# FSL91030(M) chip

## **SDK API** call description document

Manual version: **V1.4**

Wuhan Feisiling Microelectronics Technology Co., Ltd.

**December 2022**

Table of contents



# 1 Overview

This document describes the SDK and chip-related principles of the FSL91030M series chips. The overall block diagram of the SDK is shown in Figure 1-1. It is mainly provided for software developers who develop based on SDK.

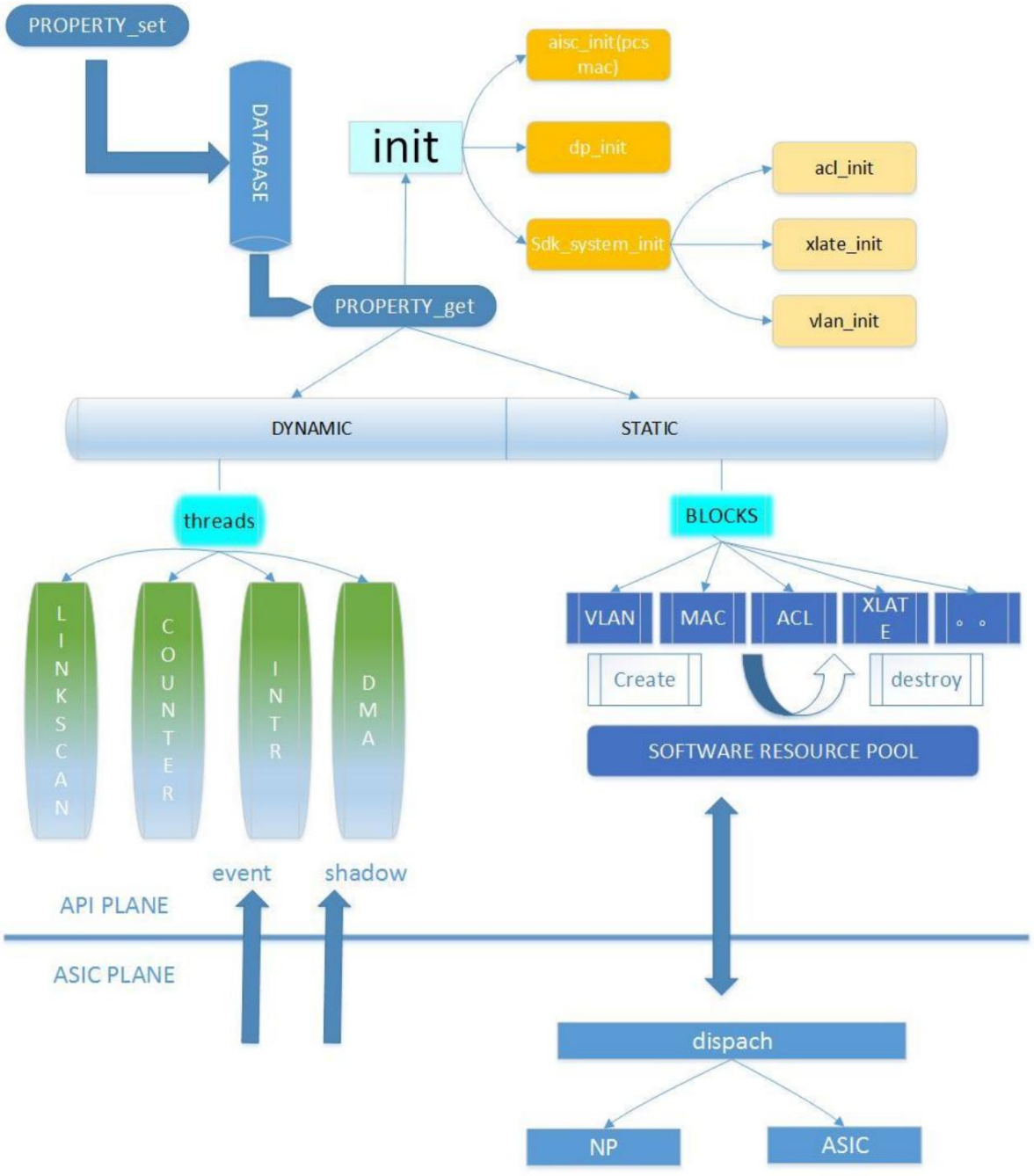
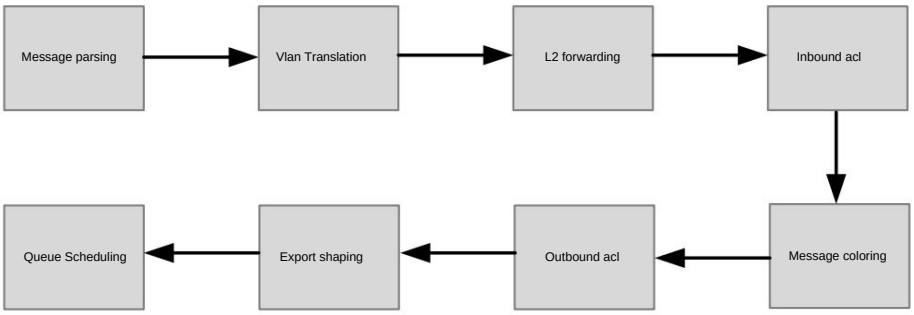



Figure 1-1 SDK overall block diagram

FSL91030M series chips can complete layer 2 forwarding, storm control, acl, vlan translation, message coloring, priority scheduling and other functions.
able:




Each module has a corresponding SDK interface to provide relevant configuration and access. This document combines specific configuration examples to provide users with Instructions for using the relevant API interfaces. Also explains the relevant chip principles.

# 2 Translate

Vlan translation processes the VLAN tag of the message, supports inbound translation and outbound translation, and supports actions such as adding, deleting, modifying, and copying.

Vlan translation supports three modes: vlan-based, mac-based and ip-based. The implementation process is: match the configured vlan translation entry key information and execute the corresponding vlan action.

## 2.1 VLAN - based inbound VLAN translation

Inbound VLAN translation requires enabling the VLAN translation function of the port (supporting xlate0 and xlate1), enabling the VLAN editing function of the port, Configure the key and action of VLAN translation; xlate0 and xlate1 are configured in the same way and can be configured at the same time, with xlate0 having a higher priority.

**int api_ingress_vlan_xlate0_action_add()**

```
{
    int unit = 0;
    fsl_port_t port = 5;
    fsl_port_control_t type1 = fslPortControlXlateEn0; /*xlate0 enable flag*/
    fsl_port_control_t type2 = fslPortControliVtEditEn; /*vlan edit enable flag*/
    int en1 = 1;
    int en2 = 1;
    int gport = port;                    /*gport is a physical port or aggregation port, here it is set to a physical port*/
    int xlate = 0;
    fsl_vlan_translate_key_t key_mode = fslVlanTranslateKeyDouble; /*Match double-layer vlan, svid and cvid*/

    int outer_vlan = 100;
    int inner_vlan = 200;
    fsl_vlan_action_set_t action;

    /* 1. Set port xlate0 to enable */
    fsl_port_control_set(unit, port, type1, en1);
```

```
    /* 2. Set port vlan editing enable*/
    fsl_port_control_set(unit, port, type2, en2);
    action.new_svid = 300;
    action.new_cvid = 400;
    action.vlan_option.dtOvid = fslVlanActionReplace_dt; /*Double-layer vlan replaces the outer layer*/
    action.vlan_option.dtIvid = fslVlanActionReplace_dt; /*Double-layer vlan replaces the inner layer*/

    /* 3. Set the translation key and action */
    fsl_vlan_translate_action_add(unit, xlate, gport, key_mode, outer_vlan, inner_vlan, &action);

        return 0;
}
```

