AN6048

Daisy Chaining KSZ Ethernet Switch Devices

Authors: Wayne Lu and Jacky Hou
Microchip Technology, Inc.

1.0 INTRODUCTION

The KSZ9897R/S, KSZ9477S, and KSZ9567R/S are highly integrated Layer 2 switches with a maximum of seven ports (five built-in copper ports and two uplink ports). For these devices, the uplink ports are comprised of two RGMII/MII/RMII in the KSZ9897R/KSZ9567R, as well as one RGMII/MII/RMII and one SGMII in the KSZ9897S, KSZ9477S, and KSZ9567S. The KSZ9893R and KSZ9563R are three-port switches with two built-in copper ports and one RGMII/MII/RMII uplink port.

If a single-chip solution does not provide enough ports for a particular design, multiple Microchip KSZ switches can be daisy chained through the RGMII/SGMII interface to expand the number of available Ethernet ports. This application note details how daisy chaining can be accomplished.

1.1 Sections

This application note covers the following topics:

- Section 2.0, Daisy Chaining by RGMII Interface
- Section 3.0, Daisy Chaining by SGMII Interface
- Section 4.0, Management Considerations

1.2 References

The following documents should be referenced when using this application note:

- *KSZ9477S Data Sheet*
- *KSZ9897S Data Sheet*
- *KSZ9567S Data Sheet*
- *KSZ9897R Data Sheet*
- *KSZ9567R Data Sheet*
- *KSZ9893 Data Sheet*
- *KSZ9563R Data Sheet*
- *KSZ9893R Data Sheet*
- Linux® Kernel Documentation

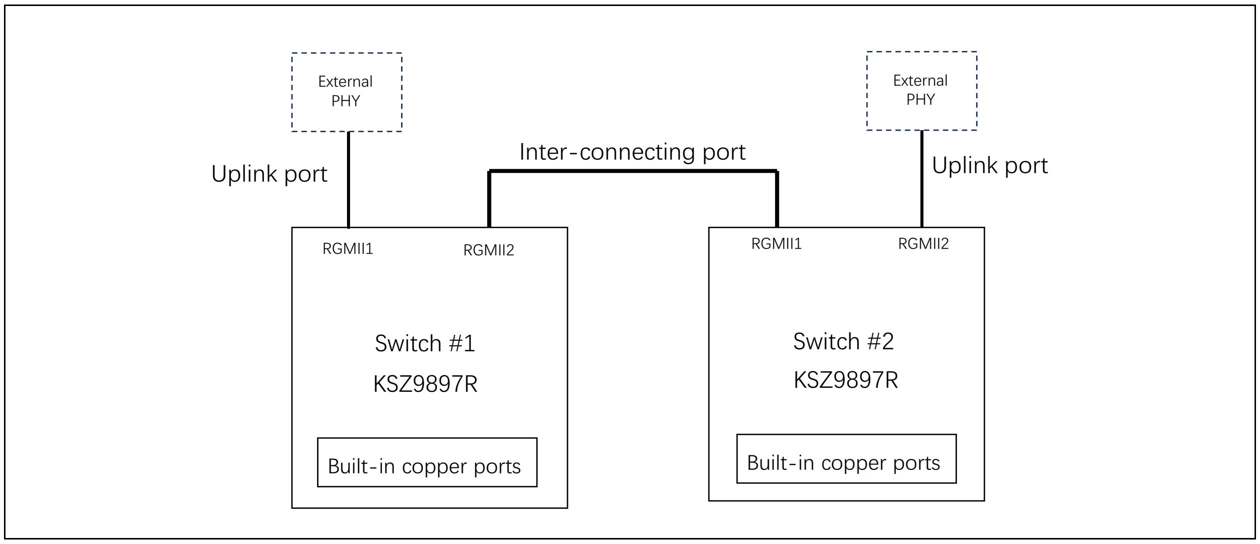
Note: In this document, the terms “CPU” and “host processor” are interchangeable as well as the terms “switch” and “Ethernet switch.”

2.0 DAISY CHAINING BY RGMII INTERFACE

2.1 Switch Connection

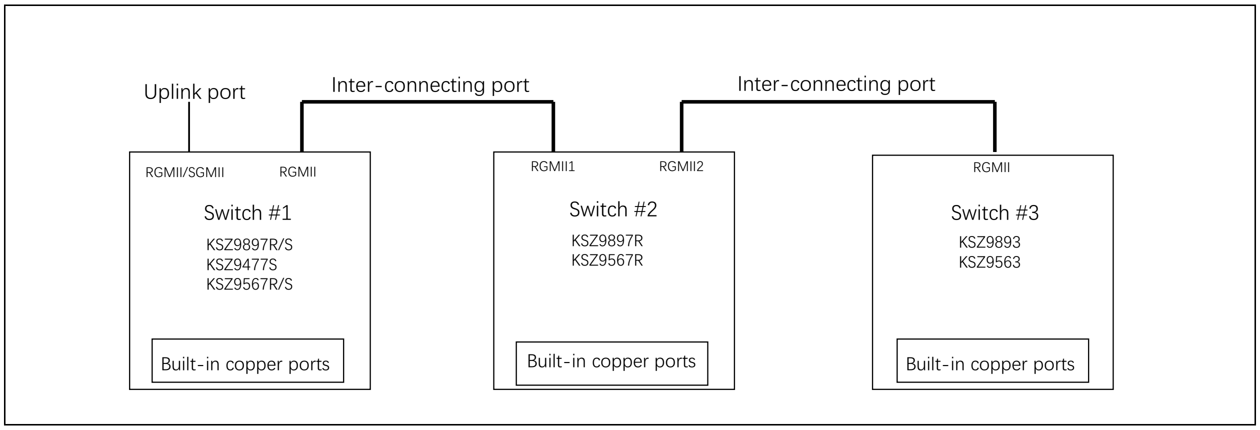
The RGMII interface is supported by the KSZ9897R/S, KSZ9477S, and KSZ9567R/S switches. It is a reduced pin count gigabit Ethernet Media independent interface ideal for connecting the switches in a daisy chain. For example, connecting two KSZ9897R switches through one of the two RGMII interfaces from each switch makes an equivalent 10*1000Base-T + 2*RGMII uplink switch. External PHYs can be attached to the uplink RGMII interfaces to provide additional Ethernet ports. Figure 1 details an example of an RGMII daisy chain between two devices.

