

EVB KSZ9477 Software Setup Guide

Rev 0.4

July 28, 2017

Table of Contents

1	Revision History	4
2	Introduction	5
3	Software Download	5
4	Booting the EVB-KSZ9477	5
5	The U-Boot	5
5.1	U-Boot Environment Variables.....	6
5.2	Ethernet MAC address and IP address	6
6	Linux Boot	7
6.1	IP and MAC Addresses	8
7	Quick Start Switch Configurations	10
7.1	Standard Switch with RSTP support.....	10
7.2	PTP	10
7.2.1	Using GPS-based time server as Grandmaster clock.....	10
7.2.2	Using one of the switch as Grandmaster clock	11
7.3	Device Level Ring (DLR)	12
7.4	High-Availability Seamless Ring (HSR)	15
7.5	802.1X Port based Authentication	18
8	Appendix.....	20
8.1	Linux/Driver command line boot arguments	20
8.1.1	multi_dev=0	21
8.1.2	multi_dev=1	21
8.1.3	multi_dev=2	22
8.1.4	iba=0	22
8.1.5	avb=0	23
8.1.6	stp=1	23
8.1.7	authen=1	23
8.1.8	sw_host_port=#	23
8.1.9	ports=#	23
8.2	Device Creation.....	23
9	AVB Support (MRP Daemon).....	23

10	PTP Stacks	24
10.1	Linux PTP Stack.....	24
10.2	PTP Profiles.....	24
10.3	Linux PTP Configurations.....	25
10.4	Linux PTP Utilities.....	26
10.4.1	phc2sys.....	26
10.4.2	pmc.....	27
10.4.3	phc_ctl.....	28

1 Revision History

Revision	Date	Summary of Changes
0.1	12/21/2016	Initial revision.
0.2	02/08/2017	Update U-Boot configurations for DLR and HSR.
0.3	05/12/2017	Add Linux PTP stack utilities information.
0.4	07/28/2017	Use EVB-KSZ9477 name.

2 Introduction

This document describes the software setup for EVB-KSZ9477 board. The EVB-KSZ9477 has SAMA5D3 SOC and KSZ9477 switch. The board NAND flash is pre-programmed with the software needed to evaluate and test the KSZ9477 switch functionality.

The EVB-KSZ9477 shipped with the U-Boot, Linux kernel with KSZ9477 drivers and applications. The applications include Linux PTP and OpenAVB stacks.

3 Software Download

Microchip distributes the binary images for the board as well as the complete source code.

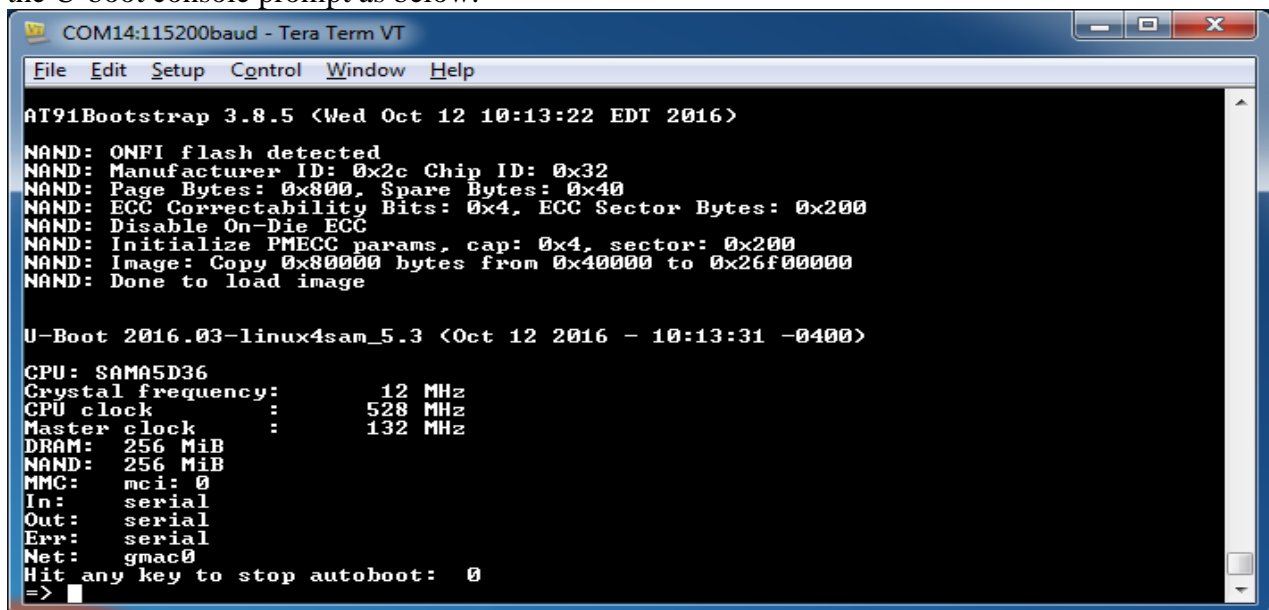
1. The binary images and instructions to program the NAND flash can be downloaded from Microchip website.
2. The complete source code can be downloaded from GitHub repository: <https://github.com/Microchip-Ethernet/EVB-KSZ9477>. The complete source code which includes U-Boot, build root, kernel and application is provided to build the image for this board.

4 Booting the EVB-KSZ9477

The serial communication interface can be connected to PC terminal (ex: Tera term) using USB to serial (Ex: FTDI TTL-232R-3V3) cable. Also, connect the 5V power to EVB-KSZ9477 board.

5 The U-Boot

After power up or reset, the board starts with the U-boot and the messages are seen on the debug console. The auto boot process can be interrupted by hitting any key. Once interrupted, it shows the U-boot console prompt as below.



```
COM14:115200baud - Tera Term VT
File Edit Setup Control Window Help

AT91Bootstrap 3.8.5 <Wed Oct 12 10:13:22 EDT 2016>
NAND: ONFI flash detected
NAND: Manufacturer ID: 0x2c Chip ID: 0x32
NAND: Page Bytes: 0x800, Spare Bytes: 0x40
NAND: ECC Correctability Bits: 0x4, ECC Sector Bytes: 0x200
NAND: Disable On-Die ECC
NAND: Initialize PMECC params, cap: 0x4, sector: 0x200
NAND: Image: Copy 0x80000 bytes from 0x40000 to 0x26f00000
NAND: Done to load image

U-Boot 2016.03-linux4sam_5.3 <Oct 12 2016 - 10:13:31 -0400>

CPU: SAMA5D36
Crystal frequency:      12 MHz
CPU clock               :    528 MHz
Master clock           :    132 MHz
DRAM: 256 MiB
NAND: 256 MiB
MMC: mci: 0
In: serial
Out: serial
Err: serial
Net: gmac0
Hit any key to stop autoboot: 0
=>
```

