Архитектура v1model

Приведённый ниже включаемый файл (include) с переведёнными на русский язык комментариями содержит определения пакета V1Switch для архитектуры v1model и платформы BMV2. Файл v1model.p4 размещается в каталоге p4include пакета p4c (https://github.com/p4lang/p4c/blob/master/p4include/v1model.p4).

Этот файл включается в программы P4, работающие с прототипом коммутатора simple_switch на основе BMV2, включая тестовые примеры из пакета p4c (https://github.com/p4lang/p4c/tree/master/testdata).

```
/*
Copyright 2013-present Barefoot Networks, Inc.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
*/

/* Объявление P4-16 для модели коммутатора P4 v1.0 */

/* Примечание 1. Более подробное описание архитектуры v1model доступно
 * по приведённой ниже ссылке.
 *
 * https://github.com/p4lang/behavioral-model/blob/master/docs/simple_switch.md[1]
 *
 * Примечание 2. В начале 2019 г. в рабочей группе P4 несколько раз
 * обсуждались способы вызова операций resubmit, recirculate и clone3
 * из соответствующих элементов управления (control), но значениями
 * сохраняемых полей будут значения на момент завершения вызова
 * элемента управления. Так эти операции определены в P4_14. См.
 * https://github.com/p4lang/behavioral-model/blob/master/docs/simple_switch.md#restrictions-on-recirculate-resubmit-and-clone-operations1
 *
 * Примечание 3. Имеются реализации P4_14, где вызов операции
 * generate_digest в field_list приводит к созданию сообщения для
 * плоскости управления, содержащего значения этих полей при завершении
 * работы элемента управления ingress, которые могут отличаться от
 * значений этих же полей в момент вызова из программы операции
 * generate_digest, если значения этих полей меняются при вызове
 * элемента управления P4_14 ingress.
 *
 * Реализации P4_16 с моделью v1model всегда следует создавать
 * сообщения digest, содержащие значения указанных полей на момент вызова
 * внешней функции digest. Программа P4_14 с описанным выше поведением,
 * скомпилированная с использованием p4c, может вести себя иначе.
 */

#ifndef _V1_MODEL_P4_
#define _V1_MODEL_P4_

#include "core.p4"

#ifndef V1MODEL_VERSION
#define V1MODEL_VERSION 20180101
#endif

match_kind {
    range,
    // Точное или шаблонное (соответствует любому) совпадение.
    optional,
    // Служит для реализации dynamic_action_selection.
    selector
}

const bit<32> __v1model_version = V1MODEL_VERSION;

#if V1MODEL_VERSION >= 20200408
typedef bit<9>  PortId_t;       // Не должно иметь постоянный размер?
#endif

@metadata @name("standard_metadata")
struct standard_metadata_t {
#if V1MODEL_VERSION >= 20200408
```

[1]Имеется перевод документа на русский язык.

```
    PortId_t    ingress_port;
    PortId_t    egress_spec;
    PortId_t    egress_port;
#else
    bit<9>      ingress_port;
    bit<9>      egress_spec;
    bit<9>      egress_port;
#endif
    bit<32>     instance_type;
    bit<32>     packet_length;
    //
    // @alias служит для создания раздела field_alias в файле BMV2 JSON.
    // Псевдоним поля создаёт отображение имени метаданных в программе P4
    // на внутреннее имя метаданных в модели поведения (bmv2). Здесь это
    // служит для раскрытия всех метаданных, поддерживаемых simple_switch,
    // для пользователя через standard_metadata_t.
    //
    // «Сглаживающие» поля из bmv2-ss для запроса метаданных.
    @alias("queueing_metadata.enq_timestamp")
    bit<32> enq_timestamp;
    @alias("queueing_metadata.enq_qdepth")
    bit<19> enq_qdepth;
    @alias("queueing_metadata.deq_timedelta")
    bit<32> deq_timedelta;
    /// Глубина очереди в момент извлечения пакета из неё.
