Санкт–Петербургский Государственный Университет
Институт Информационных Технологий

PADLA–based Codesign Toolset

версия 1.0

Д.Ю.Булычев
А.А.Симановский
М.Н.Смирнов
Д.А.Шапоренков

Санкт–Петербург
2002

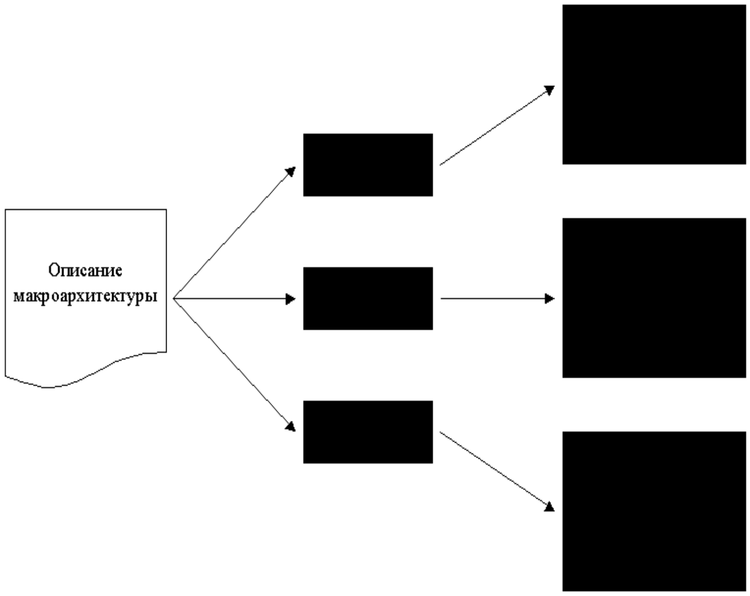
Содержание

1 Введение

Данный текст описывает комплекс программных средств, предназначенных для поддержки процесса разработки программно–аппаратных систем в рамках технологии, основанной на применении языка описания макроархитектуры.

2 Архитектура системы

В основе системы лежит описание макроархитектуры процессора, из которого автоматически генерируются все необходимые средства разработки, а также модель микроархитектуры (VHDL–описание процессора). Схематически такая ситуация изображена на иллюстрации ниже.



В качестве языка описания макроархитектуры избран язык PADLA (Processor Architecture Description Language)[1]. В состав системы входят следующие компоненты:

- верификатор описания макроархитектуры
- утилита генерации ассемблера
- тилита генерации дизассемблера
- утилита генерации описания микроархитектуры
- утилита генерации симулятора
- библиотеки поддержки
- скриптовые средства верхнего уровня

Система написана целиком на языке Objective Caml и работает в любой среде, поддерживающей Objective Caml с обеспечением динамической загрузки модулей и стандартные средства разработки типа `bash`, `make` и т.д. (Linux, Cygwin).

Поскольку язык описания макроархитектуры не определяет формат двоичного представления чисел, система использует библиотеки динамической поддержки, инкапсулирующие те операции, которые зависят от этого представления, причем для утилит генерации эти библиотеки должны загружаться динамически. По умолчанию поставляется библиотека работы с числами в двоичном дополнительном коде (`bincompl.cma`), но могут быть написаны и пользовательские. Интерфейс библиотек динамической поддержки и правила их написания изложены ниже.

Для удобства работы предусмотрена возможность предварительного запуска препроцессора на описание системы команд. По умолчанию в качестве препроцессора используется стандарнтый препроцессор C/C++, верификатор описания макроархитектуры распознает директиву `#line`.

3 Установка и конфигурирование

Для установки системы необходимо разместить где-нибудь принадлежащие ей файлы и позаботиться, чтобы каталог с утилитами был доступен для поиска с помощью системных соглашений (например, упомянуть его в переменной окружения PATH). Для удобства работы все необходимые параметры указываются в конфигурационном файле padla.cnf, который располагается в том же месте, где и все утилиты.

Конфигурационный файл состоит из последовательности строк вида `опция=значение`, опции имеют следующий смысл:

- `-pp` — имя команды для препроцессирования
- `-locallibs` — имя каталога, содержащего собственные библиотеки системы
- `-camllibs` — имя каталога, содержащего библиотеки системы программирования Objective Caml
- `-runtime` — имя библиотеки динамической поддержки

4 Верификатор описания макроархитектуры

Верификатор осуществляет проверку корректности описания макроархитектуры, написанного на языке PADLA, заданного в виде текстового файла. Помимо собственно проверки корректности верификатор преобразует описание во внутреннее промежуточное представление. Все остальные средства системы работают именно со внутренним представлением. Верификатор доступен как в виде отдельной утилиты так и в виде библиотеки.