FIGURE 1: DAISY CHAINING TWO KSZ9897R SWITCHES THROUGH AN RGMII INTERFACE



It is also possible to daisy chain more switches using the RGMII interface. Figure 2 details an example of an RGMII daisy chain between three devices.

FIGURE 2: DAISY CHAINING THREE KSZ SWITCHES THROUGH AN RGMII INTERFACE

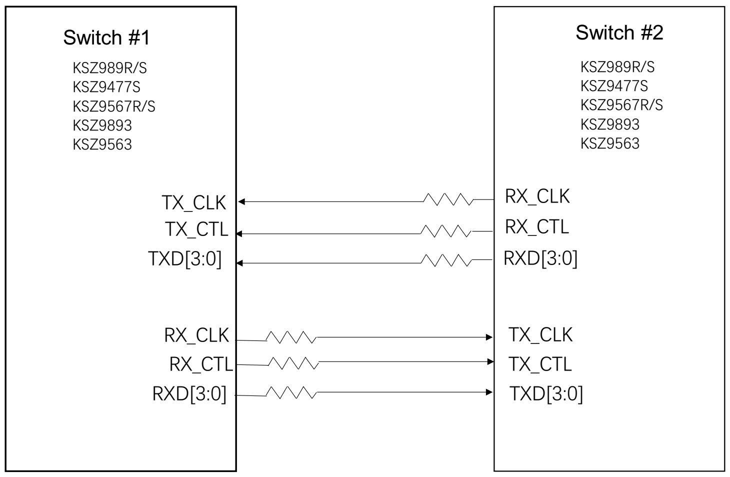


Note that the bandwidth of the interconnecting RGMII links might become the bottleneck of the daisy chained switch's throughput. The maximum traffic rate between the two switches is limited to 1G (the maximum speed of the RGMII interface). However, the local traffic rate between ports on the same switch is not limited.

2.2 RGMII Signal Connection and Switch Pin Strapping

RGMII is a symmetrical interface, meaning there is no difference between the PHY and MAC modes. A TX-to-RX cross connection is required for the switch-to-switch connection. Figure 3 details the signal connection of the RGMII interfaces from one switch to another.

FIGURE 3: RGMII SIGNAL CONNECTION BETWEEN KSZ SWITCHES



For KSZ switches, the signals named RX are output signals from the switch, and the signals named TX are the input signals. Placing a series termination resistor of approximately 22Ω close to the source on each RGMII output signal is recommended. Length matching within the TX signal group and RX signal group is required during layout. A data-to-clock skew of at least 1 ns is also required by the RGMII v2.0 standard. This can be typically achieved by the 1.5 ns internal delay (RGMII-ID) on both egress and ingress sides, which can be enabled individually by the KSZ switches. By default, the egress RGMII-ID is enabled, and the ingress RGMII-ID is disabled for all KSZ switches. This is suitable for the interconnection between the switches because only one RGMII-ID should be enabled on each signal direction, and the default configuration perfectly matches the setup. However, there is an errata on the ingress side of port 6 of KSZ9897R/S and KSZ9477S and KSZ9567R/S switches because an extended setup time is necessary. Adding a PCB trace delay of 1.4 ns (or 8 inches on FR4 PCB) on TXC6 of the KSZ9897 and KSZ9477 is recommended.

Since the RGMII interface on the KSZ switches supports other operating modes, such as MII and RMII, it is necessary to enable RGMII through either strapping pin configuration or register writing. Table 1 shows how to set up the RGMII interface in RGMII 1G mode.

TABLE 1: RGMII INTERFACE IN RGMII 1G MODE

SKU	Strapping Pin Configuration	Register Configuration
KSZ9897R, KSZ9567R	RXD6(7)_3 = 0 RXD6(7)_2 = 0 RXD6(7)_0 = 0	0x6(7)301 bit 1 = 0 0x6(7)301 bit 0 = 0 0x6(7)301 bit 6 = 0

KSZ9477S, KSZ9897S, and KSZ9567S	RXD6_3 =0
	RXD6_2 = 0
	RXD6_0 = 0
		0x6301 bit 1 = 0
		0x6301 bit 0 = 0
		0x6301 bit 6 = 0

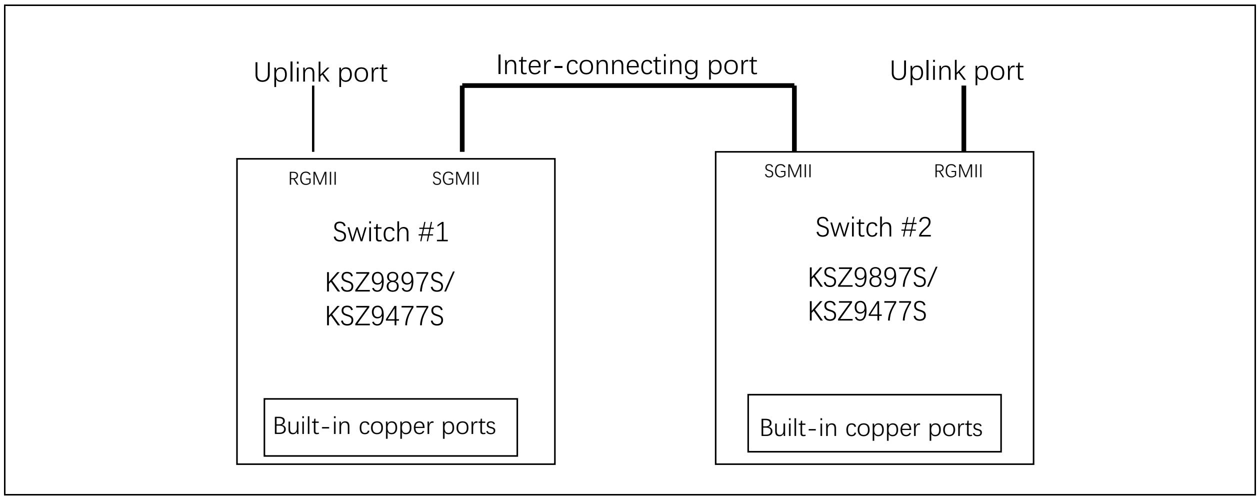
KSZ9893, KSZ9563	RXD3 = 1
	RXD2 = 1
	LED2_1 = 0
		0x3301 bit 1 = 1
		0x3301 bit 0 = 1
		0x3301 bit 6 = 0