    @alias("queueing_metadata.deq_qdepth")
    bit<19> deq_qdepth;

    // Внутренние метаданные
    @alias("intrinsic_metadata.ingress_global_timestamp")
    bit<48> ingress_global_timestamp;
    @alias("intrinsic_metadata.egress_global_timestamp")
    bit<48> egress_global_timestamp;
    /// Идентификатор multicast-группы (ключ для таблицы репликации mcast)
    @alias("intrinsic_metadata.mcast_grp")
    bit<16> mcast_grp;
    /// Идентификатор репликации для multicast
    @alias("intrinsic_metadata.egress_rid")
    bit<16> egress_rid;
    /// Указывает отрицательный результат verify_checksum().
    /// 1 при ошибке контрольной суммы, иначе 0.
    bit<1>  checksum_error;
    /// Ошибка при синтаксическом анализе.
    error parser_error;
    /// Задание приоритета для пакета.
    @alias("intrinsic_metadata.priority")
    bit<3> priority;
}

enum CounterType {
    packets,
    bytes,
    packets_and_bytes
}

enum MeterType {
    packets,
    bytes
}

extern counter
#if V1MODEL_VERSION >= 20200408
<I>
#endif
{
    /***
     * Объект counter (счётчик) создаётся вызовом конструктора. При
     * этом создаётся массив состояний счётчика с набором состояний,
     * заданным параметром size. Индексы массива имеют значения
     * [0, size-1].
     *
     * Нужно указать, будут ли учитываться только пакеты
     * (CounterType.packets), только байты (CounterType.bytes) или
     * то и другое (CounterType.packets_and_bytes).
     *
     * Счётчики могут обновляться из программы P4, но для чтения
     * доступны лишь плоскости управления. Если требуется чтение и
     * запись из программы P4, следует использовать регистры.
     */
    counter(bit<32> size, CounterType type);
    /* FIXME - Аргумент size должен иметь тип int, но это нарушает
     * проверку типов
     */

    /***
     * count() вызывает для указанного индексом состояния считывание,
```

```
     * изменение и запись обратно в виде неделимого (atomic) блока
     * относительно обработки других пакетов. Обновление счётчиков
     * пакетов, байтов или обоих зависит от CounterType экземпляра
     * счётчика при вызове для него конструктора.
     *
     * @param index Индекс обновляемого состояния счётчика в массиве.
     *              Обычно это значение из диапазона [0, size-1].
     *              Если index >= size, состояние счётчика не меняется.
     */
#if V1MODEL_VERSION >= 20200408
    void count(in I index);
#else
    void count(in bit<32> index);
#endif
}

extern direct_counter {
    /***
     * Объект direct_counter создаётся вызовом его конструктора. Нужно
     * указать подсчёт только пакетов (CounterType.packets), только
     * байтов (CounterType.bytes) или обоих (CounterType.packets_and_bytes).
     * После создания объекта его можно связать с одной таблицей, добавляя
     * в её определение указанное ниже свойство
     *     counters = <object_name>;
     *
     * Счётчики могут обновляться из программы P4, но для чтения
     * доступны лишь плоскости управления. Если требуется чтение и
     * запись из программы P4, следует использовать регистры.
     */
    direct_counter(CounterType type);
    /***
     * Метод count() реально не требуется в архитектуре v1model.
     * Это связано с тем, что после привязки объекта direct_counter к
     * таблице, как описано в документации конструктора direct_counter,
     * при каждом обращении к таблице с возвратом подходящей записи
     * состояние счётчика, связанного с этой записью, считывается,
     * изменяется и записывается обратно неделимым блоком относительно
     * обработки других пакетов независимо от вызова метода count() из
     * тела данного действия.
     */
    void count();
}

#define V1MODEL_METER_COLOR_GREEN  0
#define V1MODEL_METER_COLOR_YELLOW 1
#define V1MODEL_METER_COLOR_RED    2

extern meter
#if V1MODEL_VERSION >= 20200408
<I>
#endif
{
    /***
     * Объект meter создаётся вызовом его конструктора. При этом
     * создаётся массив состояний, число которых определяется параметром
     * size. Индексы массива лежат в диапазоне [0, size-1]. Например,
     * при наличии в системе 128 «потоков» с номерами от 0 до 127 и
     * желании независимо измерять каждый из них, можно создать
     * измеритель с size=128.