5.1 U-Boot Environment Variables

The 'printenv' command shows all the environment variables available in the U-Boot.

```

=> printenv
_avb=1
_bootargs=console=ttyS0,115200 mtdparts=atmel_nand:256k(bootstrap)ro,512k(uboot)ro,256k(env),256k(env_redundant),256k(spare),512k(dtb),6M(kernel)ro,-(rootfs) rootfstype=ubifs ubi.mtd=7 root=ubi0:rootfs rw video=LUDS-1:800x480-16 ${extra_param}_chip=9897
_drvname=spi-ksz${_chip}
_iba=1
avb=0
baudrate=115200
bootargs=console=ttyS0,115200 mtdparts=atmel_nand:256k(bootstrap)ro,512k(uboot)ro,256k(env),256k(env_redundant),256k(spare),512k(dtb),6M(kernel)ro,-(rootfs) rootfstype=ubifs ubi.mtd=7 root=ubi0:rootfs rw video=LUDS-1:800x480-16 ${extra_param}_bootcmd=nand read 0x22000000 0x00200000 0x0035DAC8; run prep_boot; run load_dts
bootdelay=1
dev1_ports=${_drvname}.eth1_ports=${eth1_ports}
dev1_proto=${_drvname}.eth1_proto=${eth1_proto}
dev1_vlan=${_drvname}.eth1_vlan=${eth1_vlan}
dev2_ports=${_drvname}.eth2_ports=${eth2_ports}
dev2_proto=${_drvname}.eth2_proto=${eth2_proto}
dev2_vlan=${_drvname}.eth2_vlan=${eth2_vlan}
eth1_ports=3
eth1_proto=hsr
ethact=gmac0
ethaddr=00:10:A1:94:77:01
extra_param=${param1} ${param2} ${param3} ${param4} ${param5} ${param6} ${param7}
ipaddr=192.168.0.1
load_dts=nand read 0x21000000 0x00180000 0x00007BE8; bootz 0x22000000 - 0x21000000
0
param1=${_drvname}.authen=${authen}
param2=${_drvname}.multi_dev=${multi_dev}
param3=${_drvname}.avb=${avb}
param4=${_drvname}.iba=${iba}
param5=${_drvname}.stp=${stp}
param6=${dev1_ports} ${dev1_vlan} ${dev1_proto}
param7=${dev2_ports} ${dev2_vlan} ${dev2_proto}
prep_boot=setenv -f subst_var 1; setenv -f bootargs "${_bootargs}"
serverip=192.168.0.100
stderr=serial
stdin=serial
stdout=serial
subst_var=0

Environment size: 1664/131067 bytes
=>

```

5.2 Ethernet MAC address and IP address

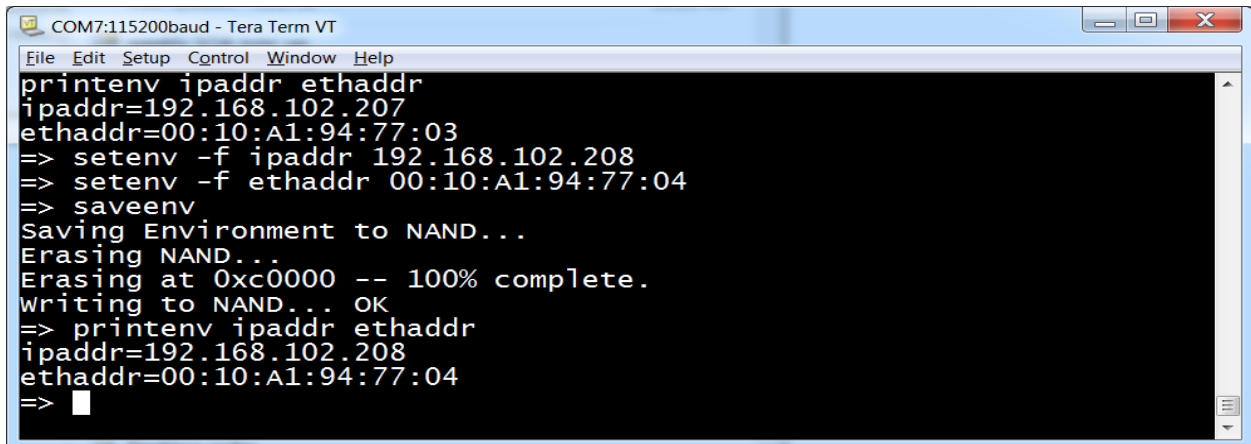
The MAC Address and IP address used by the SOC can be changed by 'setenv' command. It is important to change the MAC address and IP address if you have multiple switch boards in the network. Each board should have unique MAC address and IP address. After changing the MAC address and IP address save the changes using 'saveenv' command.

Ex:

```
setenv -f ethaddr 00:10:A1:94:77:04
```

```
setenv -f ipaddr 192.168.102.208
```

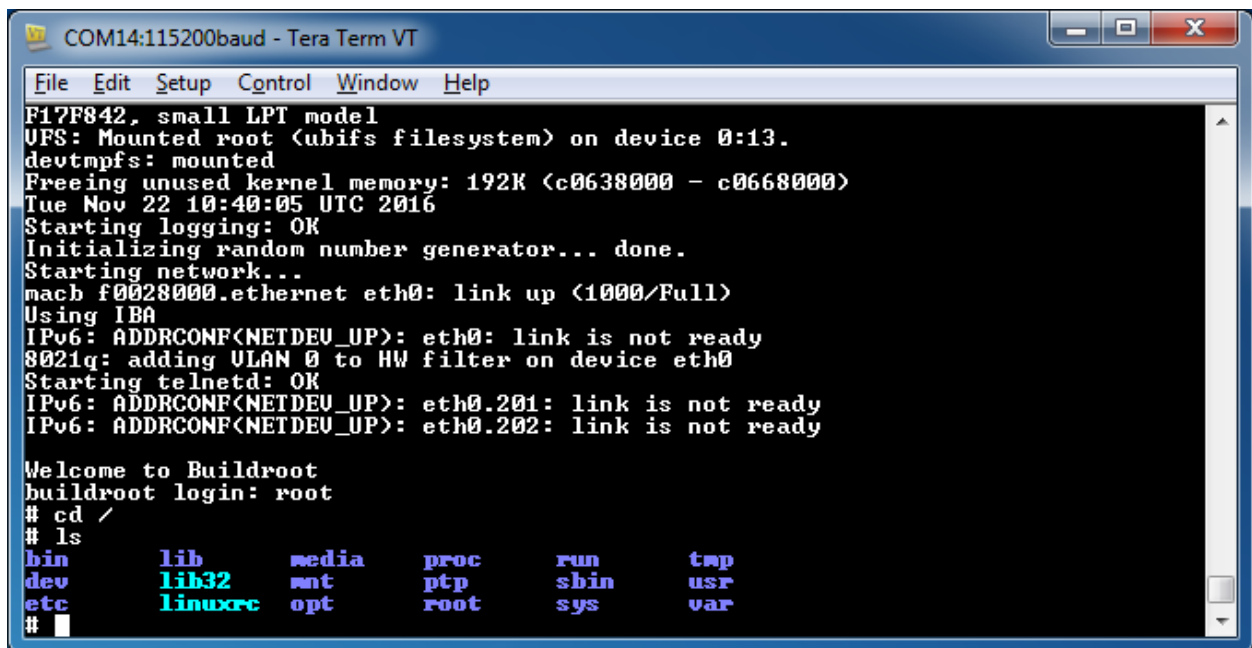
```
saveenv
```



```
COM7:115200baud - Tera Term VT
File Edit Setup Control Window Help
printenv ipaddr ethaddr
ipaddr=192.168.102.207
ethaddr=00:10:A1:94:77:03
=> setenv -f ipaddr 192.168.102.208
=> setenv -f ethaddr 00:10:A1:94:77:04
=> saveenv
Saving Environment to NAND...
Erasing NAND...
Erasing at 0xc0000 -- 100% complete.
Writing to NAND... OK
=> printenv ipaddr ethaddr
ipaddr=192.168.102.208
ethaddr=00:10:A1:94:77:04
=> █
```