3.0 DAISY CHAINING BY SGMII INTERFACE

3.1 Switches Connection

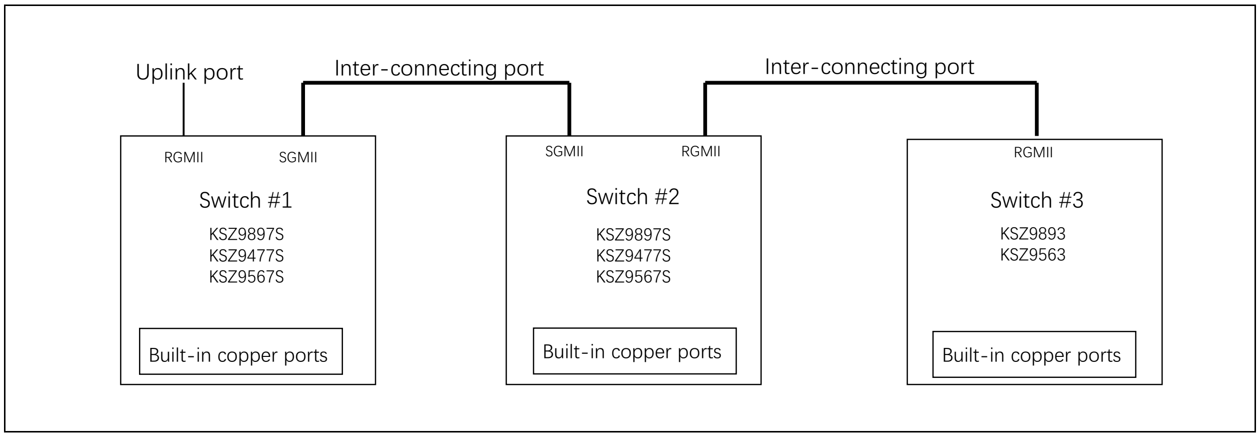
Supported by the KSZ9897S, LSZ9477S, and KSZ9567S, the SGMII is a serial differential gigabit Ethernet interface that is also ideal for interconnecting multiple switches. For example, connecting two KSZ9897S (or KSZ9477S or KSZ9567S) switches through the SGMII interface makes an equivalent 10*copper + 2*RGMII uplink switch. An external PHY can be attached to the uplink RGMII interface for additional Ethernet ports. Figure 4 details an example of an SGMII daisy chain between two devices.

FIGURE 4: TWO INTERCONNECTED KSZ9897S OR KSZ9477S SWITCHES THROUGH SGMII INTERFACE



A combination of SGMII and RGMII can be used in a daisy chain. Figure 5 details an example of daisy chaining three switches. The first two switches are interconnected through the SGMII while the third switch is connected to the daisy chain through the RGMII interface.

FIGURE 5: DAISY CHAINING WITH A COMBINATION OF SGMII AND RGMII INTERFACES



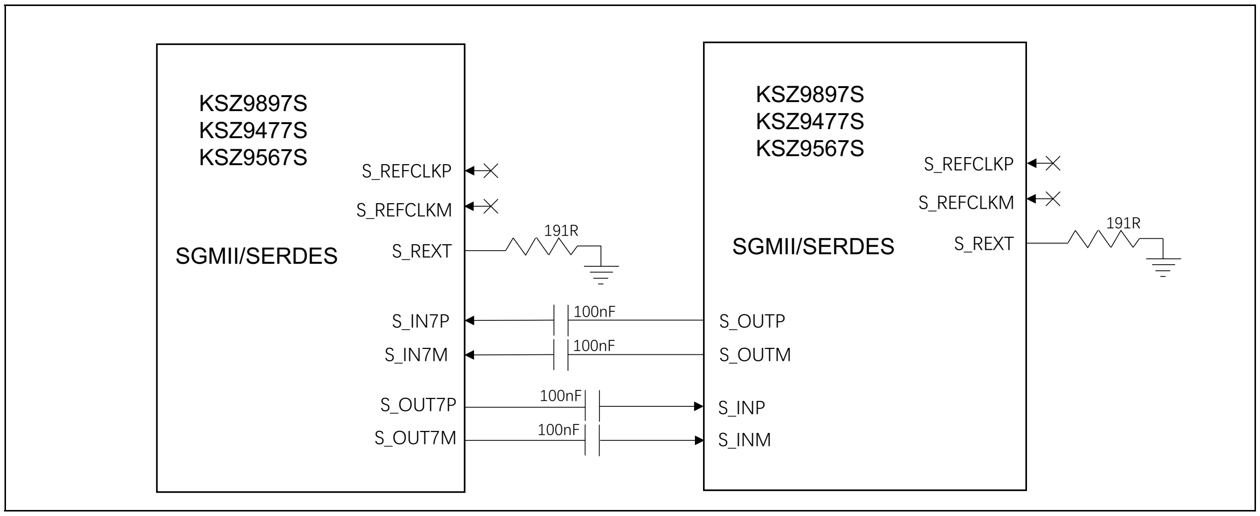
Note that the bandwidth of the interconnecting links, the RGMII or SGMII, in Figure 5 becomes the bottleneck of the daisy chained switch. The maximum traffic rate between switches is limited to 1G. However, the local traffic rate between ports on the same switch is not limited.

3.2 SGMII Signal Connection and Register Configuration

SGMII is also a symmetrical interface, meaning there is no difference between the PHY and MAC modes. A TX-to-RX cross connection is required for the switch-to-switch connection. Figure 6 shows the signal mapping of the SGMII interface from one switch to another.

The SGMII interface has one receive differential pair and one transmit differential pair for sending and receiving data, and control at a serial bit rate of 1.25G baud. The SGMII block recovers the clock from the incoming data, eliminating the need for a separate SGMII clock. Likewise, no output SGMII clock is provided, with the expectation that the connected device will also recover the clock from the receive data.