     *
     * Нужно указать для измерителя учёт пакетов независимо от размера
     * (MeterType.packets) или числа байтов в пакетах (MeterType.bytes).
     */
    meter(bit<32> size, MeterType type);
    /* FIXME - Аргумент size должен иметь тип int, но это нарушает
     * проверку типов
     */

    /***
     * execute_meter() вызывает для указанного состояния считывание,
     * изменение и запись обратно в виде неделимого блока относительно
     * обработки других пакетов и представление одним из цветов
     * (green, yellow, red) в параметре out.
     *
     * @param index Индекс обновляемого состояния в массиве. Обычно из
     *              диапазона [0, size-1]. Если index >= size, состояние
     *              не обновляется.
     * @param result Тип T должен быть bit<W> с W >= 2. Если индекс
     *              находится в диапазоне, будет указано значение 0 для
     *              GREEN, 1 - для YELLOW и 2 - для RED (см. RFC 2697 и
     *              RFC 2698). Если индекс выходит за пределы диапазона,
     *              результат становится неопределённым и его следует
     *              игнорировать вызывающему.
     */
#if V1MODEL_VERSION >= 20200408
    void execute_meter<T>(in I index, out T result);
```

```
#else
    void execute_meter<T>(in bit<32> index, out T result);
#endif
}

extern direct_meter<T> {
    /***
     * Объект direct_meter создаётся вызовом его конструктора. Нужно
     * указать учёт пакетов независимо от размера (MeterType.packets)
     * или числа байтов в пакетах (MeterType.bytes). Созданный объект
     * можно связать с одной таблицей, добавляя в её определение
     * приведённое ниже свойство
     *     meters = <object_name>;
     */
    direct_meter(MeterType type);
    /***
     * После привязки объекта direct_meter к таблице, как описано в
     * документации конструктора direct_meter, при каждом нахождении
     * в таблице совпадающей записи значение связанного с этой записью
     * измерителя считывается, обновляется и записывается обратно в
     * форме неделимого блока относительно обработки других пакетов
     * независимо от вызова метода read() из тела данного действия.
     *
     * Метод read() можно вызывать лишь внутри действия, выполняемого в
     * результате совпадения записи в таблице, с которой связан объект
     * direct_meter. Результатом вызова read() будет численное
     * представление цвета (green, yellow, red) в параметре out.
     *
     * @param result Тип T должен быть bit<W> с W >= 2. Указывается
     *               значение 0 для GREEN, 1 - для YELLOW и 2 - для RED
     *               (см. RFC 2697 и RFC 2698).
     */
    void read(out T result);
}

#if V1MODEL_VERSION >= 20200408
extern register<T, I>
#else
extern register<T>
#endif
{
    /***
     * Объект register создаётся вызовом его конструктора. При этом
     * создаётся массив из size идентичных элементов типа T с индексами
     * из диапазона [0, size-1]. Например, вызов
     *     register<bit<32>>(512) my_reg;
     * выделит место для 512 значений типа bit<32>.
     */
    register(bit<32> size);
    /* FIXME - Аргумент size должен иметь тип int, но это нарушает
     * проверку типов
     */
    /***
     * Метод read() считывает из массива регистров указанное индексом
     * состояние и возвращает его значение в параметре result.
     *
     * @param index Индекс элемента в массиве регистров для чтения.
     *              Обычно лежит в диапазоне [0, size-1].
     * @param result Поддерживаются лишь типы bit<W>. Когда индекс
     *               находится в диапазоне, result получает значение
     *               указанного элемента массива. Если index >= size,
     *               результат будет неопределённым и его следует
     *               игнорировать вызывающему.
     */
    @noSideEffects
#if V1MODEL_VERSION >= 20200408
    void read(out T result, in I index);
#else
    void read(out T result, in bit<32> index);
#endif
    /***
     * Метод write() записывает значение параметра value в массив
     * регистров по указанному индексу.
