МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное образовательное
учреждение высшего образования
ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

Институт математики, механики и компьютерных наук
имени И. И. Воровича

Направление подготовки
01.03.02 — Прикладная математика
и информатика

СТАТИЧЕСКИЙ АНАЛИЗ КОДА ДЛЯ ОПРЕДЕЛЕНИЕ БИТОВОЙ ШИРИНЫ ПРОМЕЖУТОЧНЫХ ДАННЫХ

Курсовая работа

Студента 3 курса
Д. В. Каримова

Научный руководитель:
кандидат физико-математических наук / доцент кафедры
информатики и вычислительного эксперимента Д. В. Дубров

оценка (рейтинг) подпись руководителя

Ростов-на-Дону
2017

Содержание

Введение

Архитектуры современных процессоров предназначены для решения максимально общих задач. Однако, темп развития микропроцессорных устройств в последние годы несколько замедлился и уже не подчиняется закону Мура. [1] Вместе с тем, необходимость в обратной совместимости ещё больше замедляет развитие микропроцессоров. Для ускорения микропроцессоров используются конвейеры, и различные внешние ускорители - графические или ПЛИС(FPGA) [2].

Конвейер, можно реализовать на системах с реконфигурируемой архитектурой, таких как ПЛИС.

При использовании систем с рекофигурируемой архитектурой необходимо эффективного распологать информацию в памяти ПЛИС. Для выбора такого расположения существуют различные методы — профилировки или статического анализа кода. В ОРС уже реализован метод профилировки, однако он плох, т.к. он дает ответ лишь для определенного набора входных данных. Метод статического анализа позволяет дать общий ответ, для любых наборов данных.[3]

Целью этой работы является реализация данного метода и проверка его корректности.

1. Описание алгоритма

Для реализации метода вводится понятие "важности" битов в соответсвии с тем, какая информация о них нам достоверно известна и доступна. "Важность"связана с особенностью проектировки электронных схем. Так, в какой-то момент времени, в зависимости от напряжения в схеме, значение бита может быть не только "0" или "1" , но и "unknown" , это значит, что мы достоверно не можем сказать, какое значение там сейчас.

Биты, с достоверно изветным значением будут квалифицированы этим значением — "0" или "1".

Биты, информация о которых нам не известна на этапе анализа называют "unknown" и обозначают как "u".

Биты, которые нам неважны квалифицируют как "DontCare" и обозначают как "x".

1.1. Переменные и константы

В соответствии с алгоритмом, каждая новая встреченная переменная, не инициализированная каким-либо значением сразу, получает набор битов <u>, длины равной размеру данного типа. Этот размер, является входными данными для алгоритма определения битовой ширины. Например — int может быть инициализирован 32 битами. Инициализированные сразу переменные получают набор битов, которые потребовались для инициализации. Константые значения сразу инициализируются необходимым набором бит. Размер констант, и биты, используемые в них, не могут быть изменены. Пример:

Листинг 1.1. Пример выделения памяти для различных типов данных C++

```
#include <iostream>
using namespace std;

const int ConstValue = 5; //Const <101>

int main()
{
    int a; //Variable: 32 bits <u>
    int b = 5; //Variable: <101>
    system("pause");
    return 0;
}
```

1.2. Действия над данными

Для дальнейшего описания алгоритма вводятся следующие операции. Начнем с простых: сложение и вычитание.

Сложение Для сложения используются данные из таблицы. В таблице содержатся значения битов слагаемых и переноса. Перенос никогда не может быть "DontCare".

Вычитание Принцип реализации точно такой же, как и у сложения. Есть биты уменьшаемого, вычитаемого и перенос, который так же не может никогда быть равен <x>.

