Архитектура переносимых коммутаторов PSA

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

Этот файл включается в программы P4, работающие с прототипом коммутатора psa_switch на основе модели BMV2.

```
/* 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.
*/

#ifndef __PSA_P4__
#define __PSA_P4__

#include<core.p4>

#ifndef _PORTABLE_SWITCH_ARCHITECTURE_P4_
#define _PORTABLE_SWITCH_ARCHITECTURE_P4_

/**
 *   Объявления P4-16 для архитектуры PSA
 */

/**
 * Эти типы нужно определить до включения файла архитектуры, а затем
 * следует определить защиту макросов.
 */
#define PSA_ON_BMV2_CORE_TYPES
#ifdef PSA_ON_BMV2_CORE_TYPES
/* Показанные ниже размеры в битах относятся к платформе BMv2 psa_switch.
 * Размер этих типов не требуется сохранять в других реализациях PSA.
 * Каждая реализация может указать свои размеры в форме bit<W> с нужным W.
 * Одной из причин приведённого здесь выбора является реализация PSA в
 * модели BMv2. Другая заключается в упрощении компиляции этого файла и
 * примера программы PSA (P4), включающей файл.
 *
 * Размеры в битах для BMv2 psa_switch выбраны так, чтобы они совпадали с
 * соответствующими типами InHeader. Это упрощает реализацию P4Runtime для
 * BMv2 psa_switch. */

/* В определениях используется typedef, а не type, что просто задаёт другие
 * имена для типа bit<W> с показанным значением W. В отличие от определений
 * type значения, объявленные с typedef, можно свободно смешивать в
 * выражениях со значениями типа bit<W>. Значения, объявленные с type,
 * нельзя свободно смешивать, пока они не приведены к соответствующему типу
 * typedef. Это может быть неудобно для арифметических операций, зато можно
 * пометить нужным типом все значения, объявленные с type, при генерации
 * API плоскости управления.
 *
 * Отметим, что размер typedef <name>Uint_t всегда совпадает с type <name>_t. */
typedef bit<32> PortIdUint_t;
typedef bit<32> MulticastGroupUint_t;
typedef bit<16> CloneSessionIdUint_t;
typedef bit<8>  ClassOfServiceUint_t;
typedef bit<16> PacketLengthUint_t;
typedef bit<16> EgressInstanceUint_t;
typedef bit<64> TimestampUint_t;

/* Отметим, что clone_spec в BMv2 simple_switch v1model имеет размер 32 бита,
 * но используется так, что 16 битов содержат идентификатор сеанса клонирования
 * и другие 16 битов - численный идентификатор field_list. Лишь 16 битов
 * идентификатора сессии можно сравнивать с типов CloneSessionIdUint_t. См.
 * https://github.com/p4lang/behavioral-model/blob/master/targets/simple_switch/simple_switch.cpp
 */

@p4runtime_translation("p4.org/psa/v1/PortId_t", 32)
type PortIdUint_t         PortId_t;
@p4runtime_translation("p4.org/psa/v1/MulticastGroup_t", 32)
type MulticastGroupUint_t MulticastGroup_t;
@p4runtime_translation("p4.org/psa/v1/CloneSessionId_t", 16)
```

```
type CloneSessionIdUint_t CloneSessionId_t;
@p4runtime_translation("p4.org/psa/v1/ClassOfService_t", 8)
type ClassOfServiceUint_t ClassOfService_t;
@p4runtime_translation("p4.org/psa/v1/PacketLength_t", 16)
type PacketLengthUint_t   PacketLength_t;
@p4runtime_translation("p4.org/psa/v1/EgressInstance_t", 16)
type EgressInstanceUint_t EgressInstance_t;
@p4runtime_translation("p4.org/psa/v1/Timestamp_t", 64)
type TimestampUint_t      Timestamp_t;
typedef error   ParserError_t;

const PortId_t PSA_PORT_RECIRCULATE = (PortId_t) 0xfffffffa;
const PortId_t PSA_PORT_CPU = (PortId_t) 0xfffffffd;

const CloneSessionId_t PSA_CLONE_SESSION_TO_CPU = (CloneSessionId_t) 0;

#endif  // PSA_ON_BMV2_CORE_TYPES

#ifndef PSA_ON_BMV2_CORE_TYPES
#error "Please define the following types for PSA and the PSA_EXAMPLE_CORE_TYPES macro"
// BEGIN:Type_defns
/* В определениях используется typedef, а не type, что просто задаёт другие
 * имена для типа bit<W> с показанным значением W. В отличие от определений
 * type значения, объявленные с typedef, можно свободно смешивать в
 * выражениях со значениями типа bit<W>. Значения, объявленные с type,
 * нельзя свободно смешивать, пока они не приведены к соответствующему типу
 * typedef. Это может быть неудобно для арифметических операций, зато можно
 * пометить нужным типом все значения, объявленные с type, при генерации
 * API плоскости управления.