     *
     * Если нужно выполнить read(), а затем write() для одного
     * элемента массива регистров и хочется сделать блок
     * read-modify-write неделимым относительно обработки других пакетов
     * (параллельное выполнение в архитектуре v1model) требуется
     * указать блок P4_16 с аннотацией @atomic (см. спецификацию P4_16).
     *
     * @param index Индекс элемента массива регистров для записи. Обычно
     *              из диапазона [0, size-1]. Если index >= size, состояние
     *              регистров не изменяется.
     * @param value Поддерживаются лишь типы bit<W>. Когда индекс
     *              находится в диапазоне, значение value записывается
     *              в указанный индексом элемент массива регистров.
```

```
     */
#if V1MODEL_VERSION >= 20200408
    void write(in I index, in T value);
#else
    void write(in bit<32> index, in T value);
#endif
}

// Служит атрибутом реализации таблицы
extern action_profile {
    action_profile(bit<32> size);
}

/***
 * Генерирует случайное значение из диапазона [lo..hi] и записывает
 * его в параметр result. Значение result становится неопределённым,
 * если lo > hi.
 *
 * @param T          Должно иметь тип bit<W>
 */
extern void random<T>(out T result, in T lo, in T hi);

/***
 * Вызов digest вызывает передачу сообщения, содержащего значения,
 * указанные в параметре data, программе плоскости управления. Это
 * похоже на отправку клона пакета плоскости управления, но может
 * быть более эффективным за счёт того, что сообщения обычно меньше
 * пакета и можно собрать несколько таких сообщений в блок для
 * одновременной отправки программе плоскости управления.
 *
 * Значениями полей, передаваемых в сообщении плоскости управления,
 * будут соответствующие значения в момент вызова digest, даже если
 * эти поля позднее будут изменены элементом управления ingress
 * (см. Примечание 3).
 *
 * Вызов digest поддерживается лишь из элемента управления ingress.
 * После вызова метода нельзя отказаться от производимого им эффекта.
 *
 * Если тип T является именованной структурой, имя служит для генерации
 * API плоскости управления.
 *
 * Реализация архитектуры v1model в BMv2 игнорирует значение параметра
 * receiver.
 */
extern void digest<T>(in bit<32> receiver, in T data);

enum HashAlgorithm {
    crc32,
    crc32_custom,
    crc16,
    crc16_custom,
    random,
    identity,
    csum16,
    xor16
}

@deprecated("Please use mark_to_drop(standard_metadata) instead.")
extern void mark_to_drop();

/***
 * mark_to_drop(standard_metadata) является примитивом действия, меняющим
 * standard_metadata.egress_spec на зависимое от реализации специальное
 * значение, что в некоторых случаях ведёт к отбрасыванию пакета в конце
 * входной или выходной обработки. Устанавливается также значение 0 для
 * standard_metadata.mcast_grp. Любое из этих полей метаданных может быть
 * изменено кодом P4 после вызова mark_to_drop(), что может поменять для
 * пакета поведение на отличное от его отбрасывания (drop).
 *
 * В параграфе «Pseudocode for what happens at the end of ingress and
 * egress processing» документа
 * https://github.com/p4lang/behavioral-model/blob/master/docs/simple_switch.md[1]
 * описан относительный приоритет возможных действий применительно к пакету
 * по завершении входной и выходной обработки.
 */
@pure
extern void mark_to_drop(inout standard_metadata_t standard_metadata);

/***
 * Расчёт хэш-функции для значения, указанного параметром data. Результат,
 * записываемый в параметр out с именем result, всегда будет находиться в
 * диапазоне [base, base+max-1], если max >= 1. При max=0 будет записано base.
 *
 * Отметим, что типы параметров могут быть одинаковыми или разными и битовые
```

[1]Параграф «Псевдокод для завершения входной и выходной обработки» в переводе.

```
 * размеры могут различаться.
 *
 * @param O   Должен иметь тип bit<W>
 * @param D   Должен иметь тип tuple, где все поля являются битовыми (bit<W> или int<W>) или varbit.