FIGURE 6: SGMII SIGNAL CONNECTION BETWEEN KSZ SWITCHES



The SGMII signals can be routed on any PCB trace layer with the following constraints:

- The TX output signals in a pair should have matching electrical length.
- The RX output signals in a pair should have matching electrical length.
- SGMII TX and RX pairs must be routed as 100Ω differential traces with ground plane as reference.
- Keep differential pair traces on the same layer of the PCB to minimize impedance discontinuities.
- 100 nF AC coupling capacitors can be omitted if the switches are on the same PCB. AC coupling capacitors are recommended if the switches are on different PCB linked through connectors.

The SGMII interface on the KSZ switches supports different operating modes. The SGMII mode is one of the supported modes typically used for connecting the switch to a trispeed PHY. The 10M, 100, and 1G speeds can be supported in this mode. However, in this daisy chain application, the highest possible speed is always preferred, so the 1G speed should be fixed. Configuring the port at 10M or 100M speed is not needed. Since the speed is fixed at 1G, auto-negotiation on the SGMII interface is not necessary. The following is the required configuration for the SGMII mode: SGMII register 0x1F0000 = 0x0140.

The above configuration means writing 0x0140 to the indirect SGMII register 0x1F0000. Writing 0x0140 disables auto-negotiation and sets the speed at 1G in Full-duplex mode. Since the SGMII mode is the default mode of the SGMII interface, an extra configuration is not necessary.

An alternative is configuring the SGMII interface in 1000Base-X Serdes mode. This mode of operation is normally used for the switch to support 1000Base-X fiber optic ports with external SFP modules attached. This mode is also perfect for the daisy chain interconnecting link. 1000Base-X Serdes mode can be enabled using the following register configurations:

- SGMII register 0x1F8001 = 0x0019. // enable Serdes mode
- SGMII register 0x1F0000 = 0x0140. // disable auto-negotiation

1000-Base-X auto-negotiation can also be enabled with the following configurations:

- SGMII register 0x1F8001 = 0x0019. // enable Serdes mode
- SGMII register 0x1F0004 = 0x01A0. // auto-negotiation control word: full duplex, symmetrical pause
- SGMII register 0x1F0000 = 0x1340. // enable auto-negotiation and restart auto-negotiation

4.0 MANAGEMENT CONSIDERATIONS

4.1 Management Interface

All the KSZ switches mentioned in this application note support either Unmanaged or Managed operation modes. The daisy chained switch as a whole also supports Unmanaged or Managed modes. In Managed mode, an external CPU will be attached to the switch through one of the management interfaces. The CPU also configures and monitors the switch through the switch registers. Although the switch can start operating based on the basic strapping pin configurations without any register writes in Unmanaged mode, it is still recommended to attach an external CPU to the management interface of the switches. Some switch registers need to be overwritten for performance optimization or bug fixing.

The switch provides a selection of three (3) different management interfaces: SPI, I^{2}C, and MIIM. One of these management interfaces is enabled through strapping pins. MIIM is not recommended because it only provides register access to the built-in copper PHY. Microchip recommends that the SPI interface concerning the switch driver initialization should be completed before the Linux® network protocol starts to poll the KSZ switch.

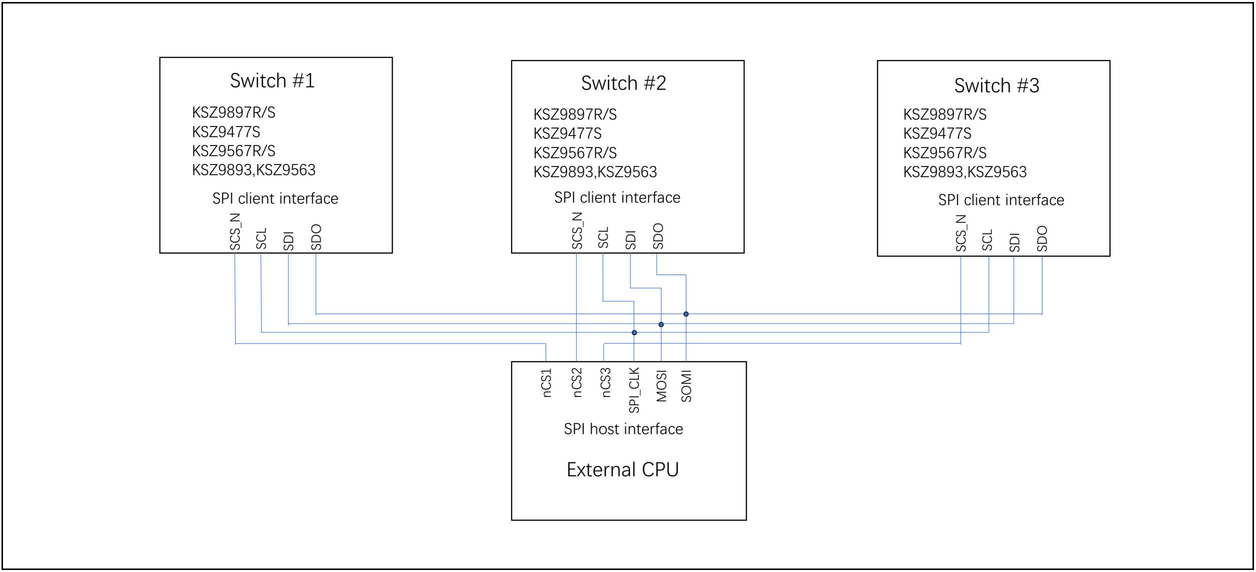
The succeeding sections provide some recommendations for using the SPI or I^{2}C interface.

4.1.1 SPI INTERFACE

An SPI interface consists of four (4) signals. The host device (external CPU) supplies the clock (SCL) and chip select (nCS). MOSI signal is output from the host device (external CPU) and input into the client device (KSZ switches). MISO signal is output from the client device (KSZ switches) and input into the host device (external CPU).

There are multiple KSZ switches in the daisy chain attached to the same CPU. The external CPU addresses different switches using separate chip select signals. In cases wherein the CPU does not have enough chip select signals, external logic might be needed to expand more chip select signals through, for example, GPIO pins. Figure 7 shows a block diagram of an external CPU managing three KSZ switches through SPI interface with different chip select signals from the CPU.

