

(5) $C_{jm} = C_n$:在调度序列的最后一个重构点(不一定为 $P_n$)使用 $C_n$。

可知,调度序列中($P_{i2} \cdots P_{im}$) $\subseteq$ ($P_2 \cdots P_n$),($C_{j1} \cdots C_{jm-1}$) $\subseteq$ ($C_1 \cdots C_{n-1}$)。求解其中的最优解问题是一个 *NP* 完全问题,我们拟用遗传算法寻找尽可能最优的调度序列。

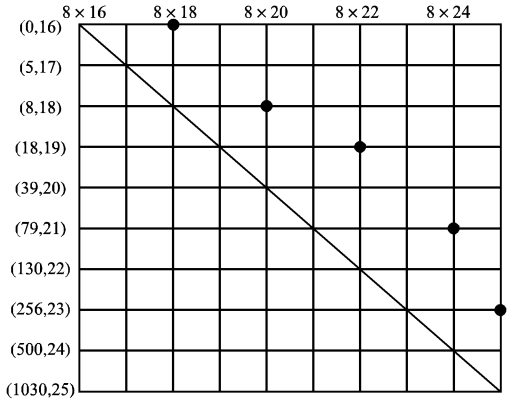


图 3　遗传编码方法

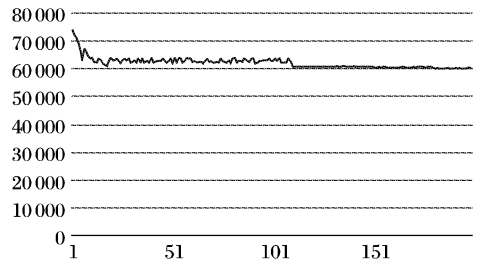


图 4　进化过程运行结果

(1) 基因编码方法:由重构点和配置文件构建一矩阵,在一重构点使用某个配置文件则选中它们的交叉点。由算法的要求,得知只有上半矩阵的部分点序列满足要求。编码时先对纵轴方向编码,后对横轴方向编码,选中的点置“1”,否则置“0”。如图 3 所示,其调度序列为( $< P_1, C_3 >$, $< P_3, C_5 >$, $< P_4, C_7 >$, $< P_6, C_9 >$, $< P_8, C_{10} >$),编码为“10110101000010101011”。

(2) 个体适应度的计算:由基因编码解码得到调度序列,由调度序列、与配置文件对应的单次循环计算开销和重构开销矩阵,计算总的开销:

$$E = \sum_{j=v}^{w} \left[ (Q_{j+1} - Q_j) \times t_{C_j} + R_{j-1j} \right]$$

(3) 算法运行参数的确定:经反复调试,筛选,确定初始群体个数为 75,终止代数为 200,交叉概率为 0.625,变异概率为 0.037。图 4 为进化过程中每代最佳个体适应度的运行结果。

## 3　实验及结果分析

整个框架使用 Java 在 Eclipse3.0 和 JDK1.5.0 上实现,其中在 FPGA 上运行的循环体的硬件实现使用 Xilinx 公司的 JBits2.8 SDK[5] 和 VirtexDS 模拟器[6]。JBits 是一个基于 Java 的开发包,其提供了 API 用来操作 Xilinx FPGA 的位流(Bitstream)文件,可访问 FPGA 上所有的可重构资源如查找表(LUT),路由连接或触发器(Flip-Flops)等,并可独个地对其进行配置。JBits 所提供的动态路由、运行时参数化核和动态部分重构 API 等使得对 FPGA 动态重构的开发较传统开发工具而言方便不少。

实验使用的循环运算为:

```
for( int i = 0; i < N1; i++) {
  for( int j = 0; j < N2; j++) {
    RSQ[j] = RSQ[j] + Xdiff[i][j] * Ydiff[i][j];
    if ( maxq < RSQ[j])
      maxq = RSQ[j];
  }
  virtxy = virtxy + maxq * Scale[i];
}
```

其中,由硬件实现:

```
RSQ[j] = RSQ[j] + Xdiff[i][j] * Ydiff[i][j];
virtxy = virtxy + maxq * Scale[i];
```