 * @param T   Должен иметь тип bit<W>
 * @param M   Должен иметь тип bit<W>
 */
@pure
extern void hash<O, T, D, M>(out O result, in HashAlgorithm algo, in T base, in D data, in M max);

extern action_selector {
    action_selector(HashAlgorithm algorithm, bit<32> size, bit<32> outputWidth);
}

enum CloneType {
    I2E,
    E2E
}

@deprecated("Please use verify_checksum/update_checksum instead.")
extern Checksum16 {
    Checksum16();
    bit<16> get<D>(in D data);
}

/***
 * Проверяет контрольную сумму представленных данных. При обнаружении
 * несоответствия поле standard_metadata checksum_error будет иметь
 * в начале входной обработки значение 1.
 *
 * Вызов verify_checksum поддерживается лишь из элемента VerifyChecksum.
 *
 * @param T           Должен иметь тип tuple, все элементы которого имеют
 *                    тип bit<W>, int<W> или ,varbit<W>. Общий размер полей
 *                    должен быть кратным выходному размеру.
 * @param O           Тип контрольной суммы (bit<X>).
 * @param condition   Значение false, задаёт совпадение при любом результате.
 * @param data        Данные для проверки контрольной суммы.
 * @param checksum    Ожидаемая контрольная сумма (должно быть l-value).
 * @param algo        Алгоритм расчёта контрольной суммы (могут поддерживаться
 *                    не все алгоритмы), должен быть известной при компиляции
 *                    константой.
 */
extern void verify_checksum<T, O>(in bool condition, in T data, in O checksum, HashAlgorithm algo);

/***
 * Рассчитывает контрольную сумму представленных данных и записывает её в
 * параметр checksum.
 *
 * Вызов update_checksum поддерживается лишь из элементов управления
 * ComputeChecksum.
 *
 * @param T           Должен иметь тип tuple, все элементы которого имеют
 *                    тип bit<W>, int<W> или ,varbit<W>. Общий размер полей
 *                    должен быть кратным выходному размеру.
 * @param O           Тип контрольной суммы (bit<X>).
 * @param condition   При значении false параметр checksum не изменяется.
 * @param data        Данные для расчёта контрольной суммы.
 * @param checksum    Контрольная сумма данных.
 * @param algo        Алгоритм расчёта контрольной суммы (могут поддерживаться
 *                    не все алгоритмы), должен быть известной при компиляции
 *                    константой.
 */
@pure
extern void update_checksum<T, O>(in bool condition, in T data, inout O checksum, HashAlgorithm algo);

/***
 * verify_checksum_with_payload идентичен verify_checksum, но учитывает
 * при расчёте контрольной суммы данные из пакета (все байты, которые не
 * разбираются синтаксическим анализатором).
 *
 * Вызов verify_checksum_with_payload поддерживается лишь из элемента
 * управления VerifyChecksum.
 */
extern void verify_checksum_with_payload<T, O>(in bool condition, in T data, in O checksum, HashAlgorithm
algo);

/**
 * update_checksum_with_payload идентичен update_checksum, но учитывает
 * при расчёте контрольной суммы данные из пакета (все байты, которые не
 * разбираются синтаксическим анализатором).
 *
 * Вызов update_checksum_with_payload поддерживается лишь из элемента
 * управления ComputeChecksum.
 */
@noSideEffects
```

```
extern void update_checksum_with_payload<T, O>(in bool condition, in T data, inout O checksum,
HashAlgorithm algo);

/***
 * Вызов resubmit в процессе выполнения элемента ingress будет при
 * некоторых документированных условиях приводить к повторному
 * представлению пакета, т. е. пакет будет заново обработан анализатором
 * с тем же содержимым, которое было при предыдущей обработке. Различие
 * состоит лишь в значении поля standard_metadata instance_type и
 * заданных пользователем полей метаданных, которые операция resubmit
 * будет сохранять.
 *
 * Значения пользовательских метаданных, сохраняемых при повторном
 * представлении пакетов, являются значениями в конце входной обработки,
 * а не значениями в момент вызова resubmit (см. Примечание 2).