## 2.2 MAC - based inbound VLAN translation

This function requires enabling the port's VLAN translation and VLAN editing functions, and configuring the VLAN translation MAC key and action, xlate0 and xlate1 The configuration methods are consistent and can be configured at the same time. xlate0 has a higher priority.

```
int api_vlan_mac_action_add()
{
    int unit = 0;
    int xlate = 1;
    int port = 4;
    fsl_vlan_translate_key_t key_mode = fslVlanTranslateKeyMac; /*mac key*/
    fsl_mac_t mac = {0, 1, 2, 2, 3, 3};
    fsl_vlan_action_set_t action;
    int gport = port;
    int en1 = 1;
```

```
    int en2 = 1;
    fsl_port_control_t type1 = fslPortControlXlateEn0;                    /*xlate0 enable flag*/
    fsl_port_control_t type2 = fslPortControliVtEditEn;                   /*vlan edit enable flag*/

    /* 1. Set port xlate0 to enable */
    fsl_port_control_set(unit, port, type1, en1);

    /* 2. Set port vlan editing enable*/
    fsl_port_control_set(unit, port, type2, en2);

    /* 3. Set the key and action of vlan translation */
    fsl_vlan_mac_action_add( unit, xlate, gport, key_mode, mac, &action);

    return 0;
}
```

## 2.3 mac ip binding

This function requires enabling the port's VLAN translation and VLAN editing functions, configuring the VLAN translation's IP key and action, and supports both ipv4 and ipv6. Configure the action (global) in the case of mac ip bind miss. The configuration methods of xlate0 and xlate1 are the same and can be configured at the same time. xlate0 has priority high.

```
int api_mac_ip_bind_add()
{
    int unit = 0;
    fsl_vlan_ip_t vlan_ip;
    fsl_mac_t mac = {0, 1, 2, 3, 4, 5};
    int bypassen = 0;
    int trapen = 0;
```

```
int dropen = 1;
int port = 10;
int en1 = 1;
int en2 = 1;
fsl_port_control_t type1 = fslPortControlXlateEn0;                    /*xlate0 enable flag*/
fsl_port_control_t type2 = fslPortControliVtEditEn; /*vlan edit enable flag*/
int gport = port;

/* 1. Set port xlate0 to enable */
fsl_port_control_set(unit, port, type1, en1);

/* 2. Set port vlan editing enable*/
fsl_port_control_set(unit, port, type2, en2);

/* 3. Set the action when bind misses */
fsl_mac_ip_bind_miss_action_set(unit, bypassen, trapen, dropen);

vlan_ip.inport = port;
vlan_ip.ip4 = 0x22446688;
vlan_ip.ip6_flag = 0;                 /*When configuring ipv4, set the flag to 0, and ipv6 to 1*/
vlan_ip.xlate = 0;

/* 4. Set ip key and action */
fsl_vlan_ip_action_add(unit, &vlan_ip, mac, gport);
```

```
    return 0;
}
```

## 2.4 PDU message processing

This function requires setting the PDU header content and its mask, and configuring the hit behavior; the source MAC is a global configuration, and multiple PDU configurations are hit at the same time. The smaller the configuration number, the higher the priority.

```
int api_pdu_action_add()
{
    int unit = 0;
    fsl_mac_t smac = {0, 2, 3, 4, 5, 6};
    fsl_pdu_option_type_t pdu_option = PDU_OPTION_TRAP;
    int index = 5;
    fsl_pdu_config_t pdu_cfg;

    /* 1. Set the global source mac of pdu */
    fsl_pdu_smac_set(unit, smac);

    /* 2. Set pdu identification content and mask*/
    pdu_cfg.ethType = 0x2345;
    pdu_cfg.mask_ethType = 1;
    fsl_pdu_config_add(unit, index, &pdu_cfg);

    /* 3. Set pdu action*/
    fsl_pdu_option_set(unit, index, pdu_option);

    /* 4. Set the trap port to the CPU port*/
    fsl_traffic_trap_port_set(unit, 31); //Port 31 is the CPU port
```

```
    return 0;
}
```

## 2.5 Outbound **VLAN** Translation Based on **VLAN**

This function requires the port VLAN editing function to be enabled (the VLAN translation function is enabled in key ctl and does not need to be configured separately). vlan translation key and action. xlate0 and xlate1 are configured in the same way and can be configured at the same time, with xlate0 having a higher priority.

```
int api_egress_vlan_xlate1_action_add()
{
    int unit = 0;
    int xlate = 1;                          /*Use xlate1*/
    int port = 8;
    int gport = port;
    fsl_vlan_translate_key_t key_mode = fslVlanTranslateKeyDouble;
    int outer_vlan = 333;
    int inner_vlan = 444;
    fsl_port_control_t type = fslPortControlEvtEditEn; /*vlan edit enable flag*/
    int en = 1;
    fsl_vlan_action_set_t action;

    /* 1. Set port vlan editing enable*/
    fsl_port_control_set(unit, port, type, en);

    action.new_svid = 555;
    action.new_cvid = 555;
    action.vlan_option.dtOvid = fslVlanActionReplace_dt; /*Double-layer vlan replaces the outer layer*/
    action.vlan_option.dtIvid = fslVlanActionReplace_dt; /*Double-layer vlan replaces the inner layer*/
```

```
        /* 2. Configure VLAN translation key and action */

        fsl_vlan_translate_egress_action_add(unit, xlate, gport, key_mode, outer_vlan,
inner_vlan, &action);



        return 0;

    }
```

## 2.6 Port protocol **VLAN** configuration

This function requires configuration of protocol key and action. Protocol has higher priority than xlate based on vlan, ip and mac. It supports 16 configurations.

```
    int api_vlan_port_protocol_action_add()

    {

        int unit = 0;

        int inIslag = 0;

        int inPort = 10;

        int ethType = 0x4567;

        fsl_vlan_action_set_t action;



        action.new_svid = 555;

        action.new_cvid = 555;

        action.vlan_option.dtOvid = fslVlanActionReplace_dt; /*Double-layer vlan replaces the outer layer*/

        action.vlan_option.dtIvid = fslVlanActionReplace_dt; /*Double-layer vlan replaces the inner layer*/



        /* 1. Add port protocol vlan configuration*/

        fsl_vlan_port_protocol_action_add(unit, inIslag, inPort, ethType, &action);

        return 0;

    }
```

## 2.7 port **aft** filtering

```
int api_port_aft_set()
{
    int unit = 0;
    int port = 10;
    fsl_port_aft_type_t aft_type = AFT_DROP_STAG; //Discard all stagged frames

    /* 1. Add port aft filtering configuration */
    fsl_port_aft_set(unit, port, aft_type);

    return 0;
}
```

# 3 Policing

Policing mainly controls data traffic and uses the token bucket algorithm to implement traffic supervision. It supports two modes: port-based and flow-based. The token bucket algorithm supports SrTCM and TrTCM, port policing supports 35 ports, and flow policing supports 4096 items.

## 3.1 Port policing

This function requires setting the global packet length calculation method of the port, enabling the port policing function, setting the token bucket refresh enable and refresh cycle, and finally configuring the speed limit related parameters. The speed limit granularity is controlled by the token bucket refresh cycle parameter, which is a global configuration. The refresh cycle parameter cannot be too small, and it is best to set it above 300.

The token bucket refresh setting must be set before the policer is created.