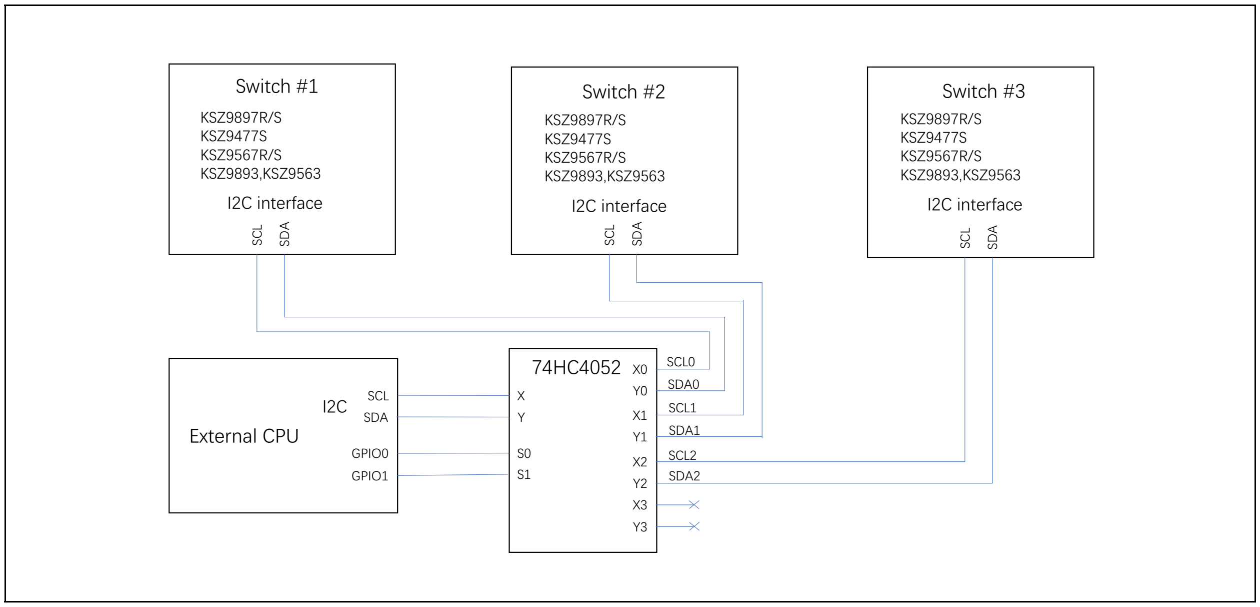
FIGURE 7: USING SPI AS THE MANAGEMENT INTERFACE FOR THE SWITCHES IN THE DAISY CHAIN



4.1.2 I^{2}C INTERFACE

An I^{2}C interface is a two-wire interface with a clock signal (SCL) driven by the host (external CPU) and a bi-directional data signal (SDA) driven by the host (external CPU) and client (KSZ switches) at different times. Although the I^{2}C is designed to support multiple clients on the same bus distinguished by a 7-bit I^{2}C device address, all the KSZ switches mentioned in this application note have the same fixed device address–1011_111. Therefore, an external I^{2}C MUX is needed to select only one switch at a time. Figure 8 shows the block diagram of three KSZ switches managed by an external CPU through an I^{2}C interface.

FIGURE 8: USING I^{2}C AS MANAGEMENT INTERFACE FOR SWITCHES IN A DAISY CHAIN



In Figure 8, a 74HC4052 is used as an I^{2}C MUX. For example, when the external CPU decides to access switch #3, it sets GPIO0 and GPIO1 to be "1,0," so that 74HC4052 connects X to X2 and Y to Y2 and therefore the I^{2}C on the external CPU will be connected to switch #3 only.

4.2 Software Design Consideration when Connecting Multiple KSZ Ethernet Switch Parts

Some management features only require the CPU to configure certain switch registers. They do not need the CPU to generate and exchange Ethernet frames with other devices. For example, VLAN, QoS, ACL, and rate limiting can be statically configured by the external CPU through certain registers. However, for some dynamic protocols such as spanning tree, PTP requires the software running on the external CPU to exchange protocol frames with other devices. In addition, the CPU should know the source port where the protocol frame was received and specify the destination port when it sends a software-generated frame out of the switch. This can be done by connecting one of the switch ports to an Ethernet port on the external CPU and enabling tail tagging on that switch port that connects to the CPU.

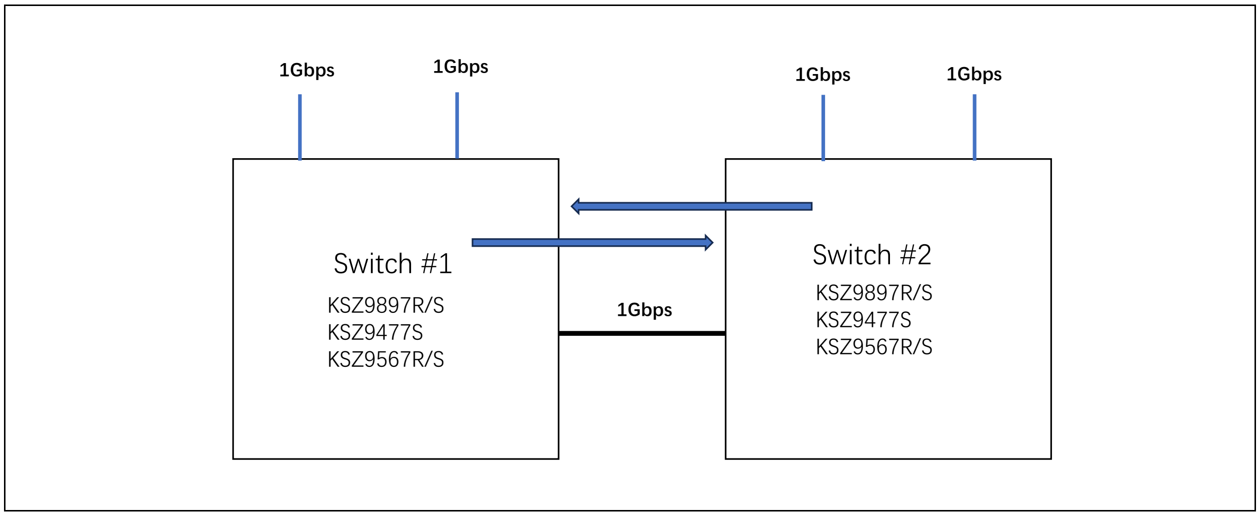
Tail tagging is a method to communicate ingress and egress port information between the host processor and the KSZ switch. When the switch forwards a received packet to the host port, one tail tagging byte is added to the packet by the switch to tell the host processor which port received the packet. In the opposite direction, the host processor must add two tail-tagged bytes to each packet that it sends to the switch to indicate the intended egress ports. When multiple priority queues are enabled, the tail tag is also used to specify the priority queue.