条件判断和循环控制则由软件实现。使用 JBits 进行动态可重构开发的另一个突出的优点在于软硬件都由同一语言描叙,软硬件的划分非常方便,缺点是需要访问每个 SRAM 的可配置位,开发太底层,工作量大。乘累加运算模块的逻辑结构图如图 5 所示。图中的每个模块 Register、Multiplier、Adder、WidthReback 都是动态可重构的 RTPCore,对其中的每个模块按位宽要求进行重构时都不会对其他模块的运行产生影响。重构之后的位流文件下载到 VirtexDS 模拟器上运行,Xdiff 和

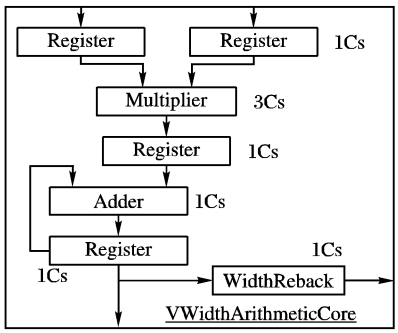


图 5　FPGA 上处理模块逻辑图

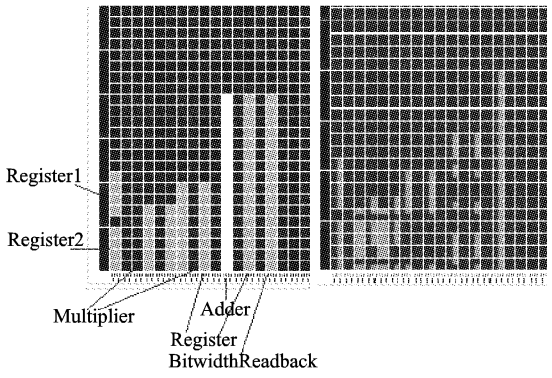


图 6　位流文件的各个模块分布图和运行时状态图

Ydiff 为随机产生的 8 位的整形数组。图 6 为各个模块在 VirtexDS 上的分布图及运行时的状态图。

Xdiff 和 Ydiff 以及 Scale 数组都是由 0 到 $2^8$ 的随机整型数组成。由内层嵌套循环生成的 $RSQ[j]$ 的理论和运行时位宽变化曲线如图 2 所示。外层循环对乘法器和加法器及相应寄存器根据位宽管理算法得出的调度序列分别进行重构。在 Pentium4 3.0G,512 内存的机器上运行 VirtexDS 对实际的 FPGA 进行模拟,执行外层循环 1 200 次并对各种情况进行模拟得出实验数据如表 1 和图 7 所示(因为 VirtexDS 还没有时间分析工具,所得数据为直接从程序中使用时钟获取 API 所得,结果并非为在实际 FPGA 上的执行时间,为模拟时间)。我们使用了 6 种不同的计算方案计算循环执行总的开销,以纵向比较我们所提出的方法对循环计算效率的改进。并以改进的百分比与文献[4]中的方法比较,说明我们所使用的动态部分重构技术和实时位宽反馈硬件模块所获得的性能改进。

方案 1:模拟在微处理器上的执行情况,使用 8 × 32 位宽的配置文件 $C_6$ 执行所有全部循环。调度:$< 1, C_6 >$

方案 2:根据理论分析,可知使用 8 × 28 的配置文件 $C_5$ 可完全满足要求。调度:$< 1, C_5 >$

方案 3:根据理论分析得出的位宽变化曲线,使用全部重构的方式进行配置,即模拟文献[4] 的 Greedy 算法。调度:$< 1, C_2 >, < 2, C_3 >, < 32, C_4 >, < 512, C_5 >$

方案 4:使用动态部分重构方法执行方案 3。

方案 5:使用理论分析得出的位宽曲线,使用我们设计的动态位宽管理算法和动态部分重构方法。调度:$< 1, C_4 >, < 512, C_5 >$。

方案 6:与方案 5 相比,加进了实时分析模块对位宽曲线进行调整。调度:$< 1, C_3 >, < 79, C_4 >$。

**表 1 使用 VirtexDS 对 Xilinx XCV800 模拟得到的实验数据**

| 配置 $C_i$ | 位宽 $\Pr(C_i)$ | 计算时间 $t_{C_j}$/ms | 全重构时间 $R_{oi}$/ms | 部分重构时间 $R_{ij}$/ms; $j = i + 1$ |
|---|---|---|---|---|
| $C_1$ | 8 × 8 | 16 | 516 | 516 |
| $C_2$ | 8 × 16 | 23 | 1 058 | 315 |
| $C_3$ | 8 × 20 | 35 | 1 365 | 437 |
| $C_4$ | 8 × 24 | 49 | 1 640 | 288 |
| $C_5$ | 8 × 28 | 67 | 1 799 | 297 |
| $C_6$ | 8 × 32 | 76 | 1 985 | 438 |