**int api_macro_policing_set()**

```
{
    int rv = 0;
    int direct = 0;
    int unit = 0;
    int port = 1;
    int enable = 1;
    fsl_policing_ctl_t pol_ctl;
    fsl_policer_config_t pol_cfg;
    fsl_policing_update_t pol_upd;

    memset(&pol_ctl, 0, sizeof(fsl_policing_ctl_t));
    pol_ctl.macroPktBytes = 0;          /*Port policing based on bytes (1: based on packets)*/
    pol_ctl.preambleLen = 20;           /*Equivalent packet length of preamble and frame gap*/
    pol_ctl.meterGran = 0;              /*Control granularity*/

    /* 1. Set the calculation method of packet length*/
```

```
rv = fsl_policing_ctl_set(unit, direct, &pol_ctl);
if(rv < 0)
{
    return rv;
}

/* 2. Enable port policing*/
rv = fsl_macro_policing_enable_set(unit, direct, port, enable);
if(rv < 0)
{
    return rv;
}

memset(&pol_upd, 0, sizeof(fsl_policing_update_t));
pol_upd.updEn = 1;
pol_upd.updMaxIndex = 10; /*Maximum address for filling the token bucket*/
pol_upd.timer0 = 1000; /*Token bucket refresh cycle parameter, refresh once every 1000 clock cycles*/
pol_upd.timer0Num = 1;
pol_upd.timer1 = 1000;
pol_upd.timer1Num = 1;

/* 3. Port policing token bucket refresh settings*/
rv = fsl_macro_policing_update_set(unit, direct, &pol_upd);
if(rv < 0)
{
    return rv;
```

```
    }

    memset(&pol_cfg, 0, sizeof(fsl_policer_config_t));
    pol_cfg.mode = fslPolicerModeTrTcm; /*Dual bucket dual rate*/
    pol_cfg.colorSense = COLOR_BLIND;
    pol_cfg.globalCFlag = 0;
    pol_cfg.sharingMode = 0;
    pol_cfg.cir = 2000000;                  /*c Bucket addition rate*/
    pol_cfg.cirMax = 10000000;              /*c Bucket maximum addition rate*/
    pol_cfg.cbs = 4000000;                  /*c barrel depth*/
    pol_cfg.eir = 6000000;
    pol_cfg.eirMax = 8000000;
    pol_cfg.ebs = 20000000;
    pol_cfg.rChangeDrop = 1;                /*Red envelope discarded*/

    /* 4. Create a port policer */
    rv = fsl_macro_policer_create(unit, direct, port, &pol_cfg);

    return rv;
}
```

# 3.2 Flow **Policing**

This function requires setting the global packet length calculation method of the flow, setting the token bucket refresh enable and refresh cycle, and finally configuring the rate limit related parameters. The speed limit granularity is controlled by the token bucket refresh cycle parameter, which is a global configuration. The refresh cycle parameter cannot be too small and is preferably set above 300. pol_id can be set to -1, indicating that the policing id is assigned by the system. 0-4095 is the specified id. In general, the flow policing id is set to vlan id.

The token bucket refresh setting must be set before the policer is created.

```
int api_flow_policing_set()
{
    int direct = 0;
    int unit = 0;
    int port = 1;
    int enable = 1;
    int pol_id = 10;                    /*Policng id is specified as 10*/
    fsl_policing_ctl_t pol_ctl;
    fsl_policer_config_t pol_cfg;
    fsl_policing_update_t pol_upd;

    memset(&pol_ctl, 0, sizeof(fsl_policing_ctl_t));
    pol_ctl.flowPktBytes = 0;           /*Flow mode, policing based on bytes*/
    pol_ctl.preambleLen = 20;

    /* 1. Set the packet length calculation method (flow)*/
    fsl_policing_ctl_set(unit, direct, &pol_ctl);

    memset(&pol_upd, 0, sizeof(fsl_policing_update_t));
    pol_upd.updEn = 1;                  /*Token bucket refresh enable flag*/
    pol_upd.updMaxIndex = 10;
    pol_upd.timer0 = 1000;
    pol_upd.timer0Num = 1;
    pol_upd.timer1 = 1000;
    pol_upd.timer1Num = 1;
```

```
    /* 2. Flow policing token bucket refresh settings*/
    fsl_flow_policing_update_set(unit, direct, &pol_upd);

    memset(&pol_cfg, 0, sizeof(fsl_policer_config_t));
    pol_cfg.mode = fslPolicerModeTrTcm;
    pol_cfg.colorSense = COLOR_BLIND;
    pol_cfg.globalCFlag = 0;
    pol_cfg.sharingMode = 0;
    pol_cfg.cir = 2000000;                  /*c Bucket addition rate*/
    pol_cfg.cirMax = 10000000;              /*c Bucket maximum addition rate*/
    pol_cfg.cbs = 4000000;                  /*c barrel depth*/
    pol_cfg.eir = 6000000;
    pol_cfg.eirMax = 8000000;
    pol_cfg.ebs = 20000000;
    pol_cfg.rChangeDrop = 1;

    /* 3. Create a traffic policer */
    fsl_flow_policer_create(unit, direct, pol_id, &pol_cfg);

    return 0;
}
```

# 4 Map

Different packets use different QoS priorities, for example, VLAN packets use 802.1p, IP packets use dscp, and MPLS packets use

To ensure the quality of service of different messages, when the message enters the device, the QoS priority carried by the message needs to be mapped to the device's

Internal service level (scheduling priority) and discard priority (color) are used to manage congestion within the device according to the internal service level.

Congestion avoidance is performed by using packet colors. When packets leave the device, the internal service level and color need to be mapped to QoS priority and re-marked to the packet.

In this article, the subsequent equipment provides corresponding service quality according to the QoS priority. This is the message priority and re-marking process.

## 4.1 Mapping of VLAN Priority to Internal Priority

This function requires setting the template of VLAN priority mapping, the VLAN priority to be mapped, the internal priority and color to be mapped.

```
int api_vlanpri_map_set()
{
    int unit = 0;
    int qos_pro_index = 1;                  /*qos_pro_index=-1 indicates system allocation*/
    int pkt_pri = 5;                        /*Message priority*/
    int cfi = 0;
    int internal_pri = 3;                   /*Internal priority*/
    fsl_color_t color = fslColorGreen;
    fsl_qos_profile_t qos_profile;

    memset(&qos_profile, 0, sizeof(fsl_qos_profile_t));
    qos_profile.useDefault = 0; /*0: other, 1: use default priority*/
    qos_profile.useL2Info = 1;              /*0: use L3 header information first, 1: use L2 header information*/
    qos_profile.trustCtag = 0; /*0: use stag first, 1: use ctag*/

    /* 1. Create a template */
    fsl_qos_profile_create(unit, qos_pro_index, &qos_profile);
```

```
    /* 2. Set up mapping */
    fsl_vlan_priority_map_set(unit, qos_pro_index, pkt_pri, cfi, internal_pri, color);

    return 0;
}
```

## 4.2 Mapping of dscp to internal priorities

This function requires setting the template of dscp priority mapping, the dscp value to be mapped, the internal priority and color to be mapped.

```
int api_dscp_map_set()
{
    int unit = 0;
    int qos_pro_index = 2;                  /*Specify the template number to be used*/
    int dscp = 55;
    int cfi = 0;
    int internal_pri = 2;
    fsl_color_t color = fslColorYellow;
    fsl_qos_profile_t qos_profile;

    memset(&qos_profile, 0, sizeof(fsl_qos_profile_t));
    qos_profile.useDefault = 0;
    qos_profile.useL2Info = 0;              /*0: use L3 header information first, 1: use L2 header information*/

    /* 1. Create a template */
    fsl_qos_profile_create(unit, qos_pro_index, &qos_profile);

    /* 2. Set up mapping */
```

```
    fsl_dscp_map_set(unit, qos_pro_index, dscp, internal_pri, color);
    return 0;
}
```

## 4.3 Internal priority re-marking **vlan pri**

This function requires enabling the port re-marking function, creating a re-marking template, and setting the internal priority, color, and final mark to be re-marked. Note the resulting vlan pri value.

```
int api_vlanpri_remark_set()
{
    int unit = 0;
    int port = 3;
    int rmk_en = 1;                 /*Re-mark enable*/
    int rmkPriPtr = -1;             /*The system automatically assigns a re-marking template number*/
    int internal_pri = 4;
    fsl_color_t color = fslColorGreen;
    int cos = 6;
    int cfi = 1;
    fsl_rmk_info_t rmk_info;

    /* 1. Enable the port re-marking function*/
    fsl_pri_remark_enable_set(unit, port, rmk_en);

    memset(&rmk_info, 0, sizeof(fsl_rmk_info_t));
    rmk_info.ccosRmkEn = 1;
    rmk_info.scosRmkEn = 1;

    /* 2. Create a re-labeling template*/
```

```
    fsl_remark_profile_create(unit, rmkPriPtr, &rmk_info);

    /* 3. Set re-mark mapping*/
    fsl_vlanpri_unmap_set(unit, rmkPriPtr, internal_pri, color, cos, cfi);

    return 0;
}
```

## 4.4 Internal Priority Re-Marking **DSCP**

This function requires enabling the port re-marking function, creating a re-marking template, and setting the internal priority, color, and final mark to be re-marked.