 *
 * Отметим, что размер typedef <name>Uint_t всегда совпадает с type <name>_t. */
typedef bit<unspecified> PortIdUint_t;
typedef bit<unspecified> MulticastGroupUint_t;
typedef bit<unspecified> CloneSessionIdUint_t;
typedef bit<unspecified> ClassOfServiceUint_t;
typedef bit<unspecified> PacketLengthUint_t;
typedef bit<unspecified> EgressInstanceUint_t;
typedef bit<unspecified> TimestampUint_t;

@p4runtime_translation("p4.org/psa/v1/PortId_t", 32)
type PortIdUint_t         PortId_t;
@p4runtime_translation("p4.org/psa/v1/MulticastGroup_t", 32)
type MulticastGroupUint_t MulticastGroup_t;
@p4runtime_translation("p4.org/psa/v1/CloneSessionId_t", 16)
type CloneSessionIdUint_t CloneSessionId_t;
@p4runtime_translation("p4.org/psa/v1/ClassOfService_t", 8)
type ClassOfServiceUint_t ClassOfService_t;
@p4runtime_translation("p4.org/psa/v1/PacketLength_t", 16)
type PacketLengthUint_t   PacketLength_t;
@p4runtime_translation("p4.org/psa/v1/EgressInstance_t", 16)
type EgressInstanceUint_t EgressInstance_t;
@p4runtime_translation("p4.org/psa/v1/Timestamp_t", 64)
type TimestampUint_t      Timestamp_t;
typedef error   ParserError_t;

const PortId_t PSA_PORT_RECIRCULATE = (PortId_t) unspecified;
const PortId_t PSA_PORT_CPU = (PortId_t) unspecified;

const CloneSessionId_t PSA_CLONE_SESSION_TO_CPU = (CloneSessiontId_t) unspecified;
// END:Type_defns
#endif  // #ifndef PSA_EXAMPLE_CORE_TYPES

// BEGIN:Type_defns2
/* Все типы с InHeader в имени предназначены для передачи лишь значений
 * соответствующих типов из заголовков пакета между устройством PSA и
 * управляющей им программой сервера P4Runtime.
 *
 * Предполагается, что размер будет не меньше, чем у любого устройства PSA
 * для данного типа. Таким образом, эти типы могут также быть полезны для
 * определения заголовков пакета, передаваемых между устройством PSA и
 * другими устройствами без прохождения через сервер P4Runtime (например,
 * для отправки пакетов контроллеру или системе сбора данных с
 * использованием скорости передачи пакетов выше, чем может обработать
 * сервер P4Runtime). При использовании для таких целей не предъявляется
 * требования к плоскости данных PSA по автоматическому численному
 * преобразованию этих типов, которое происходило бы при прохождении
 * заголовков через сервер P4Runtime. Все нужные преобразования следует
 * выполнять программисту P4 явно в кода программы.