4.1 Формат вызова и ключи

Вызов верификатора из командной строки выглядит следующим образом:

```
padla [<имя файла-описания>] [<опции>]
```

где <имя файла-описания> — имя файла с описанием макроархитектуры, возможные опции таковы:

- -o <имя выходного файла> — задает имя файла, содержащего отверифицированное описание во внутреннем формате. Если это имя не указано, то верификатор не создает файл во внутреннем формате, а только осуществляет верификацию исходного файла
- -in <имя входного файла> — задает имя файла, хранящего сериализованное внутреннее представление верифицированной макроархитектуры
- -il <имя файла> — инициирует выдачу верификатором дампа внутреннего представления в формате html. Если имя файла не содержит расширения .html, то оно добавляется автоматически
- -decompile <имя файла> — инициирует выдачу верификатором восстановленного из внутреннего представления текстового описания макроархитектуры
- -pp <команда> — задает команду для вызова препроцессора
- -runtime <имя> — задает имя библиотеки динамической поддержки
- -help — инициирует выдачу верификатором справки о возможных опциях
- -dl, -dp — недокументированные опции для отладки

5 Утилита генерации ассемблера

Данная программа (genasm) предназначена для автоматической генерации ассемблера по файлу описания макроархитектуры. При этом порождаются:

- файлы с лексическим и синтаксическим анализаторами на языке Objective Caml
- файлы common.mak и Makefile для сборки ассемблера с помощью стандартной утилиты make

Типичный сценарий использования генератора ассемблера состоит из следующих шагов:

1. написать описание архитектуры
2. сгенерировать ассемблер, вызвав genasm из командной строки
3. в случае успешной генерации вызвать make и получить бинарный файл

Необходимо учитывать, что для выполнения шага 3 нужно иметь среду Objective Caml и библиотеку поддержки genutils.a.

5.1 Формат вызова и ключи

Использование genasm из командной строки выглядит следующим образом:

```
genasm <имя файла-описания> [опции]
```

где <имя файла-описания> — входной файл на языке PADLA. Данный параметр можно не указывать, если указана опция -in. Возможные опции таковы:

- -trace — добавить в генерируемый ассемблер отладочную печать
- -in <имя файла> — указать имя сериализованного файла PADLA
- -libdir <путь> — установить путь к библиотекам в генерируемом common.mak
- -intdir <путь> — установить путь к дополнительным файлам интерфейсов библиотеки динамической поддержки в генерируемом common.mak
- -srcdir <путь> — установить путь для сгенерированных файлов ассемблера
- -bindir <путь> — установить путь для исполняемого файла ассемблера
- -o <имя файла> — задать имя генерируемого make бинарного файла ассемблера
- -yacc <имя файла> — задать имя файла с грамматикой для утилиты ocamlyacc, суффикс .mly добавляется автоматически
- -lex <имя файла> — задать имя файла с лексическими правилами для утилиты ocamllex, суффикс .mll добавляется автоматически
- -yacc-prelude <имя файла> — имя файла со стандартным вступлением для утилиты ocamlyacc
- -lex-prelude <имя файла> — имя файла со стандартным вступлением для утилиты ocamllex
- -yacc-myprelude <имя файла> — имя файла с пользовательским вступлением для утилиты ocamlyacc
- -lex-myprelude <имя файла> — имя файла с пользовательским вступлением для утилиты ocamllex
- -ourelf — использовать реализацию .elf–файла, поддерживающую символы
- -locallibs <имя каталога> — имя каталога с native-библиотеками динамической поддержки

Генерируемый ассемблер по входну файлу с текстом на языке ассемблера генерирует elf–файл и имеет следующий формат входной строки:

```
<имя-ассемблера> <файл с программой> [-o <имя elf-файла>]
```

где

- `<файл с программой>` — входной файл на языке ассемблера
- `<имя elf-файла>` — имя выходного файла в формате `elf`, к имени будет добавлен суффикс `.out`

5.2 Алгоритмы построения ассемблера

Генерируемый ассемблер представляет собой обыкновенный LALR–парсер в виде грамматики для генератора синтаксических анализаторов `ocamlyacc`. `genasm` представляет собой построитель конечного автомата, распознающего язык лексем, создаваемого на основе входного PADLA–файла: `genasm` выделяет из языка ассемблера, задаваемого входным PADLA–файлом, набор лексем согласно следующему правилу:

- параметры инструкций превращаются в специальную лексему с атрибутами ширины и значения
- остальные лексемы строятся расщеплением строковых литералов, использованных во входном описании. Расщепление производится до тех пор, пока некоторая лексема является символьным префиксом другой лексемы
- операция конкатенации с пробелом приволит к переключению состояния парсера на ожидание пробела

