NXP SJA1105 switch driver

Overview

The NXP SJA1105 is a family of 10 SPI-managed automotive switches:

- SJA1105E: First generation, no TTEthernet
- SJA1105T: First generation, TTEthernet
- SJA1105P: Second generation, no TTEthernet, no SGMII
- SJA1105Q: Second generation, TTEthernet, no SGMII
- SJA1105R: Second generation, no TTEthernet, SGMII
- SJA1105S: Second generation, TTEthernet, SGMII
- SJA1110A: Third generation, TTEthernet, SGMII, integrated 100base-T1 and 100base-TX PHYs
- SJA1110B: Third generation, TTEthernet, SGMII, 100base-T1, 100base-TX
- SJA1110C: Third generation, TTEthernet, SGMII, 100base-T1, 100base-TX
- SJA1110D: Third generation, TTEthernet, SGMII, 100base-T1

Being automotive parts, their configuration interface is geared towards set-and-forget use, with minimal dynamic interaction at runtime. They require a static configuration to be composed by software and packed with CRC and table headers, and sent over SPI.

The static configuration is composed of several configuration tables. Each table takes a number of entries. Some configuration tables can be (partially) reconfigured at runtime, some not. Some tables are mandatory, some not:

Table	Mandatory	Reconfigurable
Schedule	no	no
Schedule entry points	if Scheduling	no
VL Lookup	no	no
VL Policing	if VL Lookup	no
VL Forwarding	if VL Lookup	no
L2 Lookup	no	no
L2 Policing	yes	no
VLAN Lookup	yes	yes
L2 Forwarding	yes	partially (fully on P/Q/R/S)
MAC Config	yes	partially (fully on P/Q/R/S)
Schedule Params	if Scheduling	no
Schedule Entry Points Params	if Scheduling	no
VL Forwarding Params	if VL Forwarding	no
L2 Lookup Params	no	partially (fully on P/Q/R/S)
L2 Forwarding Params	yes	no
Clock Sync Params	no	no
AVB Params	no	no
General Params	yes	partially
Retagging	no	yes
xMII Params	yes	no
SGMII	no	yes

Also the configuration is write-only (software cannot read it back from the switch except for very few exceptions).

The driver creates a static configuration at probe time, and keeps it at all times in memory, as a shadow for the hardware state. When required to change a hardware setting, the static configuration is also updated. If that changed setting can be transmitted to the switch through the dynamic reconfiguration interface, it is; otherwise the switch is reset and reprogrammed with the updated static configuration.

Switching features

The driver supports the configuration of L2 forwarding rules in hardware for port bridging. The forwarding, broadcast and flooding domain between ports can be restricted through two methods: either at the L2 forwarding level (isolate one bridge's ports from another's) or at the VLAN port membership level (isolate ports within the same bridge). The final forwarding decision taken by the hardware is a logical AND of these two sets of rules.

The hardware tags all traffic internally with a port-based VLAN (pvid), or it decodes the VLAN information from the 802.1Q tag. Advanced VLAN classification is not possible. Once attributed a VLAN tag, frames are checked against the port's membership rules and dropped at ingress if they don't match any VLAN. This behavior is available when switch ports are enslaved to a bridge with `vlan_filtering 1`.

Normally the hardware is not configurable with respect to VLAN awareness, but by changing what TPID the switch searches 802.1Q tags for, the semantics of a bridge with `vlan_filtering 0` can be kept (accept all traffic, tagged or untagged), and therefore this mode is also supported.

Segregating the switch ports in multiple bridges is supported (e.g. 2 + 2), but all bridges should have the same level of VLAN awareness (either both have `vlan_filtering` 0, or both 1).

Topology and loop detection through STP is supported.

Offloads

Time-aware scheduling

The switch supports a variation of the enhancements for scheduled traffic specified in IEEE 802.1Q-2018 (formerly 802.1Qbv). This means it can be used to ensure deterministic latency for priority traffic that is sent in-band with its gate-open event in the network schedule.

This capability can be managed through the tc-taprio offload ('flags 2'). The difference compared to the software implementation of taprio is that the latter would only be able to shape traffic originated from the CPU, but not autonomously forwarded flows.