Умножение Если один из операндов является степенью двойки, то добавим в конец второго количество нулей, стоящих на младших позициях после единственной единицы. В противном случае:

1. Введем ta — число нулей младших разрядов в числе "A"и tb – число нулей младших разрядов в "B".
2. Введем la и lb — число нулей старших разрядов в числах "A" и "B"соответсвенно.
3. Возвращаем вектор битов <u> длины = (Длина_числа(A) + Длина_числа(B) - ta - tb - la - lb).

Опишем остальные операции:

Побитовые операции OR, AND, XOR Реализация побитовых операций схожа с реализацией операциий сложения и вычитания. Мы используем таблицы, для определения результатов даных операций.

Логические операции AND, OR: Будем проверять числа "A" и "B". Результатом операции будет один единственный бит или пара бит. При этом, True — соответвует <01>, False — <0>, Dontknow — <0u>.

Операция AND Алгоритм работы:

1. Если в "A" и "B" есть <1>, вернуть True.

2. Если в "A" или "B" есть хотя бы один бит <u>, вернуть Dontknow.

3. Вернуть False.

Операция OR Общий алгоритм:

1. Если в "A" или "B" есть <1>, вернуть True.

2. Если в "A" или "B" есть <u>, вернуть Dontknow.

3. Вернуть False.

Теперь разберёмся с целочисленным делением.

Целая часть Результатом операции "A / B" будет последовательность бит <u> длиной, равной длине числа "A".

Остаток от деления Остаток от деления так же представляется последовательностью бит <u>, однако его длина это уже min(Длина_числа(A), Длина_числа(B)).

Все остальные операции описаны в данной статье. [4]

1.3. Циклы

Описанный алгоритм требует минимального количества информационных зависимостей. В частности, встреча такой зависимости в циклах сильно замедляет его работу, т.к. вместо анализа, приходится выполнять интерпретацию кода, что не является прямой задачей данного алгоритма. Поэтому, в отличии от других работ, вместо использования разбора сложных SSA-форм было решено при встрече цикла с зависимостями все переменные циклы инициализировать векторами битов максимальной длины из <u>.

2. Реализация алгоритма.

2.1. Представление данных

Все даные представляются как члены специального класса. У этого класа есть все необходимые методы для вычисления битовой ширины. В том числе, хранится битовое представление каждого элемента и количество "важных" бит.

Были реализованы все описанные выше операции для беззнаковых целых чисел.

Рассмотрим пример реализации суммирования и умножения.

Листинг 2.1. Сложение

```
RangeParametr operator+(RangeParametr& left, RangeParametr& right
   ){
    unsigned int lenght = std::max(left.getCountBits(), right.
       getCountBits()) + 1;
    RangeParametr retValue("NameRet", lenght);
    typeBits carry = ZERO;

    //if left.countBits != right.countBits
    left.resizeBitsValue(lenght - left.getCountBits() - 1);
    right.resizeBitsValue(lenght - right.getCountBits() - 1);

    pairBits retPair;
    for (int i = lenght - 1; i > 0; i--){
        retPair = summurise(left.bitsValue[i - 1], right.
           bitsValue[i - 1], carry);
        retValue.bitsValue[i] = retPair.second;
        carry = retPair.first;
    }
    retValue.bitsValue[0] = carry;

    left.resizeBitsValue();
    right.resizeBitsValue();

    return retValue;
}
```

Листинг 2.2. Умножение1

```
RangeParametr operator*(RangeParametr& left, RangeParametr& right
   ){
    bool power2Flag;
    power2Flag = true;
    unsigned int j = 0, countOne = 0, countZeros = 0;

    RangeParametr tempConcatenate;

    while (power2Flag && j < left.bitsValue.size()){
       if (left.bitsValue[j] == ONE) {
        countOne++; countZeros++;
       };
        if (countOne > 1) power2Flag = false;
        ++j;
    }
    if (power2Flag){
        tempConcatenate.initialiseCountThisType(right.
           getCountBits() + left.getCountBits(), ZERO);
        std::copy(left.bitsValue.begin(), left.bitsValue.end(),
           tempConcatenate.bitsValue.begin() + right.getCountBits
           ());
        return tempConcatenate;
    }
    power2Flag = true;
    while (power2Flag && j < right.getCountBits()){
        if (right.bitsValue[j] == ONE) countOne++;
        if (countOne > 1) power2Flag = false;
        ++j;
    }
    if (power2Flag){
        tempConcatenate.initialiseCountThisType(right.
           getCountBits() + left.getCountBits(), ZERO);
        std::copy(right.bitsValue.begin(), right.bitsValue.end(),
            tempConcatenate.bitsValue.begin() + left.getCountBits
           ());
        return tempConcatenate;
    }
```