Tail tagging usually works well in single-switch applications. However, in daisy chain cases, the process is more complicated as the software needs to manage the frames received or destined to ports on different switches. The succeeding sections discuss suitable approaches.

4.2.1 WITHOUT USING A HOST PROCESSOR PORT

Without the switch software running the network protocols, two switch parts can be connected through a strap-in configuration to create a higher port count unmanaged switch that targets frame forwarding. Without the switch software running, the bandwidth remains an issue when connecting multiple switch devices. In Figure 9, the bandwidth in the interconnected ports or the aggregate traffic rate between the two switches is 1 Gb/second. If two end devices are communicating across the ports that are located on different switches and their data rate is 1 Gb/second, the bottleneck is on the interconnected port since the maximum bandwidth is 1 Gbps. In this example, each end device can only reach a 500 Mbps packet forwarding throughput, which is less than 1 Gbps, assuming that packet forwarding is being scheduled based on the round robin scheduling algorithm. Therefore, the wire speed of 1 Gbps between the two devices across the switch #1 and #2 in this example is not achievable.

FIGURE 9: PACKET FLOW BETWEEN TWO INTERCONNECTED KSZ SWITCHES



4.2.2 USING HOST PROCESSOR PORTS

There are two use cases to be considered when connecting two KSZ switch parts to increase switch port count:

1. Case 1: A host processor has only one MAC port.
2. Case 2: A host processor has two individual MAC ports.

Since a switch driver is required with respect to software implementation, the next sections are based on the main line Linux® kernel DSA driver.

4.2.3 USING A SINGLE-HOST PROCESSOR MAC PORT

The Microchip proprietary tail tagging mechanism cannot support this use case. This is because tail tagging is required to indicate and direct the received or transmitted port ID between the interconnected KSZ switch ports and the interconnected host processor port. Figure 10 shows an example. In this scenario, Switch #1 must have two tail-tagged interfaces (this includes the switch's SGMII and RGMII ports). Because the KSZ switch can only support one interface with tag-tailing mode enabled, this use case is not supported.

FIGURE 10: TAIL TAGGING ON TWO INTERFACES OF A SINGLE SWITCH

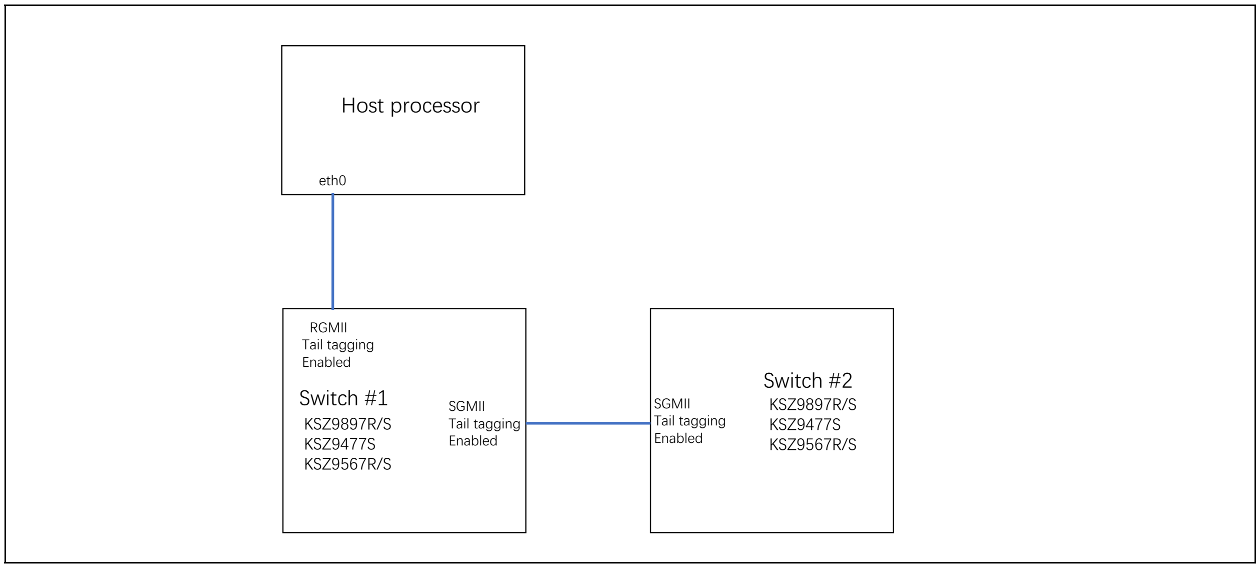
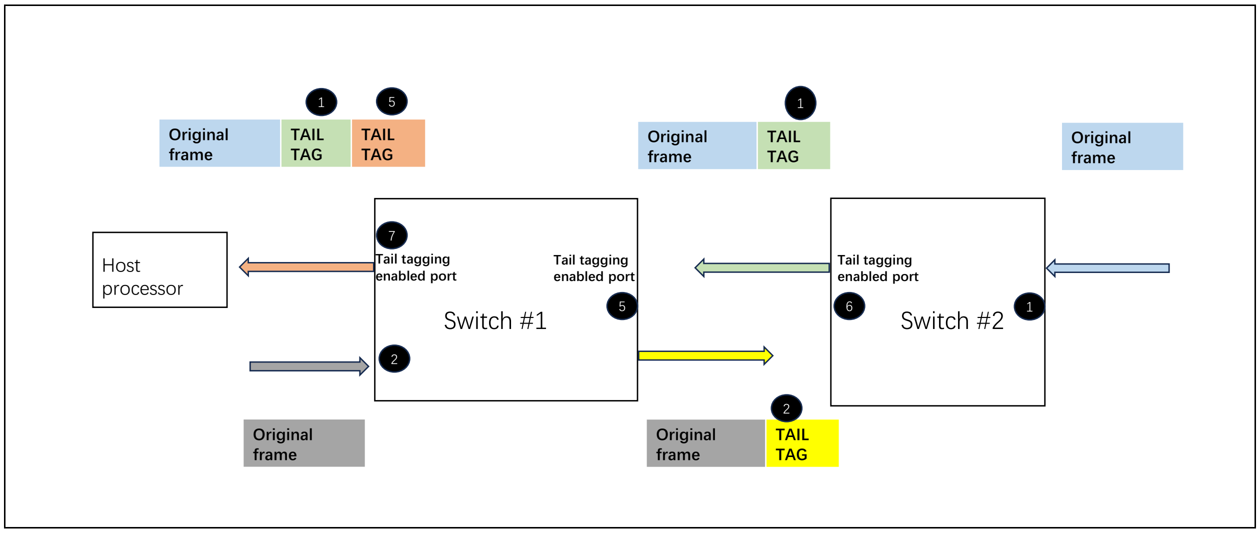