6 Linux Boot

Upon power up or reset, the board boots the Linux kernel and login prompt will be displayed. By default there is no password set for root login.



```
COM14:115200baud - Tera Term VT
File Edit Setup Control Window Help
F17F842, small LPT model
UFS: Mounted root (ubifs filesystem) on device 0:13.
devtmpfs: mounted
Freeing unused kernel memory: 192K (c0638000 - c0668000)
Tue Nov 22 10:40:05 UTC 2016
Starting logging: OK
Initializing random number generator... done.
Starting network...
mach f0028000.ethernet eth0: link up (1000/Full)
Using IBA
IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
8021q: adding VLAN 0 to HW filter on device eth0
Starting telnetd: OK
IPv6: ADDRCONF(NETDEV_UP): eth0.201: link is not ready
IPv6: ADDRCONF(NETDEV_UP): eth0.202: link is not ready

Welcome to Buildroot
buildroot login: root
# cd /
# ls
bin      lib      media   proc     run      tmp
dev      lib32   mnt     ptp      shin     usr
etc      linuxrc opt      root     sys      var
# █
```

6.1 IP and MAC Addresses

In a network, each SOC board should have its own unique IP and MAC address. When setting up the network, make sure to check that each one has a different address. SOC boards are typically shipped with the same IP and MAC address, so multiple SOC boards must have their addresses changed before being connected into a network.

If the IP address and MAC addresses are not configured as described in the section 4.2, below procedure can be used to configure the MAC and IP addresses in the Linux host.

If the addresses of an SOC board (eth0 interface) are unknown, they can be determined by executing the *ifconfig* command as below.


```

COM14:115200baud - Tera Term VT
File Edit Setup Control Window Help
# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:10:A1:94:77:01
          inet addr:192.168.0.1  Bcast:255.255.255.0  Mask:255.255.255.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:53098 errors:0 dropped:0 overruns:0 frame:0
          TX packets:53098 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:5330976 (5.0 MiB)  TX bytes:5649564 (5.3 MiB)
          Interrupt:49 Base address:0x8000

eth0.201  Link encap:Ethernet  HWaddr 00:10:A1:94:77:01
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

eth0.202  Link encap:Ethernet  HWaddr 00:10:A1:94:77:01
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

eth0.203  Link encap:Ethernet  HWaddr 00:10:A1:94:77:01
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

eth0.204  Link encap:Ethernet  HWaddr 00:10:A1:94:77:01
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

eth0.205  Link encap:Ethernet  HWaddr 00:10:A1:94:77:01
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

eth0.206  Link encap:Ethernet  HWaddr 00:10:A1:94:77:01
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

#
  
```

The switch MAC address, IP address, netmask and broadcast addresses can be configured in the /etc/network/interfaces file. The IP address is actually read from the U-Boot ipaddr environmental variable.

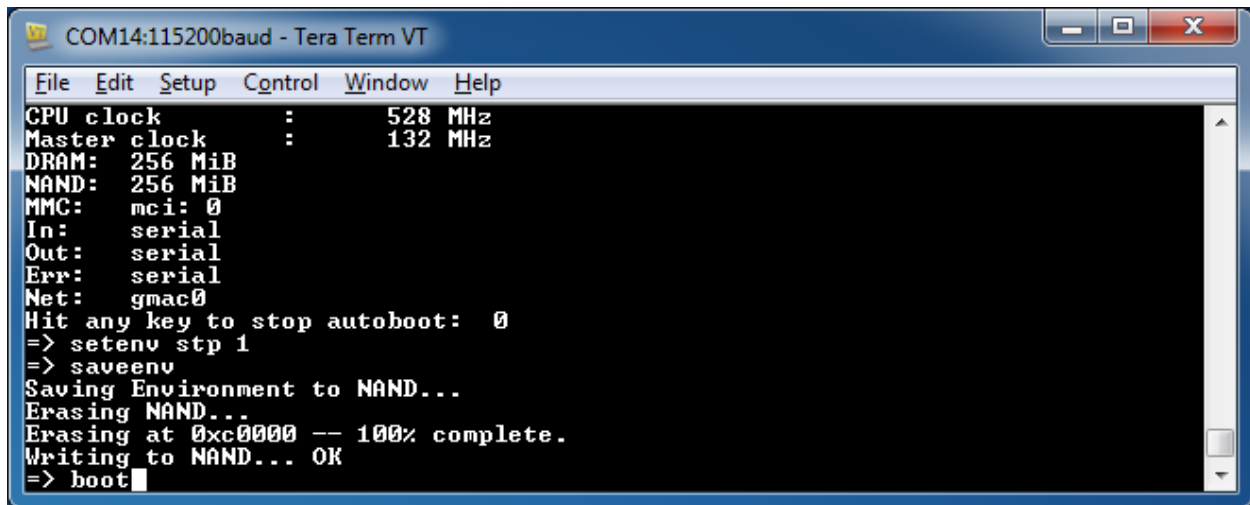
7 Quick Start Switch Configurations

Typical configuration provided below to quickly setup the switch for the required feature. In all the below examples the default configuration is specified. Unless user changes this configuration and wants to revert it to default configuration it is not required to modify these configurations.

7.1 Standard Switch with RSTP support

The stp U-Boot argument enables the RSTP support. The stp is disabled by default.

Run the below commands on the U-Boot console prompt.



```
COM14:115200baud - Tera Term VT
File Edit Setup Control Window Help
CPU clock      :      528 MHz
Master clock   :      132 MHz
DRAM: 256 MiB
NAND: 256 MiB
MMC: mci: 0
In: serial
Out: serial
Err: serial
Net: gmac0
Hit any key to stop autoboot:  0
=> setenv stp 1
=> saveenv
Saving Environment to NAND...
Erasing NAND...
Erasing at 0xc00000 -- 100% complete.
Writing to NAND... OK
=> boot
```

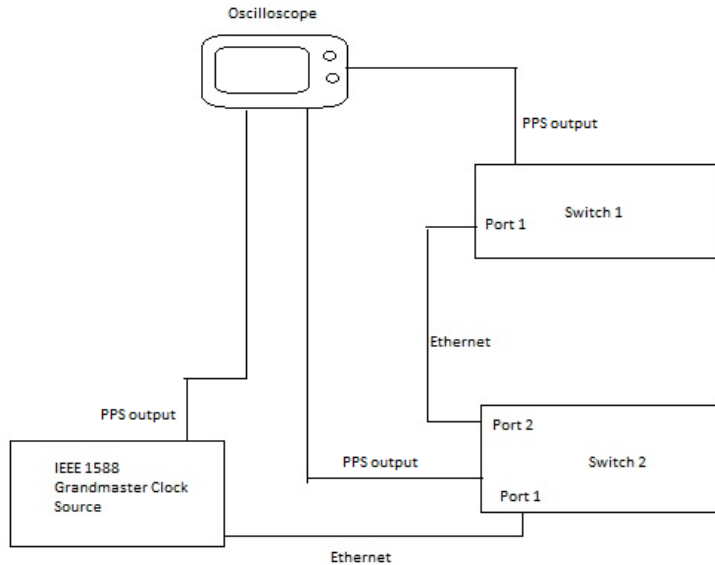
7.2 PTP

The open source Linux PTP stack is provided on the EVB-KSZ9477 board. The Linux PTP stack executable */usr/sbin/ptp4l* can run in both PTP and 802.1AS modes. The directories */ptp/e2e* and */ptp/p2p* holds the configurations files for End-to-End and Peer-to-Peer PTP setup. The Linux PTP stack was modified to work using one network device, but it is designed to use multiple network devices. The default configurations for Transparent Clock mode setup are provided under the directories */ptp/e2e/tc* and */ptp/p2p/tc*. The gPTP/AVB mode configurations are available under */ptp/avb* directory.