 *
 * Все размеры должны быть кратны 8, поэтому любое подмножество этих
 * полей можно использовать в одном определении заголовка P4, даже в
 * реализациях P4, ограничивающих заголовки полями с общим размером,
 * кратным 8 битам. */

/* См. комментарий выше (PortIdUint_t). */
typedef bit<32> PortIdInHeaderUint_t;
```

```
typedef bit<32> MulticastGroupInHeaderUint_t;
typedef bit<16> CloneSessionIdInHeaderUint_t;
typedef bit<8>  ClassOfServiceInHeaderUint_t;
typedef bit<16> PacketLengthInHeaderUint_t;
typedef bit<16> EgressInstanceInHeaderUint_t;
typedef bit<64> TimestampInHeaderUint_t;

@p4runtime_translation("p4.org/psa/v1/PortIdInHeader_t", 32)
type  PortIdInHeaderUint_t         PortIdInHeader_t;
@p4runtime_translation("p4.org/psa/v1/MulticastGroupInHeader_t", 32)
type  MulticastGroupInHeaderUint_t MulticastGroupInHeader_t;
@p4runtime_translation("p4.org/psa/v1/CloneSessionIdInHeader_t", 16)
type  CloneSessionIdInHeaderUint_t CloneSessionIdInHeader_t;
@p4runtime_translation("p4.org/psa/v1/ClassOfServiceInHeader_t", 8)
type  ClassOfServiceInHeaderUint_t ClassOfServiceInHeader_t;
@p4runtime_translation("p4.org/psa/v1/PacketLengthInHeader_t", 16)
type  PacketLengthInHeaderUint_t   PacketLengthInHeader_t;
@p4runtime_translation("p4.org/psa/v1/EgressInstanceInHeader_t", 16)
type  EgressInstanceInHeaderUint_t EgressInstanceInHeader_t;
@p4runtime_translation("p4.org/psa/v1/TimestampInHeader_t", 64)
type  TimestampInHeaderUint_t      TimestampInHeader_t;
// END:Type_defns2

/* Функции _int_to_header были написаны для преобразования значений типа
 * <name>_t (значение внутри пути данных) в значения типа <name>InHeader_t
 * в заголовке, который будет передан в порт CPU.
 *
 * Функции _header_to_int  были написаны для преобразования значений в
 * обратном направлении, обычно при назначении из заголовка, полученного из
 * CPU, значению, которое будет использоваться в остальном коде программы.
 *
 * Причина этих трёх преобразований заключается в том, что каждый из этих
 * типов объявлен через P4_16 type, поэтому без приведения типа значения
 * можно присваивать лишь идентичному типу. Первое приведение меняет
 * исходный тип на bit<W1> с тем же размером W1. Второе приведение меняет
 * размер путём добавления нулей в начале или отбрасывания старших битов.
 * Третье приведение меняет тип bit<W2> на окончательный тип с размером W2. */

PortId_t psa_PortId_header_to_int (in PortIdInHeader_t x) {
    return (PortId_t) (PortIdUint_t) (PortIdInHeaderUint_t) x;
}
MulticastGroup_t psa_MulticastGroup_header_to_int (in MulticastGroupInHeader_t x) {
    return (MulticastGroup_t) (MulticastGroupUint_t) (MulticastGroupInHeaderUint_t) x;
}
CloneSessionId_t psa_CloneSessionId_header_to_int (in CloneSessionIdInHeader_t x) {
    return (CloneSessionId_t) (CloneSessionIdUint_t) (CloneSessionIdInHeaderUint_t) x;
}
ClassOfService_t psa_ClassOfService_header_to_int (in ClassOfServiceInHeader_t x) {
    return (ClassOfService_t) (ClassOfServiceUint_t) (ClassOfServiceInHeaderUint_t) x;
}
PacketLength_t psa_PacketLength_header_to_int (in PacketLengthInHeader_t x) {
    return (PacketLength_t) (PacketLengthUint_t) (PacketLengthInHeaderUint_t) x;
}
EgressInstance_t psa_EgressInstance_header_to_int (in EgressInstanceInHeader_t x) {
    return (EgressInstance_t) (EgressInstanceUint_t) (EgressInstanceInHeaderUint_t) x;
}
Timestamp_t psa_Timestamp_header_to_int (in TimestampInHeader_t x) {
    return (Timestamp_t) (TimestampUint_t) (TimestampInHeaderUint_t) x;
}

PortIdInHeader_t psa_PortId_int_to_header (in PortId_t x) {
    return (PortIdInHeader_t) (PortIdInHeaderUint_t) (PortIdUint_t) x;
}
MulticastGroupInHeader_t psa_MulticastGroup_int_to_header (in MulticastGroup_t x) {
    return (MulticastGroupInHeader_t) (MulticastGroupInHeaderUint_t) (MulticastGroupUint_t) x;
}
CloneSessionIdInHeader_t psa_CloneSessionId_int_to_header (in CloneSessionId_t x) {
    return (CloneSessionIdInHeader_t) (CloneSessionIdInHeaderUint_t) (CloneSessionIdUint_t) x;
}
ClassOfServiceInHeader_t psa_ClassOfService_int_to_header (in ClassOfService_t x) {
    return (ClassOfServiceInHeader_t) (ClassOfServiceInHeaderUint_t) (ClassOfServiceUint_t) x;
}
PacketLengthInHeader_t psa_PacketLength_int_to_header (in PacketLength_t x) {
    return (PacketLengthInHeader_t) (PacketLengthInHeaderUint_t) (PacketLengthUint_t) x;
}
EgressInstanceInHeader_t psa_EgressInstance_int_to_header (in EgressInstance_t x) {
    return (EgressInstanceInHeader_t) (EgressInstanceInHeaderUint_t) (EgressInstanceUint_t) x;
}
TimestampInHeader_t psa_Timestamp_int_to_header (in Timestamp_t x) {
    return (TimestampInHeader_t) (TimestampInHeaderUint_t) (TimestampUint_t) x;
}

/// Диапазон поддерживаемых значений для свойств таблицы psa_idle_timeout
enum PSA_IdleTimeout_t {
  NO_TIMEOUT,
  NOTIFY_CONTROL
```

```
};

// BEGIN:Metadata_types
enum PSA_PacketPath_t {
    NORMAL,     /// Пакет, полученный ingress и не относящийся к перечисленным ниже.