Описание инструкции представляется в виде конечного языка над алфавитом построенных лексем. Затем аналогичная операция повторяется для кодов (за исключением того, что расщепление лексем не производится). Далее для конечного языка описания инструкции проводится построение конечного автомата (стандартным способом путем эмуляции его работы на всех возможных входах). Полученный автомат преобразуется в грамматику `ocamlyacc` стандартным способом (состояние — в нетерминал, переход — в правило), финальное состояние – в правило с кодогенерирующей семантикой. По полученному набору лексем генерируется лексический анализатор. По полученным правилам — синтаксический. Семантиками в синтаксическом анализаторе являются запоминание значения лексемы `unbound` и кодогенерация в финальных правилах, кроме того, на каждую конкатенацию с пробелом генерируется ещё одно правило с семантикой, переключающей состояние ожидания пробела. Добавляются правила для обработки последовательности инструкций и другие правила из стандартного вступления и пользовательского вступления (если последнее было передано генератору).

6 Утилита генерации дизассемблера

Утилита генерации дизассемблера `gendasm` предназначена для автоматической генерации дизассемблера по файлу описания макроархитектуры. На вход утилита принимает описание макроархитектуры либо в виде исходного

PADLA–файла, либо в виде результата обработки этого файла верификатором PADLA. В результате работы утилиты создается файл, содержащий исходный текст дизассемблера на языке Objective Caml и файлы `common.mak` и `Makefile`, необходимые для его сборки.

6.1 Формат вызова и ключи

Синтаксис вызова генератора из командной строки выглядит следующем образом:

```
gendasm [опции] [<имя файла-описания архитектуры>]
```

Описание значений различных ключей командной строки:

- -help — вывод краткой справочной информации
- -o <имя файла> — опция задает имя выходного файла, содержащего исходный текст дизассемблера К указанному имени присоединяется суффикс .ml. Это же имя используется для исполняемого файла дизассемблера
- -libs <путь> — указывает каталог, в котором находятся модули библиотеки поддержки. Этот путь используется в генерируемом Makefile
- -objs <путь> — указывает каталог, в котором находятся дополнительные модули. Как и в предыдущем случае, этот путь используется в Makefile
- -srcdir <путь> — задает каталог для исходных файлов дизассемблера
- -bindir <путь> — задает каталог для исполняемого файла дизассемблера
- -in <имя файла> — указывает имя файла, содержащего результат обработки исходного файла описания архитектуры верификатором PADLAЭтот файл используется в качестве источника описания архитектуры вместо исходного
- -opt — включает режим оптимизации генерируемого дизассемблера (о режиме оптимизации см. ниже)

6.2 Генерируемый дизассемблер

Этот раздел описывает устройство генерируемого дизассемблера. Следует заметить, что на текущий момент дизассемблер представляет собой объединение стандартного (т.е. не зависящего от входного описания) кода и данных, содержание которых определяется входным файлом. Эти данные представляют собой наборы масок, извлеченные из описаний битового представления инструкций. Алгоритм работы дизассемблера заключается в следующем. Дизассемблер поддерживает указатель на текущую позицию в секции .text. Он пытается считать последовательность бит длины, соответствующей минимальной длине инструкции, и сопоставить эту последовательность с одной из масок, допустимых для инструкций данной длины. Если ни одна из масок не подходит, дизассемблер повторяет ту же операцию со последовательностью большей длины, в соответствии со следующим (по длине) классом инструкций и т.д. Если подходящую маску найти так и не удалось, выдается сообщение об ошибке и процесс останавливается.

6.3 Оптимизация дизассемблера

Генерируемый по умолчанию дизассемблер может оказаться весьма неэффективным, поскольку алгоритм дизассемблирования будет представлять собой последовательный перебор всех масок для инструкций данной длины. С помощью ключа `-opt` можно указать генератору выполнить оптимизацию генерируемого дизассемблера. При включенной оптимизации генератор будет пытаться группировать маски, имеющие одинаковые биты на одних и тех же позициях. Тем самым, процесс перебора будет оптимизирован за счет устранения лишних проверок битов, что значительно уменьшает время работы результирующего дизассемблера.

7 Утилита генерации симулятора

Утилита генерации симулятора `gensim` предназначена для генерации симулятора системы команд. Утилитв воспринимает на вход описание макроархитектуры либо в виде исходного PADLA–файла, либо в виде результата обработки этого файла верификатором PADLA. В результате работы утилиты создаются файлы, содержащие исходный текст симулятора на языке Objective Caml и файлы `common.mak` и `Makefile`, необходимые для его сборки.