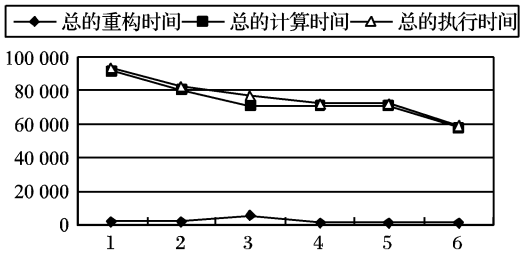

图 7 各种方法所消耗执行时间的比较

从表 1 和图 7 得到的实验数据说明:

(1) 最终的优化算法(方案 6)所消耗的总的执行时间仅为在传统计算模式(方案 1)下算法的 62.69%,改善了循环计算的执行效率。

(2) 通过使用动态部分重构方法(方案 4)与文献[4]中使用的动态全部重构(方案 3)相比,总的重构时间仅为后者的 22.81%,总的执行时间为后者的 94.09%,使用动态部分重构方法,较大地减少了重构时间,并减少了重构开销对总的执行开销的影响。

(3) 加进我们设计的动态位宽管理算法(方案 5),与未使用(方案 4)时相比,总的执行开销只有较小的减少,动态位宽管理算法主要用来选择重构的时机,使重构开销与计算开销之间达到较好的折中。在使用动态部分重构方法后,重构开销对计算开销的影响已比较小,故仅采用动态位宽管理算法对总的执行开销的改进效果不明显。不过我们通过采用基于遗传算法的位宽管理算法,与文献[4]所使用的遍历方法相比,搜索可能的最优解所需时间大大减少。

(4) 采用实时位宽反馈硬件模块和运行时位宽分析(方案 6)与仅使用理论分析得出的位宽变化曲线(方案 5)相比,前者总的执行时间仅为后者的 81.40%。而文献[4]中的方法仅使用编译时信息表示运行时位宽,总的执行时间为未采用时的 84.24%。采用位宽实时硬件反馈模块结合本文提出的运行时分析方法可取得比文献[4]中方法相对较好的性能。

## 4 结语

本文利用动态可重构技术,构建了一个框架,通过对循环计算的位宽进行优化以提高循环计算的处理性能,减少所消耗的 FPGA 芯片资源和功耗。实验结果表明,采用部分重构技术和借助运行时位宽分析的动态位宽管理方法能提高在 FPGA 上进行循环处理的性能。而对于嵌入式系统而言,功耗是个重要的参数,本文由于模拟平台的限制,未对功耗进行分析,这是下一步研究的方向。

**参考文献:**

[1] BENEDETTI A, PERONA P. Bit-width optimization for configurable DSP's by multi-interval analysis[A]. Proc. 34th Asilomar Conference on Signals, Systems and Computers[C], 2000.256 - 262.

[2] STEPHENSON M, BABB J, AMARASINGHEA S. Bitwidth analysis with application to silicon compilation[A]. Proc. of Conference on programming language Design and implementation[C], 2000.108 - 120.

[3] KUM K, SUNG W. Combined Word - Length Optimization and High-Level Synthesis of Digital Signal Processing Systems[J]. IEEE. Trans. Computer Aided Design, 2001,20(8):45 - 54.

[4] BONDALAPTI K, PRASANNA VK. Dynamic Precision Management for Loop Computations on Recongurable Architectures[EB/OL]. http://ceng. usc. edu/ ~ prasanna/papers/bondalapatiFCCM99. pdf.

[5] GUCCIONE SA, LEVI D, SUNDARARAJAN P. JBits: A java - based interface for reconfigurable computing[A]. 2nd Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD)[C], 1999.253 - 261.

[6] MCMILLAN S, BLODGET B, GUCCIONE S. VirtexDS: A Virtex Device Simulator, In Reconfigurable Technology: FPGAs for Computing and Applications II[A], Proc SPIE 4212[C], 2000.50 - 56.