    NORMAL_UNICAST,   /// Обычный индивидуальный пакет, принятый egress.
    NORMAL_MULTICAST, /// Обычный групповой пакет, принятый egress.
    CLONE_I2E,  /// Пакет, созданный клонированием в ingress и предназначенный
                /// для egress.
    CLONE_E2E,  /// Пакет, созданный клонированием в egress и предназначенный
                /// для egress.
    RESUBMIT,   /// Пакет, принятый в результате операции resubmit.
    RECIRCULATE /// Пакет, принятый в результате операции recirculate.
}

struct psa_ingress_parser_input_metadata_t {
  PortId_t                ingress_port;
  PSA_PacketPath_t        packet_path;
}

struct psa_egress_parser_input_metadata_t {
  PortId_t                egress_port;
  PSA_PacketPath_t        packet_path;
}

struct psa_ingress_input_metadata_t {
  // Все эти значения инициализируются архитектурой до начала выполнения
  // блока управления Ingress.
  PortId_t                ingress_port;
  PSA_PacketPath_t        packet_path;
  Timestamp_t             ingress_timestamp;
  ParserError_t           parser_error;
}
// BEGIN:Metadata_ingress_output
struct psa_ingress_output_metadata_t {
  // В комментариях к полям приведены исходные значения в момент начала
  // выполнения блока управления Ingress.
  ClassOfService_t        class_of_service; // 0
  bool                    clone;            // false
  CloneSessionId_t        clone_session_id; // не определено
  bool                    drop;             // true
  bool                    resubmit;         // false
  MulticastGroup_t        multicast_group;  // 0
  PortId_t                egress_port;      // не определено
}
// END:Metadata_ingress_output
struct psa_egress_input_metadata_t {
  ClassOfService_t    class_of_service;
  PortId_t            egress_port;
  PSA_PacketPath_t    packet_path;
  EgressInstance_t    instance;       /// Экземпляр приходит из PacketReplicationEngine
  Timestamp_t         egress_timestamp;
  ParserError_t       parser_error;
}

/// Эта структура является входным (in) параметром для выходного сборщика.
/// Она включает данные для выходного сборщика, позволяющие решить вопрос
/// о рециркуляции пакета.
struct psa_egress_deparser_input_metadata_t {
  PortId_t                egress_port;
}
// BEGIN:Metadata_egress_output
struct psa_egress_output_metadata_t {
  // В комментариях к полям приведены исходные значения в момент начала
  // выполнения блока управления Egress.
  bool                    clone;          // false
  CloneSessionId_t        clone_session_id; // не определено
  bool                    drop;           // false
}
// END:Metadata_egress_output
// END:Metadata_types

/// При выполнении IngressDeparser функция psa_clone_i2e возвращает true
/// тогда и только тогда, когда создаётся клон обрабатываемого пакета для
/// выхода. Какие-либо назначения параметра clone_i2e_meta в
/// IngressDeparser могут выполняться лишь внутри оператора if, который
/// позволяет выполнять такое назначение лишь в случае возврата функцией
/// psa_clone_i2e(istd) значения true. psa_clone_i2e можно реализовать
/// путём возврата istd.clone

@pure
extern bool psa_clone_i2e(in psa_ingress_output_metadata_t istd);

/// При выполнении IngressDeparser функция psa_resubmit возвращает true
/// тогда и только тогда, когда пакет представляется заново (resubmit).