Листинг 2.3. Продолжение умножения

```
    //count of trailing and leading zeros in left and right
    unsigned int tleft, tright, lleft, lright;
    unsigned int  i = 0;
    countZeros = 0;
    while (left.bitsValue[i] == ZERO){
        ++i; ++countZeros;
    }
    lleft = countZeros; i = countZeros = 0;

    while (left.bitsValue[left.bitsValue.size() - i] == ZERO){
        ++i; ++countZeros;
    }
    tleft = countZeros; i = countZeros = 0;

    while (right.bitsValue[i] == ZERO){
        ++i; ++countZeros;
    }
    lright = countZeros; i = countZeros = 0;

    while (right.bitsValue[right.bitsValue.size() - i] == ZERO){
        ++i; ++countZeros;
    }
    tright = countZeros; i = countZeros = 0;

    unsigned int lenght = std::max((left.bitsValue.size() + right
       .bitsValue.size() - tleft - tright - lleft - lright),
                                   (tleft + tright))+1;
    RangeParametr retValue("NameRet", lenght);
    retValue.initialiseAllThisType(UNKNOWN);

   tempConcatenate.initialiseCountThisType(tleft + tright, ZERO);

    return retValue + tempConcatenate;

}
```

Заключение

Статический анализ позволяет определить размеры всех промежуточных вычислений без запуска программы.

Реализация метода, описанная в этой статье будет использована в ОРС для компилятора с языка С на HDL. Она позволит верно расставить размеры промежуточных данных на конвейере, который будет располагаться на процессоре с реконфигурируемой архитектурой.

Список литературы

1. *Макмуров И.* Выбор между микропроцессором и FPGA // Электронные компоненты. — 2011. — С. 3. — URL: `http://www.russianelectronics.ru/developer-r/review/2189/doc/56299/`.
2. Языки программирования и компиляторы — 2017 / под ред. Д. Д.В. — Южный федеральный университет, 2017. — С. 282. — ISBN 978-5-9275-2349-8.
3. *Mark Stephenson Jonathan Babb S. A.* Bitwidth Analysis with Application to Silicon Compilation // Massachusetts Institute of Technology Laboratory for Computer Science. — 1999. — С. 13.
4. *Budiu M., Goldstein S. C.* BitValue Inference: Detecting and Exploiting Narrow Bitwidth Computations // Proceedings of 6th International Euro-Par Conference. — 2000. — С. 28.

Приложение А. Пример корректности работы/

Пусть значение "0" и "1" соответствуют реальным значениям "0" и "1" .

Значение "2" соответствует "unknown" и значение "3" соответствует "DontCare" .

Пусть a и b целочисленные переменные, которые были сразу инициализированны определенными значениями.

Рассмотрим следующие примеры:

a = 0 0 0

b = 1 1 1

a + b = 0 1 1 1

a - b = 1 1 0 0

Теперь изменим a.

a = 0 1 0

Power2 a * b = 0 0 0 0 1 0

a - b = 0 0 1 0

Вновь изменим входные данные.

a = 0 0 3

a * b = 0 0 0 0 0 3

Изменим a и b.

a = 2 0 3

b = 0 0 0

a + b = 0 2 0 0

a * b = 0 0 0 3 0 0

a - b = 0 0 0 0