Figure 11 explains how the two KSZ switch devices insert the tail tagging bytes when a received frame at Switch #2 is destined for a host processor MAC port through the intermediate Switch #1, or a received frame at Switch #1 is destined for Switch #2’s front ports.

Suppose a frame is received at port 1 of Switch #2 and to be transmitted at port 6. Then it will be received at port 5 of Switch #1 and be forwarded at port 7 of the switch to a host processor MAC port.

Tail-tagged bytes indicating the ingress port ID “1” by Switch #2 at its egress port 6 are inserted in the original frame. Then, the frame is inserted with additional tail-tag bytes indicating the ingress port ID “5” by Switch #1 at the egress port 7. The host processor system software needs to parse the two inserted tail-tagging bytes to understand the source ports in order to respond to the packet along the same path.

FIGURE 11: TAIL TAGGING BYTES HANDLED BETWEEN THE INTERCONNECTED KSZ SWITCH DEVICES



Similarly, if a frame is received at port 2 of Switch #1 and to be transmitted at port 5, it is received at port 6 of Switch #2 and forwarded at port 1 of the switch.

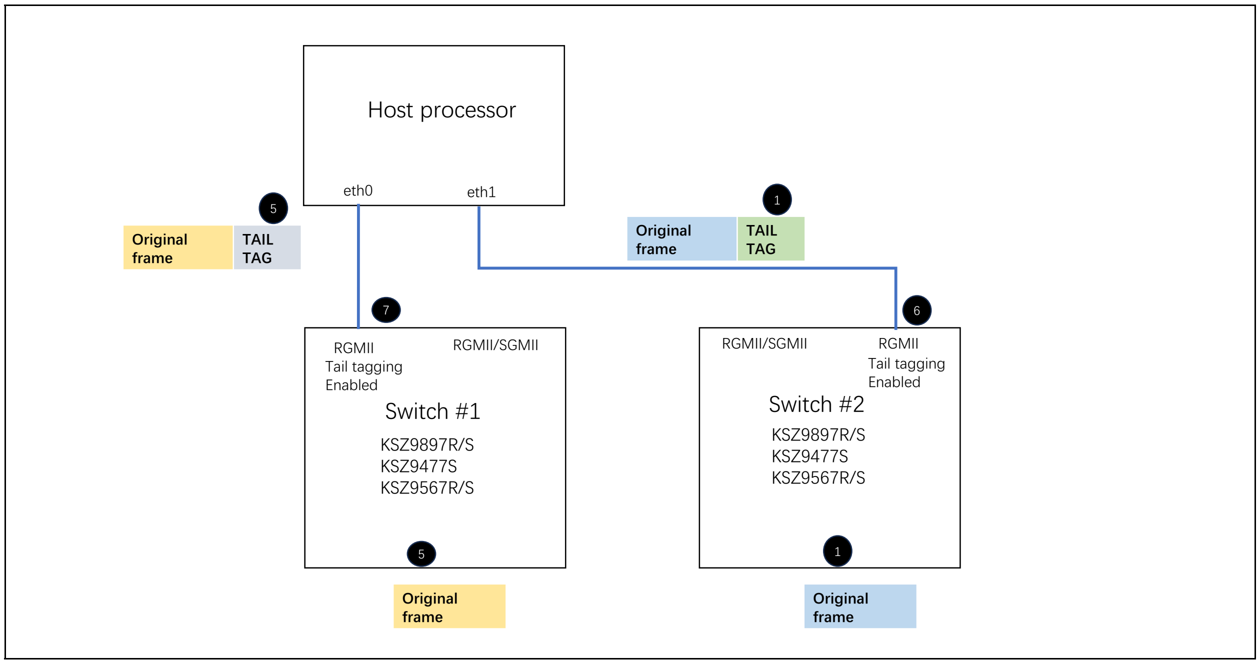
Tail-tag bytes indicating the ingress port ID “2” by Switch #1 is inserted in the original frame at its egress port 5. Then, this frame is received at Switch #2's port 6 and forwarded to its port 1.

To support frame forwarding paths indicated in Figure 11, Switch #1 must enable Tail Tagging mode on two ports (port 5 and port 7 in the example), which cannot be supported by all the underlying KSZ switch parts.

4.2.4 USING SEPARATE HOST PROCESSOR MAC PORTS

Since the Tail Tagging mode cannot be enabled in more than one switch, two separate host processor MAC ports were utilized in this use case (see Figure 12). Each port is connected to a KSZ switch device. This alternative approach requires enabling Tail Tagging mode on one switch port (i.e., the host processor-facing port).

FIGURE 12: SETUP OF TWO SEPARATE HOST MAC INTERFACES CONNECTING TO TWO SEPARATE SWITCHES



For example, if a frame is received at port 1 of Switch #2 and to be transmitted at port 6, tail-tagged bytes will be inserted in the original frame to indicate the ingress port ID “1” by Switch #2 at egress port 6. Similarly, a frame received at port 5 of Switch #1 will have tail-tagged bytes inserted to indicate the ingress port ID “5” by Switch #1 at the egress port 7. The host processor system software handles the received packets separately to understand the source ports for later use in response to the transmitted packet in the same path.