/// Какие-либо назначения параметра resubmit_meta в  IngressDeparser
```

```
/// могут выполняться лишь внутри оператора if, который  позволяет
/// выполнять такое назначение лишь в случае возврата функцией
/// psa_resubmit(istd) значения true. psa_resubmit можно реализовать
/// путём возврата (!istd.drop && istd.resubmit)


@pure
extern bool psa_resubmit(in psa_ingress_output_metadata_t istd);

/// При выполнении IngressDeparser функция psa_normal возвращает true
/// тогда и только тогда, когда пакет передаётся на выход «обычным»
/// путём как индивидуальный или групповой. Какие-либо назначения
/// параметра resubmit_meta в IngressDeparser могут выполняться лишь
/// внутри оператора if, который позволяет выполнять такое назначение
/// лишь в случае возврата функцией psa_normal(istd) значения true.
/// psa_normal можно реализовать путём возврата (!istd.drop && istd.resubmit)

@pure
extern bool psa_normal(in psa_ingress_output_metadata_t istd);

/// При выполнении EgressDeparser функция psa_clone_e2e возвращает true
/// тогда и только тогда, когда создаётся клон обрабатываемого пакета
/// для выхода. Какие-либо назначения параметра clone_e2e_meta в
/// EgressDeparser могут выполняться лишь внутри оператора if, который
/// позволяет выполнять такое назначение лишь в случае возврата функцией
/// psa_clone_e2e(istd) значения true. psa_clone_e2e можно реализовать
/// путём возврата istd.clone

@pure
extern bool psa_clone_e2e(in psa_egress_output_metadata_t istd);

/// При выполнении EgressDeparser функция psa_recirculate возвращает true
/// тогда и только тогда, когда пакет передаётся в рециркуляцию.
/// Какие-либо назначения параметра recirculate_meta в EgressDeparser
/// могут выполняться лишь внутри оператора if, который позволяет
/// выполнять такое назначение лишь в случае возврата функцией
/// psa_recirculate(istd) значения true. psa_recirculate можно
/// реализовать путём возврата (!istd.drop && (edstd.egress_port
/// == PSA_PORT_RECIRCULATE))

@pure
extern bool psa_recirculate(in psa_egress_output_metadata_t istd,
                            in psa_egress_deparser_input_metadata_t edstd);


extern void assert(in bool check);
extern void assume(in bool check);

// BEGIN:Match_kinds
match_kind {
    range,   /// Служит для представления интервалов min..max.
    selector /// Служит для динамического выбора действия с помощью
             /// внешнего метода ActionSelector.
}
// END:Match_kinds

// BEGIN:Action_send_to_port
/// Меняет выходные метаданные ingress для отправки пакета на выходную
/// обработку с последующим выводом в egress_port (при выходной
/// обработке пакет может быть отброшен).

/// Это действие не влияет на операции clone или resubmit.

@noWarnUnused
action send_to_port(inout psa_ingress_output_metadata_t meta,
                    in PortId_t egress_port)
{
    meta.drop = false;
    meta.multicast_group = (MulticastGroup_t) 0;
    meta.egress_port = egress_port;
}
// END:Action_send_to_port

// BEGIN:Action_multicast
/// Меняет выходные метаданные ingress для создания копий пакета,
/// отправляемых на выходную обработку.

/// Это действие не влияет на операции clone или resubmit.

@noWarnUnused
action multicast(inout psa_ingress_output_metadata_t meta,
                 in MulticastGroup_t multicast_group)
{
    meta.drop = false;
    meta.multicast_group = multicast_group;
}
```

```
// END:Action_multicast

// BEGIN:Action_ingress_drop
/// Меняет выходные метаданные ingress для для обычной выходной
/// обработки.

/// Это действие не влияет на операцию clone, но предотвращает
/// повторное представления пакета (resubmit).

@noWarnUnused
action ingress_drop(inout psa_ingress_output_metadata_t meta)
{
    meta.drop = true;
}
// END:Action_ingress_drop

// BEGIN:Action_egress_drop
/// Меняет выходные метаданные egress для передачи пакета из устройства.

/// Это действие не влияет на операцию clone.

@noWarnUnused
action egress_drop(inout psa_egress_output_metadata_t meta)
{
    meta.drop = true;
}
// END:Action_egress_drop

extern PacketReplicationEngine {
    PacketReplicationEngine();
    // Для этого объекта нет методов, вызываемых из программ P4. Метод
    // будет иметь экземпляр с именем, которое плоскость управления
    // может использовать для вызова объекта через API.
}

extern BufferingQueueingEngine {
    BufferingQueueingEngine();
    // Для этого объекта нет методов, вызываемых из программ P4.