Make note of the resulting dscp value.

```
int api_dscp_remark_set()
{
    int unit = 0;
    int port = 3;
    int rmk_en = 1;
    int rmkPriPtr = -1;
    int internal_pri = 4;
    fsl_color_t color = fslColorYellow;
    int dscp = 44;
    fsl_rmk_info_t rmk_info;

    /* 1. Enable the port re-marking function*/
    fsl_pri_remark_enable_set(unit, port, rmk_en);

    memset(&rmk_info, 0, sizeof(fsl_rmk_info_t));
    rmk_info.brgChgTos = 1;                    /*Re-mark and modify tos*/
```

```
    rmk_info.onlyChgDscp = 1;                    /*Only modify dscp, and use it together with the above modification tos*/


    /* 2. Create a re-labeling template*/
    fsl_remark_profile_create(unit, rmkPriPtr, &rmk_info);


    /* 3. Set re-mark mapping*/
    fsl_dscp_unmap_set(unit, rmkPriPtr, internal_pri, color, dscp);


    return 0;
}
```

# 5 Storm control

Storm control is a method of monitoring network traffic to prevent the flooding of broadcast, multicast, and unicast packets in the LAN from causing network performance loss.
A data traffic management function that implements control by setting traffic supervision thresholds.

This function supports three modes: system, port and forwarding id. The following API takes port as an example.

## 5.1 Global Configuration of Storm Control

Set the global configuration of storm control, mainly to set the equivalent packet length of the control granularity and the preamble of the frame gap, enable the storm control of the port,
Set the token refresh period, and finally set the rate limit and bucket size.

The refresh cycle setting must be performed before the rate limit setting.

**int api_storm_control_set()**

```
{
    int unit = 0;
    fsl_storm_control_mode_t mode = STORM_CONTROL_PORT;
    int gport = 8;                                  /*Support physical ports, aggregation ports, protection ports, 0-46*/
    fsl_forward_type_t fwd_type = FORWARD_TYPE_MULTICASTCAST; /*Forwarding type, supports unicast, multicast and broadcast
broadcast*/
    int enable = 1;
    uint8_t meter_gran = 0;
    uint8_t preamble_len = 20;                      /*Frame gap and preamble equivalent packet length*/
    fsl_storm_policing_type_t pol_type = STORM_POLICING_BYTE;
    uint32_t limit = 1000000;
    uint32_t burst_size = 2000000;
    fsl_storm_ctl_global_t global_ctl;

    /* 1. Set up global configuration of Storm Control*/
    fsl_storm_control_global_set(unit, meter_gran, preamble_len);
```

```
    /* 2. Enable port storm control function*/
    fsl_storm_control_enable_set(unit, mode, gport, fwd_type, enable);

    memset(&global_ctl, 0, sizeof(fsl_storm_ctl_global_t));
    global_ctl.delayInterval = 1000;
    global_ctl.maxUpdIdx = 10;
    global_ctl.updEn = 1;

    /* 3. Set token bucket update configuration*/
    fsl_storm_control_update_set(unit, mode, &global_ctl);

    /* 4. Set speed limit */
    fsl_storm_control_set(unit, mode, gport, fwd_type, pol_type, limit, burst_size);

    return 0;
}
```

# 6 Forwarding

describe

This functional block is mainly divided into three parts: forwarding, learning, and aging, all of which revolve around the mac address. The FSL91030M chip maintains two mac tables, which are divided into a mac key table and a mac action behavior table (hereinafter collectively referred to as mac tables), which correspond one to one. The mac key table mainly stores mac addresses, vlanids, valid indications, etc., and its corresponding behavior table determines a series of behavior attributes of the mac address, including port numbers, blacklists and whitelists, site shift priorities, static indications, etc.

Forwarding: This part mainly determines the forwarding behavior of the message. When forwarding data, the destination mac of the message is searched to see if it is in the mac address table to determine whether the message is unicast, multicast, flooding, etc. The search method is to form a key through keytype (default is 0) + Mac + vlanid, and use the key to hash out an index to search the mac table. If the search hits, the outbound port of the message is determined according to the port attribute in the mac behavior table. For example: there is a mac of 0x11, vlanid=4095, port=1 in the mac table. Now a message comes in from vlan 4095 and port port=2, and its destination mac is 0x11. Then the message forwarding hit will only be forwarded from port port =1, thereby reducing broadcast flooding.

Learning: Enable the mac address learning function. For the source mac address of the message, it will be learned into the mac table, including the corresponding behavior attributes. All learned are dynamic mac addresses. Mac address learning is also divided into learning hits. After hitting, the mac behavior table will be updated.

Aging: After aging is turned on, the MAC table will be aged in a certain period of time. The time period can be configured, and turning on fast aging will immediately age the MAC table.

Purpose

1. Manually add\delete mac address table by calling interface: The manually added mac address table is a static table, and mac rows can be added
   Behavioral attributes of the table.
2. Mac address table aging time is configurable
3. Disable MAC learning based on interface/VLAN
4. Limit the number of MAC addresses based on interface/VLAN/global

## 6.1 Add a **mac** address

**int api_l2_addr_add()**

{

```
sal_mac_addr_t mac = {0x00, 0x33, 0x44, 0x55, 0x66, 0x77};
fsl_vlan_t vid = 4095;
```

```
        /*mac behavior table flag, default is static address (the following flags represent static address, trunk address, blacklist, whitelist respectively)*/
        uint32_t flags = FSL_L2_STATIC | FSL_L2_TRUNK_MEMBER | FSL_L2_DISCARD_SRC |
FSL_L2_WHITE_LIST ;
        fsl_l2_addr_t l2addr;

        /* Initialize mac address structure*/
        fsl_l2_addr_t_init(&l2addr, mac, vid);
        l2addr.port = 20;
        l2addr.flags = flags;
        /*trunk id*/
        l2addr.tgid = 3;

        /*Add a mac address*/
        fsl_l2_addr_add( 0, &l2addr);
        return 0;
    }
```

## 6.2 Delete a **MAC** address

```
    int api_l2_addr_delete()
    {
        sal_mac_addr_t mac = {0x00, 0x33, 0x44, 0x55, 0x66, 0x77};
        fsl_vlan_t vid = 4095;
        fslral_mem_t mem;
        uint32_t key_index;
        fsl_l2_addr_t l2addr;
        int rv;

        /*First determine whether the mac address is in the mac address table, delete it if it exists, and return the table type and index of the mac address*/
```

```
    rv = fsl_l2_addr_get( 0, mac, vid, &l2addr, &mem, &key_index);
    if(FSLRAL_E_NONE == rv)
    {
        fsl_l2_addr_delete( 0, mac, vid, mem, key_index);
    }
    return 0;
}
```

## 6.3 Deleting **MAC** address based on **VLAN**

```
int api_l2_addr_delete_by_vlan()
{
    fsl_vlan_t vid = 4095;
    int rv;
    uint32_t flags;
    /*Delete all mac addresses under this vlan domain*/
    rv = fsl_l2_addr_delete_by_vlan(0, vid, flags);

    return 0;
}
```