7.2.1 Using GPS-based time server as Grandmaster clock

This setup uses GPS-based time server as the grandmaster clock for the PTP network. It is the source of the clock information that is distributed to the KSZ9477 switches. In the below picture

the switch ports 1 and port 2 are shown, there is no functional difference between ports and any switch port can be used. The PPS output is measured using oscilloscope. This PTP setup uses End-to-End delay mechanism.



Set up the IEEE1588 grandmaster time server for the following PTP variables:

- End-to-end (E2E) delay mechanism
- IPv4 multicast addressing

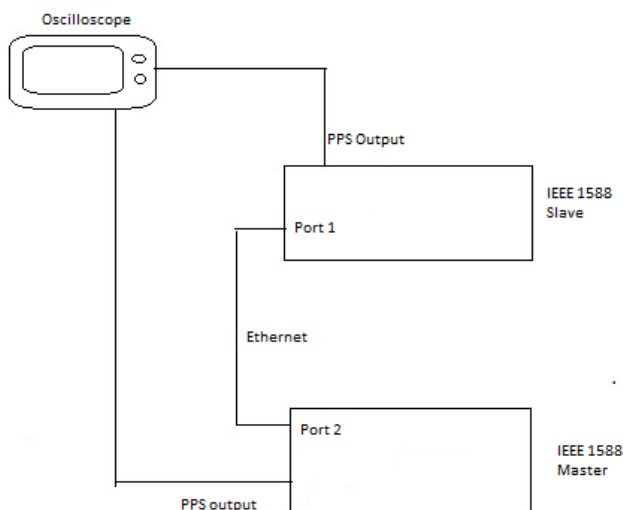
Use the `linuxptp.sh` script to start PTP stack on Switch1 and Switch2.

```
#cd /ptp/e2e  
#./linuxptp.sh
```

The PPS outputs from switches and grandmaster need to be measured using oscilloscope.

7.2.2 Using one of the switch as Grandmaster clock

This setup uses one of the switches as reference grandmaster clock for the PTP network. It is the source of the clock information that is distributed to the KSZ9xxx switches. In the below picture the switch ports 1 and port 2 are shown, there is no function difference between ports and any switch port can be used. The PPS output is measured using oscilloscope.



Set up the IEEE1588 grandmaster time server for the following PTP variables:

- End-to-end (E2E) delay mechanism
- IPv4 multicast addressing

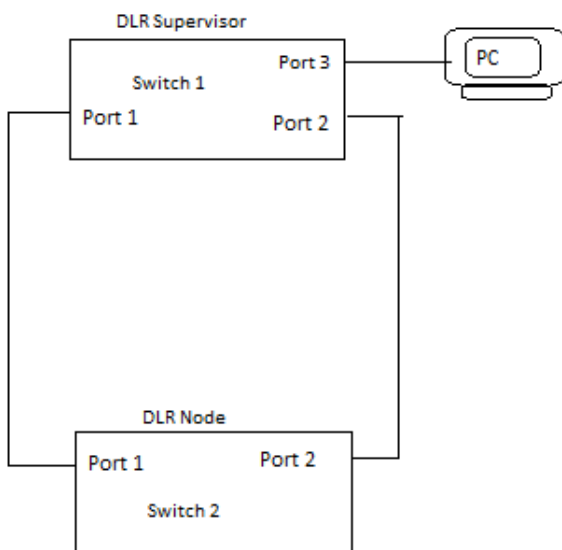
Use the `linuxptp.sh` script to start PTP stack on Switch1 and Switch2.

```
#cd /ptp/e2e  
#./linuxptp.sh
```

The PPS outputs from switches and grandmaster need to be measured using oscilloscope.

7.3 Device Level Ring (DLR)

Any two of the switch port can be configured to participate in the DLR network. In this setup the port1 and port2 of the switches are used for the DLR network. The other ports of the switch can be used to connect the switch to outside network.



1. The Linux boot arguments need to be modified to enable DLR in the driver. Run the below commands on the U-Boot prompt. To enter to U-boot prompt press the reset button and hit any key when prompted on the console.

```
setenv eth1_ports 3
setenv eth1_proto dlr
setenv eth1_vlan 0x7e
# for enabling bridge mode
setenv eth2_vlan 0x7f
setenv multi_dev 1
saveenv
boot
```

```

COM14:115200baud - Tera Term VT
File Edit Setup Control Window Help
RomBOOT

AT91Bootstrap 3.8.5 <Wed Oct 12 10:13:22 EDT 2016>
NAND: ONFI flash detected
NAND: Manufacturer ID: 0x2c Chip ID: 0x32
NAND: Page Bytes: 0x800, Spare Bytes: 0x40
NAND: ECC Correctability Bits: 0x4, ECC Sector Bytes: 0x200
NAND: Disable On-Die ECC
NAND: Initialize PMECC params, cap: 0x4, sector: 0x200
NAND: Image: Copy 0x80000 bytes from 0x40000 to 0x26f00000
NAND: Done to load image

U-Boot 2016.03-linux4sam_5.3 <Oct 12 2016 - 10:13:31 -0400>

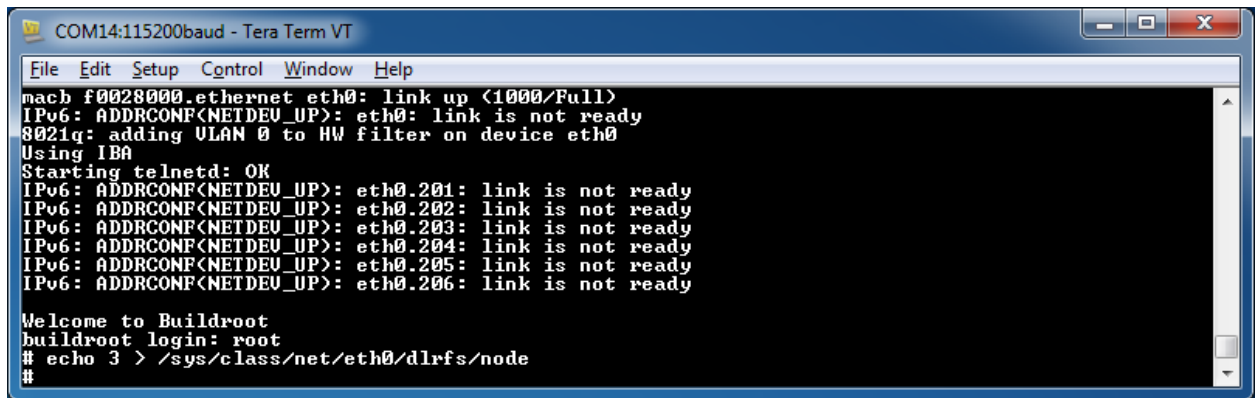
CPU: SAMA5D36
Crystal frequency:      12 MHz
CPU clock              : 528 MHz
Master clock           : 132 MHz
DRAM: 256 MiB
NAND: 256 MiB
MMC: mci: 0
In: serial
Out: serial
Err: serial
Net: gmac0
Hit any key to stop autoboot: 0
=> setenv eth1_ports 3
=> setenv eth1_proto dlr
=> saveenv
Saving Environment to NAND...
Erasing NAND...
Erasing at 0xc00000 -- 100% complete.
Writing to NAND... OK
=> boot
  
```

Make sure each switches have different HW address (it is easier to increment last digit of the HW address ie: 00:10:A1:94:77:01, 00:10:A1:94:77:02 and so on). See Section 4.2 to change the MAC address at the U-Boot prompt.