 *
 * Вызов resubmit поддерживается лишь из элемента управления ingress.
 * Отменить воздействие этого вызова невозможно. При неоднократном вызове
 * resubmit в одном процессе выполнения элемента ingress повторно
 * представляется лишь один пакет и сохраняются данные лишь для последнего
 * вызова (см. Примечание 1).
 */
extern void resubmit<T>(in T data);

/***
 * Вызов recirculate в процессе выполнения элемента ingress будет при
 * некоторых документированных условиях приводить к повторной обработке
 * пакета, начиная с синтаксического анализа пакета, созданного сборщиком
 * (deparser). Рециркулирующие пакеты могут при входной обработке
 * различаться с новыми пакетами значением поля standard_metadata
 * instance_type. Вызывающий может запросить сохранение некоторых
 * полей пользовательских метаданных при рециркуляции пакета.
 *
 * Значения пользовательских метаданных, сохраняемых при рециркуляции
 * пакетов, являются значениями в конце выходной обработки,
 * а не значениями в момент вызова recirculate (см. Примечание 2).
 *
 * Вызов recirculate поддерживается лишь из элемента управления egress.
 * Отменить воздействие этого вызова невозможно. При неоднократном вызове
 * recirculate в одном процессе выполнения элемента egress рециркулирует
 * лишь один пакет и сохраняются данные лишь для последнего
 * вызова (см. Примечание 1).
 */
extern void recirculate<T>(in T data);

/***
 * Операция clone почти идентична clone3 и отличается лишь тем, что
 * никогда не сохраняет поля пользовательских метаданных в клонах пакетов.
 * Она эквивалентна вызову clone3 с теми же параметрами type и session, но
 * с пустым параметром data.
 */
extern void clone(in CloneType type, in bit<32> session);

/***
 * Вызов clone3 при выполнении элемента управления ingress или egress
 * приводит к созданию клонов пакета (иногда называют отражением).
 * Т. е. могут создаваться копии пакета, каждая из которых затем
 * подвергается выходной обработке как независимый пакет. Для исходного
 * пакета продолжается обычная обработка независимо от клонов.
 *
 * Целочисленный параметр session содержит идентификатор сеанса клонирования
 * (иногда называется идентификатором сеанса отражения). Программы плоскости
 * управления должны настроить конфигурацию каждой сессии, которую планируется
 * использовать. Без этого клонирования происходить не будет. Обычно такая
 * настройка включает задание плоскостью управления выходного порта, куда
 * будет передаваться клон или списка пар (port, egress_rid), для которых
 * следует создавать клоны подобно групповым пакетам.
 *
 * Клонированные пакеты можно различать по значениям поля
 * standard_metadata instance_type.
 *
 * Вызывающий может запросить сохранение некоторых полей пользовательских
 * метаданных исходного пакета при клонировании. Сохраняться будут значения
 * в конце входной или выходной обработки, а не в момент вызова clone3
 * (см. Примечание 2).
 *
 * При вызове clone3 во время входной обработки первым параметром должен быть
 * CloneType.I2E, для вызова при выходной обработке - CloneType.E2E.
 *
 * Отменить воздействие этого вызова невозможно. При неоднократном вызове
 * clone3 и/или clone в одном процессе входной или выходной обработки
 * используется лишь последняя сессия клонирования и данные (см. Примечание 1).
 */
extern void clone3<T>(in CloneType type, in bit<32> session, in T data);

extern void truncate(in bit<32> length);
```

```
/***
 * Вызов assert с аргументом, имеющим значение true не даёт эффекта, за
 * исключением тех, которые может давать вычисление аргумента (см. ниже).
 * Если аргумент имеет значение false, поведение зависит от целевой платформы,
 * но цель заключается в записи или выводе информации об отказавшем операторе
 * assert и возможно дополнительных данных об отказе.
 *
 * Например, на платформе simple_switch выполнение оператора assert с
 * аргументом false вызывает сообщение с именем файла и номером строки с
 * оператором assert, после чего процесс simple_switch завершается.