## 6.4 Deleting **mac** address based on port

```
int api_l2_addr_delete_by_port()
{
    int rv;
    uint32_t flags;
    fsl_port_t port = 20;
    fsl_module_t mod;
    /*Delete all mac addresses under this port*/
```

```
    rv = fsl_l2_addr_delete_by_port( 0, mod, port, flags);
    return 0;
}
```

## 6.5 Create a multicast group **id**

Create a multicast group id and add multicast members.

```
int api_mcast_create()
{
    int group_id = 4094;
    fsl_pbmp_t pbmp;
    int rv;
    memset(&pbmp, 0, sizeof(fsl_pbmp_t));

    /*Set multicast group members*/
    pbmp.pbits[0] = 0xf;

    /* Initialize global variables */
    fsl_mcast_init(0);
    rv = fsl_mcast_create(0, group_id, pbmp);
    return 0;
}
```

## 6.6 Deleting a multicast group **id**

Delete a multicast group ID and clear multicast members.

```
int api_mcast_delete()
{
    int group_id = 4094;
    int rv;
```

```
    rv = fsl_mcast_delete( 0, group_id);


    return 0;
}
```

## 6.7 Add a multicast **mac** address

```
int api_mcast_addr_add()
{
    sal_mac_addr_t mac_address = {0x00, 0x11, 0x44, 0x55, 0x66, 0x77};
    fsl_vlan_t vid = 4095;
    fsl_mcast_addr_t mcaddr;
    int group_id = 4094;


    /*Initialize multicast address structure*/
    fsl_mcast_addr_t_init(&mcaddr, mac_address, vid);
    mcaddr.group = group_id;
    fsl_mcast_addr_add( 0, &mcaddr);


    return 0;
}
```

### 6.8 Deleting a multicast group port member

```
int api_mcast_bitmap_del()
{
    int group_id = 4094;
    fsl_pbmp_t pbmp;
    memset(&pbmp, 0, sizeof(fsl_pbmp_t));
```

```
        /*Delete members 0 and 1 in multicast id 4094*/
        pbmp.pbits[0] = 0x3;
        fsl_mcast_bitmap_del( 0, group_id, pbmp);

        return 0;
}
```

## 6.9 Delete a multicast **mac** address

```
int api_mcast_addr_remove()
{
        sal_mac_addr_t mac_address = {0x00, 0x11, 0x44, 0x55, 0x66, 0x77};
        fsl_vlan_t vid = 4095;
        int rv;
        /*Delete a multicast mac address*/
        rv = fsl_mcast_addr_remove( 0, mac_address, vid);
        return 0;
}
```

# 7 Vlan

introduce

VLAN (Virtual Local Area Network) is a communication technology that logically divides a physical LAN into multiple broadcast domains (multiple VLANs). Hosts within a VLAN can communicate directly, but VLANs cannot communicate directly with each other, thus limiting broadcast messages to one VLAN. Since VLANs cannot directly communicate with each other, network security is improved. The FSL91030M chip supports VLAN-based forwarding and defines the attributes of each VLAN, including VLAN port members, TRUNK members, blacklist and whitelist modes, priorities, etc. A series of behaviors within the VLAN are determined by creating a VLAN domain.

Purpose

1. Add\delete VLAN domain

2. Add attributes to the VLAN domain

For example: create a VLAN domain vlan=4095, add port members 0 1 2 3 pbmp=0xf to it, then the slave port with vlan=4095 Ordinary messages coming in from port 1 will be forwarded out from ports 0, 2, and 3.

## 7.1 Create a **VLAN** domain

**int api_vlan_create()**

{

fsl_vlan_t vid = 4095;

/*Create a **VLAN domain*/**

fsl_vlan_create( 0, vid);

return 0;

}

## 7.2 Deleting a **VLAN** domain

**int api_vlan_destroy()**

{

fsl_vlan_t vid = 4095;

**/* Delete a vlan domain*/**

fsl_vlan_destroy( 0, vid);

```
    return 0;
}
```

## 7.3 Delete all **VLAN** domains

```
int api_vlan_destroy_all()
{
    int rv;
    /*Delete all vlan domains*/
    rv = fsl_vlan_destroy_all(0);
}
```

## 7.4 Adding port members based on **VLAN**

```
int api_vlan_port_add()
{
    fsl_vlan_t vid = 4095;
    fsl_pbmp_t pbmp, ubmp, lbmp;
    memset(&pbmp, 0, sizeof(fsl_pbmp_t));
    memset(&ubmp, 0, sizeof(fsl_pbmp_t));
    memset(&lbmp, 0, sizeof(fsl_pbmp_t));
    pbmp.pbits[0] = 0xff;
    ubmp.pbits[0] = 0xf;
    lbmp.pbits[0] = 0x3;

    fsl_vlan_port_add( 0, vid, pbmp, ubmp, lbmp);

    return 0;
}
```

## 7.5 Deleting port members based on **VLAN**

```
int api_vlan_port_remove()
{
    fsl_vlan_t vid = 4095;
    fsl_pbmp_t pbmp, lbmp;
    memset(&pbmp, 0, sizeof(fsl_pbmp_t));
    memset(&lbmp, 0, sizeof(fsl_pbmp_t));

    /*Members to be removed*/
    pbmp.pbits[0] = 0xf;
    lbmp.pbits[0] = 0x1;
    fsl_vlan_port_remove( 0, vid, pbmp, lbmp);
    return 0;
}
```

## 7.6 Set the inbound port **stp** status

```
int api_ingress_stp_state_set()
{
    /*gport indicates a local port*/
    fsl_port_t gport = 0x000008;
    fsl_vlan_t vid = 4095;
    int stp_state = FSL_STP_BLOCKING;

    /* Enable inbound stp and erps check*/
    fsl_ingress_stp_erps_enable_set(0, gport, fslPortControlStpChEn, 1);
    fsl_ingress_stp_erps_enable_set(0, gport, fslPortControlerpsLkpEn, 1);

    /*Set the inbound direction stpid*/
```

```
    fsl_vlan_control_set(0, fslVlanStpId, vid, 63);

    /*Set port stp status*/
    fsl_ingress_port_stp_set(0, gport, vid, stp_state);

    /*gport indicates a trunk id=1, port members are assumed to be 0 1 2*/
    gport = 0xc000001;

    /* Enable inbound stp and erps check*/
    fsl_ingress_stp_erps_enable_set(0, gport, fslPortControlStpChEn, 1);
    fsl_ingress_stp_erps_enable_set(0, gport, fslPortControlerpsLkpEn, 1);

    /*Set the inbound direction stpid*/
    fsl_vlan_control_set(0, fslVlanStpId, vid, 63);

    /* Set port stp status */
    stp_state = FSL_STP_LEARNING;
    fsl_ingress_port_stp_set(0, gport, vid, stp_state);

    return 0;
}
```

## 7.7 Get the **STP** status of the incoming port

```
int api_ingress_stp_state_get()
{
    fsl_port_t gport = 0x000008;
```

```
    fsl_vlan_t vid = 4095;
    int stp_state;
    /*Get port stp status*/
    fsl_ingress_port_stp_get(0, gport, vid, &stp_state);
    gport = 0xc000001;
    fsl_ingress_port_stp_get(0, gport, vid, &stp_state);
    return 0;
}
```