Repeat the above procedure for all the switches in the DLR network. After the **boot** command on the U-Boot prompt the board boots Linux kernel. Please login as explained in the section: 5.

2. Make sure each switch has different IP address; also ensure the IP addresses are in the same subnet. The IP address can be configured in the */etc/network/interfaces* file. The IP address can also be set at U-Boot prompt. See Section 4.2.
3. Then start the DLR supervisor by executing below command on to be supervisor node.
Ex: Execute the below command on the Switch1. The Switch1 becomes the supervisory node.

echo 3 > /sys/class/net/eth0/dlrfs/node



```
COM14:115200baud - Tera Term VT
File Edit Setup Control Window Help
macb f0028000.ethernet eth0: link up <1000/Full>
IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
8021q: adding VLAN 0 to HW filter on device eth0
Using IBA
Starting telnetd: OK
IPv6: ADDRCONF(NETDEV_UP): eth0.201: link is not ready
IPv6: ADDRCONF(NETDEV_UP): eth0.202: link is not ready
IPv6: ADDRCONF(NETDEV_UP): eth0.203: link is not ready
IPv6: ADDRCONF(NETDEV_UP): eth0.204: link is not ready
IPv6: ADDRCONF(NETDEV_UP): eth0.205: link is not ready
IPv6: ADDRCONF(NETDEV_UP): eth0.206: link is not ready

Welcome to Buildroot
buildroot login: root
# echo 3 > /sys/class/net/eth0/dlrf/s/node
#
```

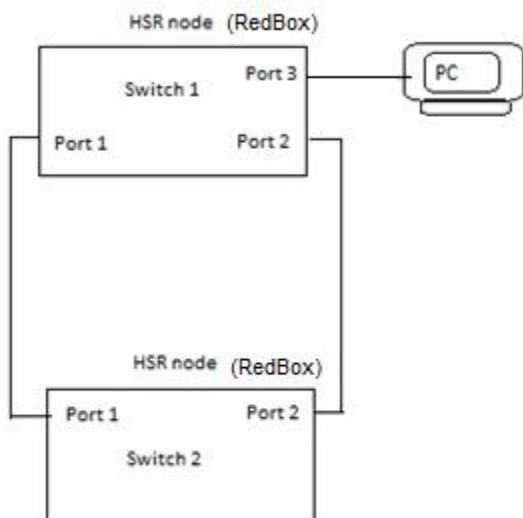
Note: The DLR supervisor selection can also be done U-Boot setting. This setting should be done only on one node (Switch1).

```
setenv dlr_prec 1
```

4. Connect the port1 and port2 of the switches as shown in the above diagram.
5. The external PC can be connected to other ports on the switch. Ex: PC is connected to port3 of the Switch1. Make sure the PC IP address is in the same subnet as the DLR nodes.
6. The connectivity between the two switches or between the external PC can be tested using ping command.

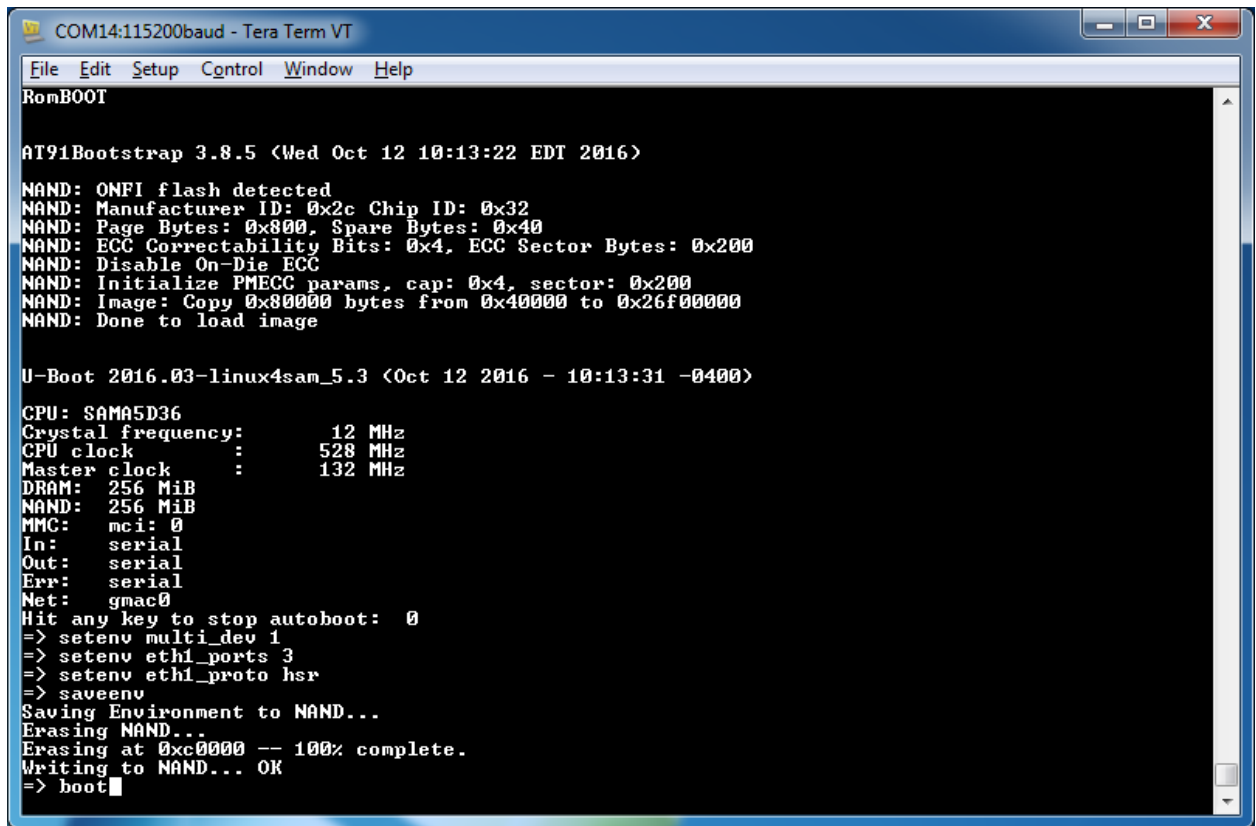
7.4 High-Availability Seamless Ring (HSR)

Any two of the switch port can be configured to participate in the HSR network. In this setup the port1 and port2 of the switches are used for the HSR network. The below instructions configures the switches as RedBox (Redundancy Box).