7.1 Формат вызова и ключи

Синтаксис вызова генератора из командной строки выглядит следующем образом:

```
gensim [опции] [<имя файла-описания архитектуры>]
```

Описание значений различных ключей командной строки:

- `-help` — вывод краткой справочной информации
- `-o <имя файла>` — опция задает имя для исполняемого файла симулятора
- `-libs <путь>` — указывает каталог, в котором находятся модули библиотеки поддержки. Этот путь используется в генерируемом `Makefile`
- `-srcdir <путь>` — задает каталог для исходных файлов симулятора
- `-intdir <путь>` — задает каталог для поиска дополнительных интерфейсных файлов
- `-bindir <путь>` — задает каталог для исполняемого файла симулятора

- -in <имя файла> — указывает имя файла, содержащего результат обработки исходного файла описания архитектуры верификатором PADLA. Этот файл используется в качестве источника описания архитектуры вместо текстового

- -back — включает генерацию поддержки обратного исполнения в порождаемом симуляторе

7.2 Команды порождаемого симулятора

Порожденный симулятор является интерактивной средой и поддерживает исполнение следующих команд:

- l <filename> <address> — загрузить файл в память по данному адресу. <filename> — имя файла, <address> — выражение, определяющее адрес. Загружаемый файл должен иметь формат загрузочного модуля, порожденного с помощью сгенерированного ассемблера

- r — запустить программу на исполнение, начиная с текущего места

- b <condition> — установить условие останова. <condition> — выражение, определяющее условие

- d <number> — удалить условие останова. <number> — десятичная константа, определяющая номер условия останова

- list — выдать список условий останова

- s — выполнить одну инструкцию

- j <address> — установить счетчик команд. <address> — выражение, определяющее его значение

- dec — установить десятичный формат вывода чисел

- bin — установить двоичный формат вывода чисел

- hex — установить шестнадцатеричный формат вывода чисел

- oct — установить восьмиричный формат вывода чисел

- ? <resource> [(<address> [, <occurencies>])] — вывести содержимое нескольких экземпляров ресурса, начиная с данного адреса. Здесь <resource> — идентификатор ресурса, <address>, <occurencies> — выражения, определяющие начальный адрес и количество экземпляров соответственно

- ! <resource> <value> — записать значение в ресурс. Здесь <resource> — выражение, определяющее ресурс, <value> — выражение, определяющее записываемое значение

- info — выдать список доступных ресурсов

- a <address> <length> — дизассемблировать содержимое <length> последовательных ячеек программной памяти, начиная с адреса <address>. Здесь <address> и <length> — выражения

- `q` — выйти из симулятора
- `h` — выдать краткую справку о командах

Выражения, используемые в командах управления симулятором, могут состоять из констант в том же формате, что и в языке PADLA, идентификаторов ресурсов, скобок, бинарных и унарных арифметических операций, операций сравнения и логических связок. Например,

```
b pc == 0x100
```

определяет точку останова — выполнение программы будет приостановлено при условии `pc == 0x100`. Аналогично

```
b r(1) == mem(r(2))
```

определяет точку останова по условию равенства содержимого ресурса `r(1)` ресурсу `mem(r(2))`.

8 Утилита генерации VHDL

Утилита генерации архитектуры на языке VHDL `genvhdl` предназначена для автоматической генерации архитектуры по файлу описания макроархитектуры. Типичный сценарий использования утилиты:

1. написать файл с описанием макроархитектуры
2. получить верифицированное представление описания макроархитектуры с помощью верификатора
3. сгенерировать архитектуру, вызвав из командной строки `genvhdl`
4. в случае успешной генерации получившийся файл на языке VHDL можно синтезировать и развести в реальный кристалл ПЛИС с помощью стандартных средств проектирования.

Необходимо учитывать, что для выполнения шага 3 нужно иметь стандартные средства проектирования ПЛИС.

8.1 Формат вызова и ключи

Использование `genvhdl` из командной строки выглядит следующим образом:

```
genvhdl -in [<имя файла-описания>]
```

Здесь `<имя файла-описания>` — входной файл, содержащий представление описания макроархитектуры, полученный из файла на языке PADLA. Данный параметр можно не указывать, в таком случае будет браться файл с именем `device.mm`. В результате генерации получается файл на языке VHDL с именем `device.vhd`. Получившийся файл на языке VHDL содержит описание интерфейса и функциональности описанного устройства.

Также, получаются отдельные файлы, содержащие описание внешних ресурсов (`ext.vhd`), тестовой оболочки (`tb.vhd`), интерфейса с внешними ресурсами (`str.vhd`) и интерфейса с тестовой оболочкой (`tbstr.vhd`). Особенностью текущей версии генератора архитектуры является то, что он предоставляет возможность генерировать только описание архитектуры с регистровой памятью. Другие виды памяти пока не подержаны. Поддерживаются арифметические команды, команды загрузки и выгрузки, переходы и другие, более сложные команды. Недостатком текущей версии является то, что все команды являются однотактными, что может снизить производительность устройства.

Список литературы

[1] PADLA — Processor Architecture Description Language. Технический отчет.