## 7.8 Set the outbound port **stp** status

```
int api_egress_stp_state_set()
{
    fsl_port_t gport = 0x000004;
    fsl_vlan_t vid = 4095;
    int stp_state = FSL_STP_LEARNING;

    /* Enable outbound stp and erps check*/
    fsl_egress_stp_erps_enable_set(0, gport, fslPortControlOutStpChkEn, 1);
    fsl_egress_stp_erps_enable_set(0, gport, fslPortControlEerpsLkEn, 1);

    /*Set the outgoing direction stpid*/
    fsl_vlan_control_set(0, fslVlanOutStpId, vid, 127);

    /* Set port stp status */
    fsl_egress_port_stp_set(0, gport, vid, stp_state);
    gport = 0xc000002;
    stp_state = FSL_STP_BLOCKING;
```

```
    fsl_egress_stp_erps_enable_set(0, gport, fslPortControlOutStpChkEn, 1);
    fsl_egress_stp_erps_enable_set(0, gport, fslPortControlEerpsLkEn, 1);
    fsl_egress_port_stp_set(0, gport, vid, stp_state);

    return 0;
}
```

### 7.9 Get **the stp** status of the outbound port

```
int api_egress_stp_state_get()
{
    fsl_port_t gport = 0x000004;
    fsl_vlan_t vid = 4095;
    int stp_state;

    /*Get port stp status*/
    fsl_egress_port_stp_get(0, gport, vid, &stp_state);
    gport = 0xc000002;
    fsl_egress_port_stp_get(0, gport, vid, &stp_state);

    return 0;
}
```

## 7.10 Add port **tpid** type

```
int api_port_tpid_add()
{
    fsl_port_t port = 1;
    uint16_t tpid = 0x88a8;
```

```
    /*tpid initialization, put it in the initialization code*/
    fsl_port_tpid_init(0);
    fsl_port_tpid_add( 0, port, tpid);
    return 0;
}
```

## 7.11 Delete port **tpid** type

```
int api_port_tpid_delete()
{
    fsl_port_t port = 1;
    uint16_t tpid = 0x88a8;

    /* tpid initialization, put it in the initialization code*/
    fsl_port_tpid_init(0);
    fsl_port_tpid_delete( 0, port, tpid);

    return 0;
}
```

## 7.12 Set port **tpid** type

```
int api_port_tpid_set()
{
    fsl_port_t port = 1;
    uint16_t tpid = 0x88a8;

    // tpid initialization, put it in the initialization code
    fsl_port_tpid_init(0);
    fsl_port_tpid_set( 0, port, tpid);
```

```
    return 0;
}
```

### 7.13 Set the erps ring network protection for the incoming port

In this example, the inbound member port pbmp = 0x3 is set for VLAN = 4095. In the VLAN domain of VLAN = 4095, the port Port gport=4 opens the erps protection enable, which is not in the protection member port, so the message coming from port 4 will be discarded.

```
int api_vlan_port_ingress_erps_set()
{
    fsl_vlan_t vid = 4095;
    fsl_pbmp_t pbmp;
    fsl_pbmp_t lbmp;

    /*gport indicates a normal local port*/
    gport=0x0000004
    memset(&pbmp, 0, sizeof(fsl_pbmp_t));
    memset(&lbmp, 0, sizeof(fsl_pbmp_t));

    /* Enable the inbound erps check*/
    fsl_ingress_stp_erps_enable_set(0, gport, fslPortControlerpsLkpEn, 1);

    /*Only ports 0 and 1 are allowed to pass in the inbound direction*/
    pbmp.pbits[0] = 0x3;
    fsl_vlan_control_set( 0, fslVlanErpsId, vid, 2);
    fsl_vlan_port_ingress_erps_set( 0, vid, pbmp, lbmp);

    return 0;
}
```

## **7.14** Set **the erps** ring network protection for the outgoing port .

In the example, the outbound member port pbmp = 0xc of the ring network protection is set for vlan = 4095. In the vlan domain of vlan = 4095,
The erps protection is enabled for the outgoing port gport=2, which is in the protection member port, so the message going out from port 2 will be forwarded normally.

```
int api_vlan_port_egress_erps_set()
{
    fsl_vlan_t vid = 4095;
    fsl_pbmp_t pbmp;
    fsl_pbmp_t lbmp;
   gport = 0x0000004;
    memset(&pbmp, 0, sizeof(fsl_pbmp_t));
    memset(&lbmp, 0, sizeof(fsl_pbmp_t));

   /* Enable departure eprs check*/
   fsl_egress_stp_erps_enable_set(0, gport, fslPortControlEerpsLkEn, 1);

    /*Only ports 2 and 3 are allowed to pass in the outbound direction*/
    pbmp.pbits[0] = 0xc;
    fsl_vlan_control_set( 0, fslVlanOutErpsId, vid, 3);
    fsl_vlan_port_egress_erps_set( 0, vid, pbmp, lbmp);
    return 0;
}
```

# 8 Field Processor

The field processor can classify the various protocol fields of the message, and can also classify the user-defined protocol fields.
Based on these classifications, different actions can be taken, such as dropping the message, sending the packet to the central processor, modifying the VLAN field, etc.

Each Field Processor entry is assigned to a specific group. Each group is able to match data based on a specific set of qualifying attributes.
Each group can contain multiple entries, each entry supports matching of multiple protocol fields and specifies multiple different operation behaviors.

1. Create a field group with limited attributes

ÿ fsl_field_group_create_mode_id(int unit,fsl_field_qset_t qset,int pri

ÿ fsl_field_group_mode_t mode,uint16_t entry_num

ÿ fsl_field_key_tp_t key_tp, fsl_field_group_t group)

Qset specifies the stage that this group is used for, only supports:

ÿ fslFieldQualifyStageIngress(entry acl)

ÿ fslFieldQualifyStageEgress(egress acl)

ÿ fslFieldQualifyStageLooku (inbound vlan transformation)

ÿ fslFieldQualifyStageLookupEgress (egress vlan transformation).

Key_Tp specifies the key type supported by this group. Different key types support different matching protocol fields.

Mode specifies the mode of this group. You can configure the number of matching protocol fields supported by this group.

Table 8-1 Protocol fields supported by different key_tp and mode in the ingress ACL module

| Mode / Key tp | ModeSingle | ModeDouble | ModeTriple | ModeQuad |
|---|---|---|---|---|
| KEY_TP_0 | InPorts, SrcMac, DstMac... | EtherType , Stag, Ctag... | — | — |
| KEY_TP_1 | InPorts, SrcIpÿDstIpÿ L4SrcPort , L4DstPort... | TcpControl... | — | — |
| KEY_TP_2 | InPorts, DstIp6... | SrcIp6ÿTos... | Stag, Ctag, DstMac, EtherType... | L4SrcPort, L4DstPort, TcpControl... |
| KEY_TP_3 | InPorts, SrcMac, | L4SrcPort, L4DstPort, | DstIp6ÿTtl... | SrcIp6, TcpControl... |

| | | | | |
|---|---|---|---|---|
| | DstMac, EtherType ... | SrcIp, DstIp... | | |

2. Create entry

fsl_field_entry_create_id(int unit,fsl_field_group_t group,fsl_field_entry_t entry)

3. Configure the matching protocol field

For example, match the DstIp field of the message:

fsl_field_qualify_DstIp(int unit, fsl_field_entry_t entry, fsl_ip_t data, fsl_ip_t mask)

4. Specify the action after matching

fsl_field_action_add(int unit, fsl_field_entry_t entry, fsl_field_action_t action,

uint32_t param0, uint32_t param1)

5. Write hardware table entries

fsl_field_entry_create_id(int unit,fsl_field_group_t group,fsl_field_entry_t entry)