1. The Linux boot arguments need to be modified to enable HSR in the driver. Run the below commands on the U-Boot prompt. To enter to U-boot prompt press the reset button and hit any key when prompted on the console.

```
setenv multi_dev 1
setenv eth1_ports 3
setenv eth1_proto hsr
setenv eth1_vlan 0x7e
setenv eth2_proto redbox
setenv eth2_vlan 0x7f
saveenv
boot
```

```
COM14:115200baud - Tera Term VT
File Edit Setup Control Window Help
RomBOOT

AT91Bootstrap 3.8.5 <Wed Oct 12 10:13:22 EDT 2016>

NAND: ONFI flash detected
NAND: Manufacturer ID: 0x2c Chip ID: 0x32
NAND: Page Bytes: 0x800, Spare Bytes: 0x40
NAND: ECC Correctability Bits: 0x4, ECC Sector Bytes: 0x200
NAND: Disable On-Die ECC
NAND: Initialize PMECG params, cap: 0x4, sector: 0x200
NAND: Image: Copy 0x80000 bytes from 0x40000 to 0x26f00000
NAND: Done to load image

U-Boot 2016.03-linux4sam_5.3 <Oct 12 2016 - 10:13:31 -0400>

CPU: SAMA5D36
Crystal frequency:      12 MHz
CPU clock              :    528 MHz
Master clock           :    132 MHz
DRAM: 256 MiB
NAND: 256 MiB
MMC: mci: 0
In: serial
Out: serial
Err: serial
Net: gmac0
Hit any key to stop autoboot: 0
=> setenv multi_dev 1
=> setenv eth1_ports 3
=> setenv eth1_proto hsr
=> saveenv
Saving Environment to NAND...
Erasing NAND...
Erasing at 0xc0000 -- 100% complete.
Writing to NAND... OK
=> boot
```

Make sure each switch has different HW address (it is easier to increment last digit of the HW address ie: 00:10:A1:94:77:01, 00:10:A1:94:77:02 and so on). See Section 4.2 to change the MAC address at the U-Boot prompt.

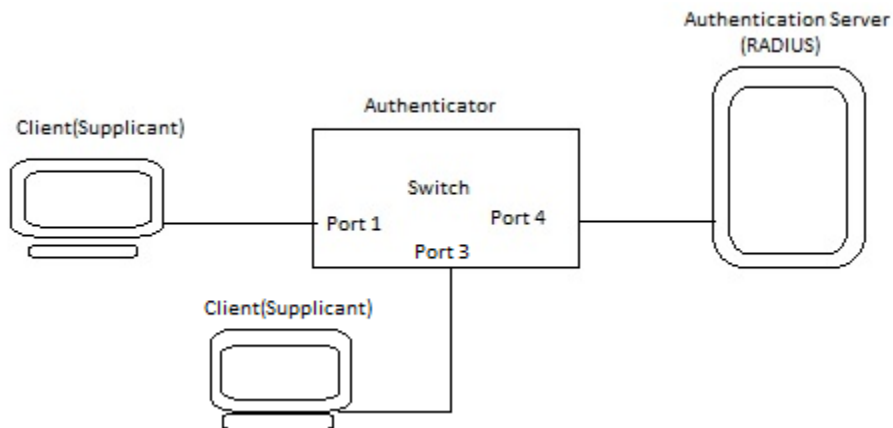
Repeat the above procedure for all the switches in the HSR network. After the **boot** command on the U-Boot prompt the board boots Linux kernel. Please login as explained in the section: 5.

2. Make sure each switch has different IP address, also ensure the IP addresses are in the same subnet. The IP address can be configured in the */etc/network/interfaces* file. The IP address can also be set at U-Boot prompt. See Section 4.2.
3. Connect the port1 and port2 of the switches as shown in the above diagram.
4. The connectivity between the two switches or between the external PC can be tested using ping command.

7.5 802.1X Port based Authentication

Below is the typical setup for 802.1x Authentication. The RADIUS server can be connected to any port on the switch. For example port 4 is used in the below setup.

The RADIUS server needs to be configured for the required authentication types Ex: passwd, certificate etc. The RADIUS server setup is outside the scope of this document.



1. The Linux boot arguments need to be modified to enable 801.X Authentication in the driver. Run the below commands on the U-Boot prompt. To enter to U-boot prompt press the reset button and hit any key when prompted on the console.

Run the below commands on the Switch U-Boot console prompt.

```
setenv authen 1
saveenv
boot
```

```

COM14:115200baud - Tera Term VT
File Edit Setup Control Window Help
RomBOOT

AT91Bootstrap 3.8.5 <Wed Oct 12 10:13:22 EDT 2016>

NAND: ONFI flash detected
NAND: Manufacturer ID: 0x2c Chip ID: 0x32
NAND: Page Bytes: 0x800, Spare Bytes: 0x40
NAND: ECC Correctability Bits: 0x4, ECC Sector Bytes: 0x200
NAND: Disable On-Die ECC
NAND: Initialize PMECC params, cap: 0x4, sector: 0x200
NAND: Image: Copy 0x80000 bytes from 0x40000 to 0x26f00000
NAND: Done to load image

U-Boot 2016.03-linux4sam_5.3 <Oct 12 2016 - 10:13:31 -0400>

CPU: SAMA5D36
Crystal frequency:      12 MHz
CPU clock               :    528 MHz
Master clock            :    132 MHz
DRAM: 256 MiB
NAND: 256 MiB
MMC: mci: 0
In: serial
Out: serial
Err: serial
Net: gmac0
Hit any key to stop autoboot: 0
=> setenv authen 1
=> saveenv
Saving Environment to NAND...
Erasing redundant NAND...
Erasing at 0x100000 -- 100% complete.
Writing to redundant NAND... OK
=> boot
  
```

After the **boot** command on the U-Boot prompt the board boots Linux kernel. Please login as explained in the section: 5.

2. The init script *S65authen* at */etc/init.d* will start the authenticator. The file */etc/sysconfig* contains the RADIUS server information. See the */etc/sysconfig* section to configure the authenticator information.

```

AUTHEN: on
RADIUS_SERVER: 192.168.102.40
RADIUS_SECRET: "testing123"
  
```

3. The 802.1X is enabled by executing below command. Using a value of 0 will disable this function.

```
echo 1 > /sys/class/net/eth0/sw/authen
```

```

COM14:115200baud - Tera Term VT
File Edit Setup Control Window Help
Starting network...
IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
8021q: adding VLAN 0 to HW filter on device eth0
macb f0028000.ethernet eth0: link up (1000/Full)
Using IBA
Starting telnetd: OK
IPv6: ADDRCONF(NETDEV_UP): eth0.201: link is not ready
IPv6: ADDRCONF(NETDEV_UP): eth0.202: link is not ready
IPv6: ADDRCONF(NETDEV_UP): eth0.203: link is not ready
IPv6: ADDRCONF(NETDEV_UP): eth0.204: link is not ready
IPv6: ADDRCONF(NETDEV_UP): eth0.205: link is not ready
IPv6: ADDRCONF(NETDEV_UP): eth0.206: link is not ready
Welcome to Buildroot
buildroot login: root
# echo 1 > /sys/class/net/eth0/sw/authen
#
    
```

4. Connect the RADIUS server to any of the port. Ex: port4. The PC running supplicant application can be connected to any of the ports. Ex: port1 and port2.
5. After successful authentication the connectivity can be tested between two supplicant using ping.

Note: 1. The ACL table is configured to allow only EAPOL and RADIUS traffic to pass through the ports. Hence, the port connected to do RADIUS server open for EAOPL traffic and all other traffic is blocked to server.

2. Once the authentication is enabled each port is blocked and this is controllable (block/unblock) through `/sys/class/net/eth0/sw#/#_authen_mode` where # is port number.

8 Appendix

8.1 Linux/Driver command line boot arguments

The EVB-KSZ9477 driver is pre-configured to enable most of the switch functions. If user wants to enable/disable certain feature, it can be done by changing the kernel boot arguments.