    // См. предыдущий комментарий.
}

// BEGIN:Hash_algorithms
enum PSA_HashAlgorithm_t {
  IDENTITY,
  CRC32,
  CRC32_CUSTOM,
  CRC16,
  CRC16_CUSTOM,
  ONES_COMPLEMENT16,  /// 16-битовая контрольная сумма с дополнением до 1,
                      /// используемая в заголовках IPv4, TCP и UDP.
  TARGET_DEFAULT      /// Определяется реализацией целевой платформы.
}
// END:Hash_algorithms

// BEGIN:Hash_extern
extern Hash<O> {
  /// Constructor
  Hash(PSA_HashAlgorithm_t algo);

  /// Расчёт хэш-значения для данных.
  /// @param data - данные для расчёта хэш-значения.
  /// @return - хэш-значение.
  @pure
  O get_hash<D>(in D data);

  /// Расчёт хэш-значения для данных с модулем max и добавлением base.
  /// @param base - минимальное возвращаемое значение.
  /// @param data - данные для расчёта хэш-значения.
  /// @param max - хэш-значение делится на max.
  ///        Реализация может ограничивать поддерживаемое максимальное
  ///        значение (например, 32 или 256), а также может поддерживать
  ///        для него лишь степени 2. Разработчикам P4 следует выбирать
  ///        такие значения для обеспечения переносимости.
  /// @return (base + (h % max)), где h - хэш-значение.
  @pure
  O get_hash<T, D>(in T base, in D data, in T max);
}
// END:Hash_extern

// BEGIN:Checksum_extern
extern Checksum<W> {
  /// Constructor
  Checksum(PSA_HashAlgorithm_t hash);

  /// Сбрасывает внутреннее состояние и готовит модуль к расчёту. Каждый
  /// экземпляр объекта Checksum автоматически инициализируется как при
```

```
  /// вызове clear(). Инициализация выполняется при каждом создании
  /// экземпляра объекта, независимо от применения в анализаторе или
  /// элементе управления. Все состояния, поддерживаемые объектом
  /// Checksum независимы между пакетами.
  void clear();

  /// Добавление данных в контрольную сумму.
  void update<T>(in T data);

  /// Получение контрольной суммы для добавленных (и не удалённых)
  /// с момента последней очистки данных.
  @noSideEffects
  W    get();
}
// END:Checksum_extern

// BEGIN:InternetChecksum_extern
// Контрольная сумма на основе алгоритма ONES_COMPLEMENT16, используемая в
// IPv4, TCP и UDP. Поддерживается инкрементальное обновление методом
// subtract (см. IETF RFC 1624).
extern InternetChecksum {
  /// Конструктор
  InternetChecksum();

  /// Сбрасывает внутреннее состояние и готовит модуль к расчёту. Каждый
  /// экземпляр объекта InternetChecksum автоматически инициализируется как
  /// при вызове clear(). Инициализация выполняется при каждом создании
  /// запуске анализатора или элемента управления, где применяется объект.
  /// Все состояния, поддерживаемые объектом независимы между пакетами.
  void clear();

  /// Добавляет в расчёт контрольной суммы данные data, размер которых
  /// должен быть кратным 16 битам.
  void add<T>(in T data);

  /// Исключает из расчёта контрольной суммы данные data, размер которых
  /// должен быть кратным 16 битам.
  void subtract<T>(in T data);

  /// Возвращает контрольную сумму для данных, добавленных (и не удалённых)
  /// после предшествующего вызова clear.
  @noSideEffects
  bit<16> get();

  /// Возвращает состояние расчёта контрольной суммы для использования при
  /// последующем вызове метода set_state.
  @noSideEffects
  bit<16> get_state();

  /// Возвращает состояние экземпляра InternetChecksum к возвращённому при
  /// предшествующем вызове метода get_state. Состояние может возвращаться
  /// для одного или разных экземпляров InternetChecksum.
  void set_state(in bit<16> checksum_state);
}
// END:InternetChecksum_extern

// BEGIN:CounterType_defn
enum PSA_CounterType_t {
    PACKETS,
    BYTES,
    PACKETS_AND_BYTES
}
// END:CounterType_defn

// BEGIN:Counter_extern
/// Опосредованный счётчик с n_counters независимых значений, где каждое
/// значение имеет заданный плоскостью данных размер W.