## **8.1** Calling Example

Taking the ingress ACL as an example, the flow entering port 1 matches the DstIp field, and the behavior after matching is redirection to port 2 for output.

```
int api_fp_test(int unit)
{
    int                     rv = 0;
    fsl_field_group_t       group_id;
    fsl_field_entry_t       entry_id;
    fsl_field_qualify_t     stage;
    fsl_field_key_tp_t      key_tp;
    fsl_field_group_mode_t mode;
    uint16_t                entry_num;
    fsl_field_qset_t        qset;
    fsl_port_t              import;
    fsl_port_t              redirect_port;
```

```
fsl_port_control_t                  type;
fsl_ip_t                            dip;
fsl_ip_t                            dip_mask;
group_id            = 1;
stage               = fslFieldQualifyStageIngress;
key_tp              = _FSL_FIELD_KEY_TP_1;
mode                = fslFieldGroupModeDouble;
entry_num = 100;
Import              = 1;
redirect_port = 2;
type                = fslPortControliAcl0LkpVld;


/* Enable the inbound ACL query function based on the inbound port*/
rv = fsl_port_control_set(unit, inport, type, 1);
        If (rv) {
                return rv;
        }


/*fp initialization, needs to be called once first*/
rv = fsl_field_init(unit);
if (rv) {
        return rv;
}


sal_memset(&qset, 0, sizeof(fsl_field_qset_t));
FSL_FIELD_QSET_ADD(qset, stage);
```

```
/*Create a group, you need to specify stage, key_tp, mode, and number of entries. Group ID is globally unique*/
rv = fsl_field_group_create_mode_id(unit, qset, 0, mode, entry_num, key_tp, group_id);

if (rv) {
        return rv;
}


entry_id = 1;
        /*Create an entry. Before creating an entry, make sure the group has been created and add a matching item. Entry ID is globally unique*/
rv = fsl_field_entry_create_id(unit, group_id, entry_id);
if (rv) {
        return rv;
}
 dip                    = 0xc0a80001;
 dip_mask               = 0xffffff00;


/*This entry matches the destination ipv4 address*/
rv = fsl_field_qualify_DstIp(unit, entry_id, dip, dip_mask);
if (rv) {
        return rv;
}


/*Specify the matching behavior as redirection*/
rv = fsl_field_action_add(unit, entry_id, fslFieldActionRedirectPort, redirect_port,
0);
if (rv) {
        return rv;
```

```
        }

        /*Write hardware table entry*/

        rv = fsl_field_entry_install(unit, entry_id);

        if (rv) {

                return rv;

        }

        return rv;

}
```

# 9 Trunking (Link Aggregation)

Trunk (also known as port bundling, link aggregation) is a method of bundling many Ethernet ports together to form a trunk. Trunk is considered a logical link and is very useful when the switch needs high bandwidth. Trunk has many valuable features, such as co-source and co-destination, link redundancy (if a trunk port fails, the failed port will be removed from the trunk).

## 9.1 Calling Example

Create a trunk group, add multiple ports to the trunk group, and enable failover protection. When the link status of a port member in the trunk group changes from up to down, the traffic of the port is load balanced to other ports in the trunk group* for output; when the link status of the port recovers from down to up, the traffic is restored from the port.

```
int api_trunk_test(int unit)
{
    int                     rv = 0;
    fsl_trunk_t             tid;
    int                     psc;
    fsl_trunk_add_info_t add_info;

    /*trunk initialization, needs to be called first*/
    rv = fsl_trunk_init(unit);
    if (rv) {
        return rv;
    }

    /*Create a trunk group, supporting up to 8 trunk groups, with trunk ID ranging from 0 to 7*/
    tid = 1;
    rv = fsl_trunk_create_id(unit, tid);
    if (rv) {
        return rv;
    }
```

```
/*Set the hash key, for example, in this example, sip+dip counts the hash value*/
psc = FSL_TRUNK_PSC_SRCIP | FSL_TRUNK_PSC_DSTIP;
rv = fsl_trunk_psc_set(unit, tid, psc);
if (rv) {
        return rv;
}
memset(&add_info, 0, sizeof(fsl_trunk_add_info_t));
add_info.num_ports = 4;
add_info.psc                    =psc;
add_info.alg                    = HASH_CRC8_DEFAULT;
add_info.tp[0]                  = 1;
add_info.tp[0]                  = 5;
add_info.tp[0]                  = 6;
add_info.tp[0]                  = 10;

/*Add a port member to the trunk group, and specify the hash key and hash algorithm type. The same port can only belong to one trunk group*/
rv = fsl_trunk_set(unit, tid, &add_info);
if (rv) {
        return rv;
}

/* Enable the trunk group protection function. When the link status of a port member in the trunk group changes from up to down, the traffic on that port will be load balanced.
To other ports in the trunk group for output; when the port link status recovers from down to up, the traffic resumes output from this port. */
rv = fsl_trunk_failover_set(unit, tid, 1);
if (rv) {
        return rv;
```

```
        }

        return rv;

    }
```

# 10 TM

The TM module supports queue scheduling and management, supports port and queue-based traffic shaping, and supports queue-based wred functions.

## 10.1 Port Shaping

Use token bucket to shape the port traffic in single bucket mode.

```
int api_port_traffic_shape_set()
{
    int unit = 0;
    int port = 1; /* outport */
    fsl_shape_mode_t mode = BYTE_MODE;
    int fill_rate = 10000; /* speed limit 10000kb */
    uint16_t burst_size = 100000;
    uint8_t quantum = 2;                /* Bucket depth granularity*/
    fsl_port_traffic_shape_set(unit, port, mode, fill_rate, burst_size, quantum);
    return 0;
}
```

## 10.2 Queue Shaping

Use a token bucket to shape traffic for a specific queue on a port, in dual-bucket mode.

```
int api_port_traffic_shape_set()
{
    int unit = 0;
    int port = 2; /* outport */
    uint8_t queue = 3;
    fsl_shape_mode_t mode = BYTE_MODE;
    int c_fill_rate = 10000; /*guaranteed bandwidth*/
    int p_fill_rate = 20000;
```

```
    uint16_t c_burst_size = 100000;
    uint16_t p_burst_size = 300000;
    uint8_t quantum = 2;                    /* Bucket depth granularity*/
    fsl_queue_traffic_shape_set(unit, port, queue, mode, c_fill_rate,
                                p_fill_rate, c_burst_size, p_burst_size, quantum)
    return 0;
}
```

## **10.3** Queue Scheduling

Set the scheduling mode for the packets entering the queue, supporting SP, WRR and DWRR scheduling.

```
int api_queue_traffic_schedule_set()
{
    int unit = 0;
    int port = 4;
    fsl_queue_schedule_t que_sch_cfg;
    memset(&que_sch_cfg, 0, sizeof(fsl_queue_schedule_t));
    que_sch_cfg.wrr_pri = 3; /*WRR/DWRR scheduling priority, if it is the same as SP scheduling priority, SP scheduling takes precedence*/
    que_sch_cfg.sch_mode = 1; /*DWRR mode*/
    que_sch_cfg.sch_bmp = 0x1c; /*The queues using DWRR mode are 2/3/4*/
    que_sch_cfg.pri2_wrr_weight = 2; /*DWRR scheduling weight*/
    que_sch_cfg.pri3_wrr_weight = 4;
    que_sch_cfg.pri4_wrr_weight = 6;
    fsl_queue_traffic_schedule_set(unit, port, &que_sch_cfg);
    return 0;
}
```

# 11 Pkt DMA

Pkt DMA is used to send and receive packets of the built-in CPU. Before using Pkt DMA, you need to install the network driver xy1000_net.ko.

There are two main functions: as the network driver of the built-in CPU, it sends the packets of the debugging network port to the protocol stack;

The packets are sent to the application layer according to the filtering rules or sent to the protocol stack after stripping off the VLAN.

After the driver is installed, ifconfig -a can show that there are two network devices, eth0 and vir0. The two network devices share the same packet receiving and sending hardware.

eth0 corresponds to the debugging network port, and vir0 corresponds to other in-band panel ports.

The debugging network port is one of the ports on the switch chip panel. After installing the driver, you can view it through the cat command or through the echo command.

Command modification.

```
cat /sys/module/xy1000_net/parameters/net_port
```

```
echo 0 > /sys/module/xy1000_net/parameters/net_port
```

API functions are provided for the filters of non-debugging network ports and the sending and receiving of application layer packets.