The Linux kernel command line boot arguments are provided to configure the KSZ9897 switch driver to enable some of the features. The switch operation and how the switch driver exposes itself to user or upper layer is depends on these command line arguments. The command line can be configured inside the kernel, but most often it can be set in a U-Boot environmental variable so that it can be changed easily.

The command line has the format `modulename.variablename`. The first part of the each command parameter is the driver module name, which is variable depending on the switch chip used. In this example the generic KSZ9897 name is used for all chips in the KSZ9897 family which includes KSZ947x and KSZ956x. The switch driver of KSZ9897 is implemented as an SPI driver, so its module name is `spi-ksz9897`.

The second part of the command parameter is the driver module variable name. For example, the variable name to enable AVB is `avb` and the variable name for STP support is `stp`. To enable AVB support the command parameter is `spi-ksz9897.avb=1`.

The U-Boot environmental variable used to specify the Linux command line is **bootargs**. Typical U-Boot command line can be “**console=ttyS0,115200 init=/etc/preinit spi-ksz9897.avb=1**.” Use the U-Boot “**setenv bootargs**” command to set the line or clear it. Use the “**saveenv**” command to save the environmental variables.

For some system platforms like the Atmel SAMA5D3 which has a very long **bootargs** variable already it is hard to modify it, so U-Boot ship with EVB-KSZ9477 was modified to expand variables automatically when the variable is executed as command.

For example, a long **bootargs** variable can have another variable inside it: “**console=ttyS0,115200 ... \${extra_param}**.” The **extra_param** variable then can be “**\${param1} \${param2} ... \${param#}**.” The **param1** variable is then used to set specific driver option: “**spi-ksz9897.avb=\${avb}**.” The variable **avb** then can be easily changed by users using U-Boot **setenv** command. The **bootargs** variable will be updated properly when U-Boot boots the kernel so the driver can get all its options.

As some driver options are set in certain values inside the driver it is not necessary to pass all driver options through the Linux command line. The U-Boot looks at the variable names inside those variables that contain other variables. If the variable is not defined and its name is not started with ‘_’ then the whole variable is dropped and not expanded.

For example, there are 2 variables setting the parameters:

```
param1=spi-ksz${_chip}.avb=${avb}
param2=spi_ksz${_chip}.stp=${stp}
_chip=9897
avb=1
```

The parameter “**spi-ksz9897.avb=1**” is placed inside the **bootargs** variable. The variable **stp** is not defined, so the parameter “**spi-ksz9897.stp=0**” will not appear at all.

The driver supports multiple device modes through **multi_dev** variable which specifies how the network device/interfaces (like: **eth0**, **eth1**) are created. The device mode setting allows the switch to expose its external ports as network devices so that they can be controlled individually.

8.1.1 multi_dev=0

This is the default operation of the switch when no special function is required. There is only a single **eth0** device and tail tagging is not used unless 1588 PTP function is required.

8.1.2 multi_dev=1

The driver will create as many network devices as the number of ports. There will be **eth0**, **eth1**, **eth2**, and so on. Tail tagging will be enabled to handle receiving and transmitting through the ports. The specific ports can be grouped into a single network device, so there will be many network devices associated with different groups of ports. VLAN is used to handle

receiving and transmitting through these devices.

In this mode the inherent switching functionality basically is disabled depending on how the ports are setup. The HSR setup needs this mode.

```
eth1_ports=<port bitmap>
eth1_vlan=<VLAN used>
eth1_proto=<dlr><hsr><redbox>
eth2_ports=<port bitmap>
eth2_vlan=<VLAN used>
```

The `eth1_ports` variable is used to specify the ports that are grouped into the first network device, `eth0`. The port bitmap is the combination of bits representing each port, so 3 mean ports 1 and 2; 4 mean port 3; 8 for port 4; and so on. For better understanding the hexadecimal value can be used, like 0x13. The `eth1_vlan` variable is used to indicate the VLAN Id used for that network device. The VLAN Id is virtual and is only used by the driver; it should not conflict with any VLAN Id found in actual network traffic. The `eth1_proto` variable is optional, and is used only when DLR or HSR protocol is used by the ports.

The `eth2_ports` is then used to specify the ports that are grouped into the second network device, `eth1`. The `eth2_vlan` is optional as the VLAN id used will be the last one used plus 1.

The `eth3_ports` is used to specify the ports that are grouped into the next network device, `eth2`. The `eth<#>_ports` after `eth1_ports` is optional, as any ports not in `eth1_ports` will be put into `eth1`.

For DLR and HSR support the number of ports in the network device should be exactly 2. They do not need to be ports 1 and 2 but it can be any 2 ports.

For DLR the extra ports in the switch still can participate in the DLR network. For that to work the `eth1_vlan` variable should not be specified and the `multi_dev` should be left in mode 0.

8.1.3 multi_dev=2

There is a single `eth0` device, and several virtual VLAN devices are used to support applications that need to communicate through specific port. In this case a network device like `eth0.201` is used by the application to only send and receive packet through port 1. The starting VLAN is set internally by the driver. If this number conflicts with VLAN traffic then a different number should be chosen.

8.1.4 iba=0

This is used to turn off IBA if the network connection between the switch's host port and the MAC controller is not reliable. This option is enabled by default.

8.1.5 avb=0

This is used to turn off AVB function as it is the default operation for AVB switch. When “avb=1” is used the “multi_dev=2” parameter is then assumed as that mode is required for correct operation. This option is enabled by default in the driver.

8.1.6 stp=1

This is used to turn on RSTP support. This option is disabled by default in the driver.

8.1.7 authen=1

This is used to turn on 802.1X Authentication. All ports will be blocked. However, a RADIUS server can be connected to any port. This option is disabled by default in the driver.

8.1.8 sw_host_port=#

The host port is the port that is connected to the MAC controller. The design of the KSZ9897 switch is any port can be a host port, but typically the last port of the switch is used as one. As such the driver selects the last port as host port, but in specific case like KSZ9477 where the last port can only be used in SGMII mode and port 6 can use RGMII/MII, port 6 is selected as the host port. In certain situation where those ports cannot be used the host port can be set in one of the 5 physical ports. In this case the `sw_host_port` variable can be set to specify the host port. A value of 0 in default means the driver selects the host port; any number up to the maximum number of ports in the switch can be used. Note that if the host port is not the last one the driver automatically moves the port number down when converting virtual port number to physical port number. So if `sw_host_port=1`, `eth0.201` means port 2, and so on.

8.1.9 ports=#

The driver gets the number of ports from the switch configuration register, but in certain situation it may be desirable to limit the port number as some ports are not used. The `ports` variable allows the port number to be set. Note the host port can be outside this number.

8.2 Device Creation

The virtual VLAN devices `eth0.201` and such need to be created using the `vconfig` command after the kernel boots. A script `vlan.sh` is provided to create those devices. For the EVB-KSZ9477 board the `/etc/init.d/S60vlandev` script is used to create these devices.

9 AVB Support (MRP Daemon)

Multiple Stream Reservation Protocol (MSRP) is used to communicate the stream bandwidth required for a stream talker to send data to stream listeners. Together with Multiple MAC Registration Protocol (MMRP) and Multiple VLAN Registration Protocol (MVRP), it is based from Multiple Registration Protocol (MRP), so a single MRP daemon program can be used to support all three protocols. This user application processes the MRP packets and allows AVB

application to start the MSRP declarations.