extern Counter<W, S> {
  Counter(bit<32> n_counters, PSA_CounterType_t type);
  void count(in S index);

  /*
  /// API плоскости управления использует 64-битовые значения считчиков.
  /// Это не указывает размеры счётчиков в плоскости управления.
  /// Предполагается, что программы управления периодически считывают
  /// значения счётчиков плоскости данных и аккумулирует их в счётчиках
  /// большего размера, в которых максимальное значение достигается реже.
  /// 64-битовые счётчики позволяют работать при скорости порта 100 Гбит/с
  /// в течение 46 лет без переполнения.

  @ControlPlaneAPI
  {
    bit<64> read      (in S index);
    bit<64> sync_read (in S index);
    void set          (in S index, in bit<64> seed);
```

```
    void reset        (in S index);
    void start        (in S index);
    void stop         (in S index);
  }
  */
}
// END:Counter_extern

// BEGIN:DirectCounter_extern
extern DirectCounter<W> {
  DirectCounter(PSA_CounterType_t type);
  void count();

  /*
  @ControlPlaneAPI
  {
    W    read<W>      (in TableEntry key);
    W    sync_read<W> (in TableEntry key);
    void set          (in TableEntry key, in W seed);
    void reset        (in TableEntry key);
    void start        (in TableEntry key);
    void stop         (in TableEntry key);
  }
  */
}
// END:DirectCounter_extern

// BEGIN:MeterType_defn
enum PSA_MeterType_t {
    PACKETS,
    BYTES
}
// END:MeterType_defn

// BEGIN:MeterColor_defn
enum PSA_MeterColor_t { RED, GREEN, YELLOW }
// END:MeterColor_defn

// BEGIN:Meter_extern
// Индексируемый измеритель с n_meters независимых состояний.

extern Meter<S> {
  Meter(bit<32> n_meters, PSA_MeterType_t type);

  // Этот метод служит для «перекрашивания» трафика (см. RFC 2698).
  // «Цвет» пакета перед вызовом метода указан параметром color.
  PSA_MeterColor_t execute(in S index, in PSA_MeterColor_t color);

  // Этот метод служит для «окрашивания» трафика вслепую (см. RFC 2698).
  // Метод может быть реализован вызовом execute(index, MeterColor_t.GREEN).
  PSA_MeterColor_t execute(in S index);

  /*
  @ControlPlaneAPI
  {
    reset(in MeterColor_t color);
    setParams(in S index, in MeterConfig config);
    getParams(in S index, out MeterConfig config);
  }
  */
}
// END:Meter_extern

// BEGIN:DirectMeter_extern
extern DirectMeter {
  DirectMeter(PSA_MeterType_t type);
  // См. аналогичный метод для extern Meter.
  PSA_MeterColor_t execute(in PSA_MeterColor_t color);
  PSA_MeterColor_t execute();

  /*
  @ControlPlaneAPI
  {
    reset(in TableEntry entry, in MeterColor_t color);
    void setConfig(in TableEntry entry, in MeterConfig config);
    void getConfig(in TableEntry entry, out MeterConfig config);
  }
  */
}
// END:DirectMeter_extern

// BEGIN:Register_extern
extern Register<T, S> {
  /// Создание массива из <size> регистров с неопределёнными значениями.
  Register(bit<32> size);
  ///  Создание массива из <size> регистров с заданными значениями.
```

```
  Register(bit<32> size, T initial_value);

  @noSideEffects
  T    read  (in S index);
  void write (in S index, in T value);

  /*
  @ControlPlaneAPI
  {
    T    read<T>      (in S index);
    void set          (in S index, in T seed);
    void reset        (in S index);
  }
  */
}
// END:Register_extern

// BEGIN:Random_extern
extern Random<T> {

  /// Возвращает случайное значение из диапазона [min, max].
  /// Реализациям разрешается поддерживать лишь диапазоны, где
  /// значение (max - min + 1) является степенью 2. Программистам P4
  /// следует ограничивать аргументы значениями, обеспечивающими
  /// переносимость программы.

  Random(T min, T max);
  T read();

  /*
  @ControlPlaneAPI
  {
    void reset();
    void setSeed(in T seed);
  }
  */
}
// END:Random_extern

// BEGIN:ActionProfile_extern
extern ActionProfile {
  /// Создаёт профиль действия записей size
  ActionProfile(bit<32> size);

  /*
  @ControlPlaneAPI
  {
     entry_handle add_member    (action_ref, action_data);
     void         delete_member (entry_handle);
     entry_handle modify_member (entry_handle, action_ref, action_data);
  }
  */
}
// END:ActionProfile_extern

// BEGIN:ActionSelector_extern
extern ActionSelector {
  /// Создаёт селектор действий с числом записей size.