## 11.1 Filter Configuration

The filter is used to filter the messages sent from the non-debugging network port to the CPU core (the messages sent from the debugging network port to the CPU do not need to be filtered and are directly sent to the CPU core).

Send protocol stack), the filter can set the message filtering rules and behaviors.

The rule is to perform an AND operation on the corresponding field in the message and the mask specified by the filter. If they are equal to the field specified by the filter,

The message is processed according to the behavior specified by the filter.

There are three types of filter behaviors: no processing (KCOM_DEST_T_NULL), sending to the protocol stack (KCOM_DEST_T_NETIF),

Send to the application layer (KCOM_DEST_T_API).

The filter matches the message according to the priority. The smaller the priority, the higher the level, and the earlier the message is filtered.

```
#define KCOM_FILTER_MAX 256
#define KCOM_DEST_T_NULL                0
#define KCOM_DEST_T_NETIF               1
#define KCOM_DEST_T_API                 2
typedef uint8_t sal_mac_addr_t[6];


void main()
{
```

```
sal_mac_addr_t mac,mac_mask;
uint16_t type,type_mask;
uint16_t dest;
uint8_t ids_num,ids[KCOM_FILTER_MAX];
kcom_filter_t filter;

/*Create filter "dmac_filter", the message offset is 0, the length is 6 (that is, the dmac field of the message):
*   mac =0x11:0x22:0x33:0x44:0x55:0x66
* mask=0xff:0xff:0xff:0xff:0xff:0xff
* Behavior: The message offset is 0, the length is 6, and the mask is ANDed. If it is equal to mac & mask, it is sent to the application layer.
*/
mac={0x11,0x22,0x33,0x44,0x55,0x66};
memset(mac_mask,0xff,6);
dest=KCOM_DEST_T_API;
if(fsl_rx_filter_create(0,6,mac,mac_mask,5,"dmac_filter",strlen("dmac_filter"),de
st))
{
    printf("create filter error\n");
}

/*Create a filter "type_filter" with a message offset of 12 and a length of 2 (i.e. the type field of the message):
*   type = 0x0806
* mask = 0xffff
* Behavior: The message offset is 12, the length is 2, and the mask is ANDed. If it is equal to type & mask, it is sent to the protocol stack.
*/
type = 0x0806;
type_mask = 0xffff
```

```
type = FSLOS_ENDIAN_CPU_TO_BIG_U16(type);
dest=KCOM_DEST_T_NETIF;
if(fsl_rx_filter_create(12,2,&type,&type_mask,4,"type_filter",strlen("type_filter "),dest))
{
    printf("create filter error\n");
}

/*Query the ID and number of created filters*/
if(fsl_rx_filter_list(ids,&ids_num))
{
    printf("get filter list error\n");
}
printf("get filter list[%d]: ",ids_num);
for(i=0;i<ids_num;i++)
{
    printf("%d ",ids[i]);
}
printf("\n");

/*View the filter according to the filter id and print the filter information*/
if(fsl_rx_filter_get(ids[0],&filter))
{
    printf("get filter error\n");
    return;
}
printf("filter.desc: %s\n",filter.desc);
```

```
printf("filter.id: %d\n",filter.id);
printf("filter.priority: %d\n",filter.priority);
printf("filter.dest_type: ",filter.dest_type);
switch(filter.dest_type)
{
    case KCOM_DEST_T_NULL:
        printf("KCOM_DEST_T_NULL\n");
        break;
    case KCOM_DEST_T_NETIF:
        printf("KCOM_DEST_T_NETIF\n");
        break;
    case KCOM_DEST_T_API:
        printf("KCOM_DEST_T_API\n");
        break;
    default:
        break;
}
printf("filter.pkt_data_offset: %d\n",filter.pkt_data_offset);
printf("filter.pkt_data_size: %d\n",filter.pkt_data_size);
printf("filter.data: ");
for(i=0;i<filter.pkt_data_size;i++)
{
    printf("%02x ",filter.data.b[i]);
}
printf("\nfilter.mask: ");
for(i=0;i<filter.pkt_data_size;i++)
{
```

```
            printf("%02x ",filter.mask.b[i]);
        }
        printf("\n");

        /*Delete the filter with id ids[0]*/
        if(ids_num != 0)
        if (fsl_rx_filter_destroy(ids[0]))
        {
            printf("destroy filter error\n");
        }
    }
```

**11.2** Application Layer Packet Transmission

When sending a packet, there is an 8-byte private header at the front of the message, which carries information such as packet length, destination port, packet priority, and whether it passes through the PP module. Information is used internally by the switching chip. The packet header format is as follows:

SOP: { start_ind_cfg[31:0], 8'h0, priority[2:0], ppbypass_ind, dport[5:0], pktlen[13:0] }

start_ind_cfg: configurable, default 32'hc704dd7b

```
void trig_tx(pkt_par *pkt_ptr)
{
    u8i tx_buf[256];
    int unit=0;
    fsl_pkt_t pkt;
    mem_clr(tx_buf, 256); /*packet buffer cleared*/
    pkt.next = NULL;
    pkt.pkt_data = tx_buf;
    pkt.pkt_len = sizeof(tx_buf);
    pkt.port = 0;
    pkt.ppbypass = 1;
```

```
        pkt.priority = 0;
        prepare_pkt(&pkt); /*Set private packet header information*/
        fsl_common_tx(unit, &pkt, NULL); /*send packet*/
    }
```

**11.3** Application Layer Packet Receiving

The received packet contains an 8-byte private header and an 8-byte private tail. The header carries information such as packet length, source port, packet priority, whether it passes through the PP module, etc. The tail carries information such as the time when the packet was received.

SOP: { start_ind_cfg[31:0], 12'h0, sport[5:0], len[13:0]}

EOP: {end_ind_cfg[31:0] , ts_txout[31:0] }

ts_txout[31:0]: The time when the packet is received. ts_txout[31:30] is the last two digits of seconds; ts_txout[29:0] is nanoseconds.

start_ind_cfg: configurable, default 32'hc704dd7b.

end_ind_cfg: Configurable, default 32'h004c1db7.

The return value of the packet receiving function is defined as follows:

```
typedef enum fsl_rx_e {
    FH_RX_INVALID,              /* Invalid packet, terminate packet processing*/
    FH_RX_NOT_HANDLED, /* The packet has not been processed and can continue to be processed by other packet processing functions*/
    FH_RX_HANDLED,              /* The Handled count is increased by 1, and can continue to be processed by other packet processing functions*/
    FH_RX_HANDLED_OWNED /* owned count plus 1, can no longer be processed by other packet processing functions */
} fsl_rx_t;
fsl_rx_t process_pkt(int unit,fsl_pkt_t *pkt,void *pare) /*Packet receiving function*/
{
    int i;
    uint8_t *data=pkt->pkt_data;
    printf("pkt(len=%d):\n",pkt->pkt_len);
    for(i=0;i<pkt->pkt_len;i++)
    {
```

```
            printf("0x%02x ",*(data+i));
        }
        printf("\n");
        return FH_RX_NOT_HANDLED; /*This return value indicates that the packet has not been processed and can be processed by other processing functions*/
}
int trig_rx()
{
        if (fsl_common_rx_start(0)!=FSL_ERR_OK)                    /*Start receiving*/
        {
            printf("start fail\n");
            return -1;
        }
        /*Register the packet receiving processing function. You can register multiple packet receiving processing functions and process the received packets in order of priority. Priority 0 is the lowest*/
        if(fsl_common_rx_register(0,"print",process_pkt,2,NULL,0)!=FSL_ERR_OK)
        {
            printf("register fail\n");
            fsl_common_rx_shutdown(0);
            return -1;
        }
}
```

## **12** Revision Information

| Revision time | Version | describe |
|---|---|---|
| 2021.5.8 | V1.0 | initial version. |
| 2022.12.23 | V1.4 | Content optimization. |