The device has 8 traffic classes, and maps incoming frames to one of them based on the VLAN PCP bits (if no VLAN is present, the port-based default is used). As described in the previous sections, depending on the value of `vlan_filtering`, the EtherType recognized by the switch as being VLAN can either be the typical 0x8100 or a custom value used internally by the driver for tagging. Therefore, the switch ignores the VLAN PCP if used in standalone or bridge mode with `vlan_filtering=0`, as it will not recognize the 0x8100 EtherType. In these modes, injecting into a particular TX queue can only be done by the DSA net devices, which populate the PCP field of the tagging header on egress. Using `vlan_filtering=1`, the behavior is the other way around: offloaded flows can be steered to TX queues based on the VLAN PCP, but the DSA net devices are no longer able to do that. To inject frames into a hardware TX queue with VLAN awareness active, it is necessary to create a VLAN sub-interface on the DSA master port, and send normal (0x8100) VLAN-tagged towards the switch, with the VLAN PCP bits set appropriately.

Management traffic (having DMAC 01-80-C2-xx-xx-xx or 01-19-1B-xx-xx-xx) is the notable exception: the switch always treats it with a fixed priority and disregards any VLAN PCP bits even if present. The traffic class for management traffic has a value of 7 (highest priority) at the moment, which is not configurable in the driver.

Below is an example of configuring a 500 us cyclic schedule on egress port `swp5`. The traffic class gate for management traffic (7) is open for 100 us, and the gates for all other traffic classes are open for 400 us:

```
#!/bin/bash

set -e -u -o pipefail

NSEC_PER_SEC="1000000000"

gatemask() {
        local tc_list="$1"
        local mask=0

        for tc in ${tc_list}; do
                mask=$((${mask} | (1 << ${tc})))
        done

        printf "%02x" ${mask}
}

if ! systemctl is-active --quiet ptp4l; then
        echo "Please start the ptp4l service"
        exit
fi

now=$(phc_ctl /dev/ptp1 get | gawk '/clock time is/ { print $5; }')
# Phase-align the base time to the start of the next second.
sec=$(echo "${now}" | gawk -F. '{ print $1; }')
base_time="$(((${sec} + 1) * ${NSEC_PER_SEC}))"

tc qdisc add dev swp5 parent root handle 100 taprio \
        num_tc 8 \
        map 0 1 2 3 5 6 7 \
        queues 1@0 1@1 1@2 1@3 1@4 1@5 1@6 1@7 \
        base-time ${base_time} \
        sched-entry S $(gatemask 7) 100000 \
        sched-entry S $(gatemask "0 1 2 3 4 5 6") 400000 \
        flags 2
```

It is possible to apply the tc-taprio offload on multiple egress ports. There are hardware restrictions related to the fact that no gate event may trigger simultaneously on two ports. The driver checks the consistency of the schedules against this restriction and errors out when appropriate. Schedule analysis is needed to avoid this, which is outside the scope of the document.

Routing actions (redirect, trap, drop)

The switch is able to offload flow-based redirection of packets to a set of destination ports specified by the user. Internally, this is implemented by making use of Virtual Links, a TTEthernet concept.

The driver supports 2 types of keys for Virtual Links:

- VLAN-aware virtual links: these match on destination MAC address, VLAN ID and VLAN PCP.
- VLAN-unaware virtual links: these match on destination MAC address only.

The VLAN awareness state of the bridge (vlan_filtering) cannot be changed while there are virtual link rules installed.

Composing multiple actions inside the same rule is supported. When only routing actions are requested, the driver creates a "non-critical" virtual link. When the action list also contains tc-gate (more details below), the virtual link becomes "time-critical" (draws frame buffers from a reserved memory partition, etc).

The 3 routing actions that are supported are "trap", "drop" and "redirect".

Example 1: send frames received on swp2 with a DA of 42:be:24:9b:76:20 to the CPU and to swp3. This type of key (DA only) when the port's VLAN awareness state is off:

```
tc qdisc add dev swp2 clsact
tc filter add dev swp2 ingress flower skip_sw dst_mac 42:be:24:9b:76:20 \
        action mirred egress redirect dev swp3 \
        action trap
```

Example 2: drop frames received on swp2 with a DA of 42:be:24:9b:76:20, a VID of 100 and a PCP of 0:

```
tc filter add dev swp2 ingress protocol 802.1Q flower skip_sw \
        dst_mac 42:be:24:9b:76:20 vlan_id 100 vlan_prio 0 action drop
```

Time-based ingress policing

The TTEthernet hardware abilities of the switch can be constrained to act similarly to the Per-Stream Filtering and Policing (PSFP) clause specified in IEEE 802.1Q-2018 (formerly 802.1Qci). This means it can be used to perform tight timing-based admission control for up to 1024 flows (identified by a tuple composed of destination MAC address, VLAN ID and VLAN PCP). Packets which are received outside their expected reception window are dropped.

This capability can be managed through the offload of the tc-gate action. As routing actions are intrinsic to virtual links in TTEthernet (which performs explicit routing of time-critical traffic and does not leave that in the hands of the FDB, flooding etc), the tc-gate action may never appear alone when asking sja1105 to offload it. One (or more) redirect or trap actions must also follow along.