 *
 * Если была задана опция --ndebug в командной строке p4c при компиляции,
 * скомпилированная программа ведёт себя как при отсутствии операторов
 * assert в исходном коде.
 *
 * Настоятельно рекомендуется избегать выражений в качестве аргумента
 * assert, поскольку это может приводить к побочным эффектам, например,
 * вызову внешнего метода или функции с побочным эффектом. Компилятор p4c
 * позволяет это, не выдавая предупреждений. Это рекомендуется потому, что
 * программа будет вести себя так же, как при удалении операторов assert.
 */
extern void assert(in bool check);

/***
 * Для целей компиляции и запуска программ P4 на целевом устройстве
 * операторы assert и assume являются идентичными, включая использование
 * опции --ndebug компилятор p4c для их исключения. См. описание assert.
 *
 * Причиной использования assume как отдельной функции является допущение
 * о её специальной трактовке инструментами формальной проверки. Для
 * некоторых таких инструментов цель состоит в попытке найти примеры
 * пакетов и установленных записей таблиц, где условие оператора assert
 * имеет значение false.
 *
 * Предположим, что такой инструмент запускается для программы и примером
 * служит пакет MPLS, где hdr.ethernet.etherType == 0x8847. При рассмотрении
 * примера видно, что условие assert имеет значение false. Однако планируется
 * развёртывание программы P4 в сети, где нет пакетов MPLS. Можно добавить
 * в программу P4 дополнительные условия для корректной обработки пакетов без
 * отказов assert, но проще сказать инструменту, что «такие экземпляры пакетов
 * не применимы и не нужно их показывать» путём добавления оператора
 *     assume(hdr.ethernet.etherType != 0x8847);
 * в подходящее место программы. Тогда формальный инструмент не будет показывать
 * такие примеры, ограничиваясь лишь теми, которые дают условие true.
 *
 * Причина одинакового поведения операторов assume и assert при компиляции для
 * целевого устройства заключается в том, что значение false при оценке
 * условия во время работы в сети может говорить об ошибочности допущения
 * и необходимости его проверки.
 */
extern void assume(in bool check);

/*
 * Вывод заданного пользователем сообщения.
 * Например, log_msg("User defined message");
 * или log_msg("Value1 = {}, Value2 = {}",{value1, value2});
 */
extern void log_msg(string msg);
extern void log_msg<T>(string msg, in T data);

// Имя standard_metadata является зарезервированным.

/*
 * Архитектура.
 *
 * M должно иметь тип struct.
 *
 * H должно иметь тип struct, где каждый из элементов является заголовком,
 * стеком или объединением заголовков.
 */

parser Parser<H, M>(packet_in b,
                    out H parsedHdr,
                    inout M meta,
                    inout standard_metadata_t standard_metadata);

/*
 * В теле элемента управления VerifyChecksum допускаются лишь операторы
 * блока, вызовы методов verify_checksum и verify_checksum_with_payload,
 * а также операторы return.
 */
control VerifyChecksum<H, M>(inout H hdr,
                             inout M meta);
@pipeline
control Ingress<H, M>(inout H hdr,
```

```
                      inout M meta,
                      inout standard_metadata_t standard_metadata);
@pipeline
control Egress<H, M>(inout H hdr,
                     inout M meta,
                     inout standard_metadata_t standard_metadata);

/*
 * В теле элемента управления ComputeChecksum допускаются лишь операторы
 * блока, вызовы методов update_checksum и update_checksum_with_payload,
 * а также операторы return.
 */
control ComputeChecksum<H, M>(inout H hdr,
                              inout M meta);

/*
 * В теле элемента управления Deparser допускается лишь метод packet_out.emit().
 */
@deparser
control Deparser<H>(packet_out b, in H hdr);

package V1Switch<H, M>(Parser<H, M> p,
                       VerifyChecksum<H, M> vr,
                       Ingress<H, M> ig,
                       Egress<H, M> eg,
                       ComputeChecksum<H, M> ck,
                       Deparser<H> dep
                       );

#endif  /* _V1_MODEL_P4_ */
```

Николай Малых

nmalykh@protokols.ru