For this approach, the required change is to enable the two KSZ switches and two host processor MAC ports in the Linux Device tree configuration file.

The key is on the host platform-specific Linux device tree. To support two switches on DSA, the addition of "dsa,member = <0 0>;" and either "dsa,member = <1 0>;" or "dsa,member = <0 1>;" within the section of the I^{2}C or SPI definition of the device tree is required as shown in Example 2.

The "dsa,member" is a two-element list that indicates which DSA cluster and position within the cluster are taken by the switch. <0 0> represents cluster 0, switch 0. <0 1> represents cluster 0, switch 1. <1 0> represents cluster 1, switch 0. A switch that is not part of any cluster (a single device hanging off a CPU port) must not specify this property.

For more details, refer to https://www.kernel.org/doc/Documentation/networking/dsa/dsa.txt.

Example 1 and Example 2, based on KSZ9477, are meant to enable two switch devices, along with two host processor MAC ports, to support the use case in the platform-specific device tree configuration file.

For the host processor port configuration, Example 1 shows that the host processor has two RGMII MAC ports, with eth0 and eth1 being the port instances in Linux. eth0 corresponds to my_mac0 controller and eth1 corresponds to my_-mac1 node in the device tree `.dts` source file.

EXAMPLE 1: DEVICETREE CONFIGURATION FOR TWO HOST PROCESSOR MAC INTERFACES DECLARATION

```
&my_mac0 {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_my_mac0>;
        phy-mode = "rgmii-id";
        status = "okay";

        fixed-link{
            speed = <1000>;
            full-duplex;
        };
        //additional platform-specific properties
};

&my_mac1 {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_my_mac1>;
        phy-mode = "rgmii-id";
        status = "okay";

        fixed-link{
            speed = <1000>;
            full-duplex;
        };
        //additional platform-specific properties
};
```

Example 1 details where the necessary properties of the two MAC nodes are specified.

For KSZ switch device configuration, Example 2 assumes that the I^{2}C interfaces are being used as the management interfaces. Users can define the two I^{2}C interfaces with DSA members in the device tree.

Since there are six (6) front ports in each KSZ9477 switch, the host processor software needs to identify all the ports within the two switches, which include twelve ports in total. Customers should define twelve ports (e.g., lan1 ~ lan12) clearly to avoid confusion with the port names. Note that the id in “label = "lan?" should be unique.

EXAMPLE 2: DEVICETREE CONFIGURATION FOR THE TWO I^{2}C DEVICES DECLARATION

```
/* The first I2C interface with das member and port definition of device tree*/
&i2c4 {
      clock-frequency = <400000>;
      pinctrl-names = "default";
      pinctrl-0 = <&pinctrl_i2c4>;
      status = "okay";

      ksz9477_1: ksz9477_1@5f {
            compatible = "microchip,ksz9477";
            reg = <0x5f>;
            interrupt-parent = <&gpio4>;
            interrupts = <21 IRQ_TYPE_LEVEL_LOW>;
            phy-mode = "rgmii-id";
            status = "okay";

            /*cluster 0 , switch device 0*/
            dsa,member = <0 0>;
        ports {
                #address-cells = <1>;
                #size-cells = <0>;
            port@0 {
             reg = <0>;
              label = "lan1";
            };
            port@1 {
                reg = <1>;
             label = "lan2";
            };
            port@2 {
               reg = <2>;
                label = ”lan3”;

            port@3 {
             reg = <0>;
              label = "lan4";
            };
            port@4 {
                reg = <1>;
             label = "lan5";
            };


          port@5 {
              reg = <5>;
             label = "cpu";


              ethernet = <&my_mac0>;
           phy-mode = "rgmii-id";
          fixed-link {
             speed = <1000>;
                full-duplex;
              };
            };
            port@6 {
                reg = <6>;
             label = "lan6";
            };

         };
    };
};
```

EXAMPLE 2: DEVICETREE CONFIGURATION FOR THE TWO I^{2}C DEVICES DECLARATION

```
/*The second I2C interface definition for DSA*/
&i2c5 {
      clock-frequency = <400000>;
      pinctrl-names = "default";
      pinctrl-0 = <&pinctrl_i2c5>;
      status = "okay";

      ksz9477_2: ksz9477_2@5f {
            compatible = "microchip,ksz9477";
            reg = <0x5f>;
            interrupt-parent = <&gpio4>;
            interrupts = <3 IRQ_TYPE_LEVEL_LOW>;
            phy-mode = "rgmii-id";
            status = "okay";

            /*cluster 1 , switch device 0*/
            dsa,member = <1 0>;
        ports {
                #address-cells = <1>;
                #size-cells = <0>;
            port@0 {
             reg = <0>;
              label = "lan7";
            };
            port@1 {
                reg = <1>;
            label = "lan8";
            };
            port@2 {
              reg = <2>;
                label = "lan9";

            port@3 {
             reg = <0>;
              label = "lan10";
            };
            port@4 {
                reg = <1>;
            label = "lan11";
            };
          port@5 {
               reg = <5>;
                 label = "cpu";
                 ethernet= <&my_mac1>;
            phy-mode = "rgmii-id";



            fixed-link {
             speed = <1000>;
                full-duplex;
               };
            };
            port@6 {
                reg = <6>;
            label = "lan12";
            };

          };
     };
};
```

Where, ksz9477_1 and ksz9477_2 are the two switch devices connected to the two I^{2}C bus, i2c4 and i2c5. Each switch has six ports defined with unique labels from lan1 to lan12.

The Example 2 setup ensures that the host processor's two MAC ports and all 12 switch ports are properly configured in the device tree.

APPENDIX A: REVISION HISTORY

TABLE A-1: REVISION HISTORY

Revision Level & Date	Section/Figure/Entry	Correction
DS00006048A (07-03-25)	Initial release	

NOTES:

Microchip Information

Trademarks

ISBN: 979-8-3371-1524-5

Legal Notice

Microchip Devices Code Protection Feature