Example: create a tc-taprio schedule that is phase-aligned with a tc-gate schedule (the clocks must be synchronized by a 1588 application stack, which is outside the scope of this document). No packet delivered by the sender will be dropped. Note that the reception window is larger than the transmission window (and much more so, in this example) to compensate for the packet propagation delay of the link (which can be determined by the 1588 application stack).

Receiver (sja1105):

```
tc qdisc add dev swp2 clsact
now=$(phc_ctl /dev/ptp1 get | awk '/clock time is/ {print $5}') && \
        sec=$(echo $now | awk -F. '{print $1}') && \
        base_time="$(((sec + 2) * 1000000000))" && \
        echo "base time ${base_time}"
tc filter add dev swp2 ingress flower skip_sw \
        dst_mac 42:be:24:9b:76:20 \
        action gate base-time ${base_time} \
        sched-entry OPEN  60000 -1 -1 \
        sched-entry CLOSE 40000 -1 -1 \
        action trap
```

Sender:

```
now=$(phc_ctl /dev/ptp0 get | awk '/clock time is/ {print $5}') && \
        sec=$(echo $now | awk -F. '{print $1}') && \
        base_time="$(((sec + 2) * 1000000000))" && \
        echo "base time ${base_time}"
tc qdisc add dev eno0 parent root taprio \
        num_tc 8 \
        map 0 1 2 3 4 5 6 7 \
        queues 1@0 1@1 1@2 1@3 1@4 1@5 1@6 1@7 \
        base-time ${base_time} \
        sched-entry S 01  50000 \
        sched-entry S 00  50000 \
        flags 2
```

The engine used to schedule the ingress gate operations is the same that the one used for the tc-taprio offload. Therefore, the restrictions regarding the fact that no two gate actions (either tc-gate or tc-taprio gates) may fire at the same time (during the same 200 ns slot) still apply.

To come in handy, it is possible to share time-triggered virtual links across more than 1 ingress port, via flow blocks. In this case, the restriction of firing at the same time does not apply because there is a single schedule in the system, that of the shared virtual link:

```
tc qdisc add dev swp2 ingress_block 1 clsact
tc qdisc add dev swp3 ingress_block 1 clsact
tc filter add block 1 flower skip_sw dst_mac 42:be:24:9b:76:20 \
        action gate index 2 \
        base-time 0 \
        sched-entry OPEN 50000000 -1 -1 \
        sched-entry CLOSE 50000000 -1 -1 \
        action trap
```

Hardware statistics for each flow are also available ("pkts" counts the number of dropped frames, which is a sum of frames dropped due to timing violations, lack of destination ports and MTU enforcement checks). Byte-level counters are not available.

Limitations

The SJA1105 switch family always performs VLAN processing. When configured as VLAN-unaware, frames carry a different VLAN tag internally, depending on whether the port is standalone or under a VLAN-unaware bridge.

The virtual link keys are always fixed at {MAC DA, VLAN ID, VLAN PCP}, but the driver asks for the VLAN ID and VLAN PCP when the port is under a VLAN-aware bridge. Otherwise, it fills in the VLAN ID and PCP automatically, based on whether the port is standalone or in a VLAN-unaware bridge, and accepts only "VLAN-unaware" tc-flower keys (MAC DA).

The existing tc-flower keys that are offloaded using virtual links are no longer operational after one of the following happens:

- port was standalone and joins a bridge (VLAN-aware or VLAN-unaware)
- port is part of a bridge whose VLAN awareness state changes
- port was part of a bridge and becomes standalone
- port was standalone, but another port joins a VLAN-aware bridge and this changes the global VLAN awareness state of the bridge

The driver cannot veto all these operations, and it cannot update/remove the existing tc-flower filters either. So for proper operation, the tc-flower filters should be installed only after the forwarding configuration of the port has been made, and removed by user space before making any changes to it.

Device Tree bindings and board design

This section references `Documentation/devicetree/bindings/net/dsa/nxp,sja1105.yaml` and aims to showcase some potential switch caveats.