  /// @param algo - алгоритм хэширования для выбора записи в группе;
  /// @param size - число записей в селекторе действий;
  /// @param outputWidth - размер ключа выбора.
  ActionSelector(PSA_HashAlgorithm_t algo, bit<32> size, bit<32> outputWidth);

  /*
  @ControlPlaneAPI
  {
     entry_handle add_member        (action_ref, action_data);
     void         delete_member     (entry_handle);
     entry_handle modify_member     (entry_handle, action_ref, action_data);
     group_handle create_group      ();
     void         delete_group      (group_handle);
     void         add_to_group      (group_handle, entry_handle);
     void         delete_from_group (group_handle, entry_handle);
  }
  */
}
// END:ActionSelector_extern

// BEGIN:Digest_extern
extern Digest<T> {
  Digest();                       /// Определяет поток сообщений (digest) для
                                  /// плоскости управления.
  void pack(in T data);           /// Передаёт данные в поток сообщений.
  /*
  @ControlPlaneAPI
  {
```

```
  T data;                           /// Если T является списком, плоскость управления
                                    /// создаёт структуру (struct).
  int unpack(T& data);              /// Неупакованные данные находятся в T&, int
                                    /// int возвращает код состояния.
  }
  */
}
// END:Digest_extern

// BEGIN:Programmable_blocks
parser IngressParser<H, M, RESUBM, RECIRCM>(
    packet_in buffer,
    out H parsed_hdr,
    inout M user_meta,
    in psa_ingress_parser_input_metadata_t istd,
    in RESUBM resubmit_meta,
    in RECIRCM recirculate_meta);

control Ingress<H, M>(
    inout H hdr, inout M user_meta,
    in    psa_ingress_input_metadata_t  istd,
    inout psa_ingress_output_metadata_t ostd);

control IngressDeparser<H, M, CI2EM, RESUBM, NM>(
    packet_out buffer,
    out CI2EM clone_i2e_meta,
    out RESUBM resubmit_meta,
    out NM normal_meta,
    inout H hdr,
    in M meta,
    in psa_ingress_output_metadata_t istd);

parser EgressParser<H, M, NM, CI2EM, CE2EM>(
    packet_in buffer,
    out H parsed_hdr,
    inout M user_meta,
    in psa_egress_parser_input_metadata_t istd,
    in NM normal_meta,
    in CI2EM clone_i2e_meta,
    in CE2EM clone_e2e_meta);

control Egress<H, M>(
    inout H hdr, inout M user_meta,
    in    psa_egress_input_metadata_t  istd,
    inout psa_egress_output_metadata_t ostd);

control EgressDeparser<H, M, CE2EM, RECIRCM>(
    packet_out buffer,
    out CE2EM clone_e2e_meta,
    out RECIRCM recirculate_meta,
    inout H hdr,
    in M meta,
    in psa_egress_output_metadata_t istd,
    in psa_egress_deparser_input_metadata_t edstd);

package IngressPipeline<IH, IM, NM, CI2EM, RESUBM, RECIRCM>(
    IngressParser<IH, IM, RESUBM, RECIRCM> ip,
    Ingress<IH, IM> ig,
    IngressDeparser<IH, IM, CI2EM, RESUBM, NM> id);

package EgressPipeline<EH, EM, NM, CI2EM, CE2EM, RECIRCM>(
    EgressParser<EH, EM, NM, CI2EM, CE2EM> ep,
    Egress<EH, EM> eg,
    EgressDeparser<EH, EM, CE2EM, RECIRCM> ed);

package PSA_Switch<IH, IM, EH, EM, NM, CI2EM, CE2EM, RESUBM, RECIRCM> (
    IngressPipeline<IH, IM, NM, CI2EM, RESUBM, RECIRCM> ingress,
    PacketReplicationEngine pre,
    EgressPipeline<EH, EM, NM, CI2EM, CE2EM, RECIRCM> egress,
    BufferingQueueingEngine bqe);

// END:Programmable_blocks

#endif  /* _PORTABLE_SWITCH_ARCHITECTURE_P4_ */

#endif   // __PSA_P4__
```

Перевод на русский язык

Николай Малых

nmalykh@protokols.ru