RMII PHY role and out-of-band signaling

In the RMII spec, the 50 MHz clock signals are either driven by the MAC or by an external oscillator (but not by the PHY). But the spec is rather loose and devices go outside it in several ways. Some PHYs go against the spec and may provide an output pin where they source the 50 MHz clock themselves, in an attempt to be helpful. On the other hand, the SJA1105 is only binary configurable - when in the RMII MAC role it will also attempt to drive the clock signal. To prevent this from happening it must be put in RMII PHY role. But doing so has some unintended consequences. In the RMII spec, the PHY can transmit extra out-of-band signals via RXD[1:0]. These are practically some extra code words (/J/ and /K/) sent prior to the preamble of each frame. The MAC does not have this out-of-band signaling mechanism defined by the RMII spec. So when the SJA1105 port is put in PHY role to avoid having 2 drivers on the clock signal, inevitably an RMII PHY-to-PHY connection is created. The SJA1105 emulates a PHY interface fully and generates the /J/ and /K/ symbols prior to frame preambles, which the real PHY is not expected to understand. So the PHY simply encodes the extra symbols received from the SJA1105-as-PHY onto the 100Base-Tx wire. On the other side of the wire, some link partners might discard these extra symbols, while others might choke on them and discard the entire Ethernet frames that follow along. This looks like packet loss with some link partners but not with others. The take-away is that in RMII mode, the SJA1105 must be let to drive the reference clock if connected to a PHY.

RGMII fixed-link and internal delays

As mentioned in the bindings document, the second generation of devices has tunable delay lines as part of the MAC, which can be used to establish the correct RGMII timing budget. When powered up, these can shift the Rx and Tx clocks with a phase difference between 73.8 and 101.7 degrees. The catch is that the delay lines need to lock onto a clock signal with a stable frequency. This means that there must be at least 2 microseconds of silence between the clock at the old vs at the new frequency. Otherwise the lock is lost and the delay lines must be reset (powered down and back up). In RGMII the clock frequency changes with link speed (125 MHz at 1000 Mbps, 25 MHz at 100 Mbps and 2.5 MHz at 10 Mbps), and link speed might change during the AN process. In the situation where the switch port is connected through an RGMII fixed-link to a link partner whose link state life cycle is outside the control of Linux (such as a different SoC), then the delay lines would remain unlocked (and inactive) until there is manual intervention (ifdown/ifup on the switch port). The take-away is that in RGMII mode, the switch's internal delays are only reliable if the link partner never changes link speeds, or if it does, it does so in a way that is coordinated with the switch port (practically, both ends of the fixed-link are under control of the same Linux system). As to why would a fixed-link interface ever change link speeds: there are

Ethernet controllers out there which come out of reset in 100 Mbps mode, and their driver inevitably needs to change the speed and clock frequency if it's required to work at gigabit.

MDIO bus and PHY management

The SJA1105 does not have an MDIO bus and does not perform in-band AN either. Therefore there is no link state notification coming from the switch device. A board would need to hook up the PHYs connected to the switch to any other MDIO bus available to Linux within the system (e.g. to the DSA master's MDIO bus). Link state management then works by the driver manually keeping in sync (over SPI commands) the MAC link speed with the settings negotiated by the PHY.

By comparison, the SJA1110 supports an MDIO slave access point over which its internal 100base-T1 PHYs can be accessed from the host. This is, however, not used by the driver, instead the internal 100base-T1 and 100base-TX PHYs are accessed through SPI commands, modeled in Linux as virtual MDIO buses.

The microcontroller attached to the SJA1110 port 0 also has an MDIO controller operating in master mode, however the driver does not support this either, since the microcontroller gets disabled when the Linux driver operates. Discrete PHYs connected to the switch ports should have their MDIO interface attached to an MDIO controller from the host system and not to the switch, similar to SJA1105.

Port compatibility matrix

The SJA1105 port compatibility matrix is:

Port	SJA1105E/T	SJA1105P/Q	SJA1105R/S
0	xMII	xMII	xMII
1	xMII	xMII	xMII
2	xMII	xMII	xMII
3	xMII	xMII	xMII
4	xMII	xMII	SGMII

The SJA1110 port compatibility matrix is:

Port	SJA1110A	SJA1110B	SJA1110C	SJA1110D
0	RevMII (uC)	RevMII (uC)	RevMII (uC)	RevMII (uC)
1	100base-TX or SGMII	100base-TX	100base-TX	SGMII
2	xMII or SGMII	xMII	xMII	xMII or SGMII
3	xMII or SGMII or 2500base-X	xMII or SGMII or 2500base-X	xMII	SGMII or 2500base-X
4	SGMII or 2500base-X	SGMII or 2500base-X	SGMII or 2500base-X	SGMII or 2500base-X
5	100base-T1	100base-T1	100base-T1	100base-T1
6	100base-T1	100base-T1	100base-T1	100base-T1
7	100base-T1	100base-T1	100base-T1	100base-T1
8	100base-T1	100base-T1	n/a	n/a
9	100base-T1	100base-T1	n/a	n/a
10	100base-T1	n/a	n/a	n/a