The MRP and other bandwidth configurations are handled by the switch driver so it is not necessary to run this daemon if no AVB application is running in the switch.

The MRP daemon used is from the open source OpenAVB project. The script `mrpd.sh` is in the `/ptp/avb` directory can be used to start the daemon. It is started automatically when AVB is enabled as indicated by the `/sys/class/net/eth0/sw/avb` file. The `/etc/init.d/S65ptp` script is used to start the service.

To terminate the `mrpd` program it is necessary to use the command “`killall mrpd.`”

10 PTP Stacks

Currently Microchip supports Linux PTP stack and is also provided on the EVB-KSZ9477. This is an open source application.

10.1 Linux PTP Stack

The Linux PTP stack executable can run in both PTP and AVB modes, and it is selected to run as the default PTP stack. Its executable is `ptp41`. As it requires a lot of parameters to specify its behavior a script is provided for easy invocation. In the directories `e2e` and `p2p` the script `linuxptp.sh` can be used to start the Linux PTP stack. The configuration file used is `default.cfg`. The Linux PTP stack was modified to work using one network device, but it is designed to use multiple network devices. In that case another subdirectory `tc` is provided under the directories `e2e` and `p2p` to run the PTP stack in multiple devices mode.

The Linux PTP stack can be run in AVB mode under the subdirectory `avb`. The configuration file used is `gPTP.cfg`.

To terminate the `ptp41` program enter `<Ctrl-C>` when run manually.

If one of the devices is wanted to be a grandmaster, modify the `priority1` variable in the `/ptp/avb/gPTP.cfg` configuration file to lower its priority so that it wins the BMC algorithm.

10.2 PTP Profiles

For user convenience several PTP configurations are provided to test PTP in different profiles.

The subdirectory `e2e` under `ptp` contains stack configuration files and scripts for easy invocation of PTP stacks to run in E2E mode. The script `linuxptp.sh` is used to invoke Linux PTP stack. The file `default.cfg` is Linux PTP configuration file.

There is a subdirectory `tc` which is used to run some PTP stacks in TC mode. It requires setting up the VLAN port forwarding feature as mentioned above. The script `linuxptp.sh` is used to invoke Linux PTP stack. The Linux PTP stack is not actually required to run in this mode.

The subdirectory `p2p` is about the same as subdirectory `e2e` but for P2P. The Linux PTP stack can run in OC mode, but it expects port 1 is the port connected to its peer. This can be setup

easily in a simple 2 device master-slave operation. With more devices it has to run in TC mode for proper operation.

The subdirectory `power` is for power profile. It requires setting up a VLAN device with the `power.sh` script.

The subdirectory `avb` is for AVB. The file `gPTP.cfg` is Linux PTP configuration file. It runs in Boundary Clock mode.

10.3 Linux PTP Configurations

The Linux PTP reads the configuration file specified by the “-f” parameter. The program has default values for all supported configurations. The configuration file can be used to override those. Most of the configurations can be left alone. The important ones are described below:

`twoStepFlag` `0,1`

It indicates running the clock in two-step mode. Note that two-step Transparent Clock is not supported in software. It is only used for Boundary Clock.

`slaveOnly` `0,1`

It indicates the clock cannot become a master clock.

`priority1` `0-255`

`priority2` `0-255`

These priorities can be changed so that a clock can become a master over another.

`domainNumber`

Some profiles require a specific domain number, like Telecom.

`transparent` `0,1`

It indicates the clock is running in Transparent Clock mode. For Boundary Clock it has to be 0.

`neighborPropDelayThresh`

This value is used by 802.1AS to determine whether the port is asCapable.

`step_threshold` `0.0000008`

`first_step_threshold` `0.0000005`

It is recommended to set these values to 800 and 500 ns respectively.

`transportSpecific` `0,1`

For 802.1AS the transportSpecific value is 1. Note some utilities talking to the stack need to use the same value.

`network_transport` `UDPv4, UDPv6, L2`

The PTP messages can be sent in IPv4, IPv6, or IEEE 802.3 frame.

`delay_mechanism` E2E, P2P, None

The delay mechanism can be E2E or P2P. It can also be None for testing purpose.

`productDescription`

`revisionData`

`manufacturerIdentity`

`userDescription`

These configurations are used for product identification only.

10.4 Linux PTP Utilities

There are additional utilities to support Linux PTP stack running in a system.

The main program is the PTP stack `ptp4l`, which has the following parameters:

`-p /dev/ptp0`

This specifies the PTP clock, which is normally `ptp0`.

`-i eth0`

This specifies the network device. For additional devices this parameter can be repeated.

`-n <num>`

For convenience this parameter can be used to indicate the number of ports, so the network device index is automatically incremented.

`-f <file>`

This specifies the configuration file.

`-m`

This specifies verbose mode for debugging and testing purpose.

`-l 4`

This specifies the logging level. If running in background the recommended level is 4 or lower. Otherwise there is too much logging in the system log file stored in `/var/log/messages`.

10.4.1 `phc2sys`

The `phc2sys` utility is used to synchronize the system clock with the PTP hardware clock so the system time is in sync between systems.

It supports the following parameters:

`-s /dev/ptp0`

This specifies synchronizing the system time with the PTP clock `ptp0`.

`-w`

This indicates waiting for the `ptp4l` program by sending Management messages to it and getting a response. One of the messages is used to get the UTC offset so that the utility can set the system clock correctly.

`-m`

This specifies verbose mode for debugging and testing purpose.

`-l 4`

This specifies the logging level. If running in background the recommended level is 4 or lower. Otherwise there is too much logging in the system log file stored in `/var/log/messages`.

`-a`

This starts the utility in automatic mode so that it configures everything by getting the information from the `ptp4l` program. It synchronizes the system clock when it detects the port is in slave mode and synchronizes the PTP clock if the port is in master mode. This is used in combination of the “`-r`” parameter.

`-T 1`

This is used to change the `transportSpecific` value to 1, which is used in 802.1AS.

10.4.2 `pmc`

The `pmc` utility is used to send PTP Management messages to the stack and display their responses.

It uses the following parameters:

`-u`

This is required to send PTP messages to the local device.

`-2`

The PTP Management message is sent with IEEE 802.3 frame.

`-4`

The PTP Management message is sent with IPv4 packet.

`-6`

The PTP Management message is sent with IPv6 packet.

`-t 1`

This is used to change the `transportSpecific` value to 1, which is used in 802.1AS.

If no more commands are entered then it enters into an interactive mode where the users enter each command individually. These commands start with either “`get`,” “`set`,” or “`target`.” The next parameter specifies the attribute of the PTP stack. The “`help`” command can be used to display which attributes are supported. Some of the common ones are

DEFAULT_DATA_SET

CURRENT_DATA_SET

PORT_DATA_SET

Some are used only in 802.1AS, with names end with _NP:

PORT_DATA_SET_NP

TIME_STATUS_NP

The command “target” is used to specify the target. The default is * (ALL). The target is the portIdentity, which is the combination of clockIdentity and the portNumber, and the format is “xxxxxx.ffff.yyyyyy-n.” The clockIdentity is most likely created using the device MAC address xx.xx.xx.yy.yy.yy and adding 0xffff in the middle.

The command and attribute are case insensitive and can be shortened. However any additional parameter is not. An example is the PORT_DATA_SET_NP command. A get command will display the neighborPropDelayThresh value and the asCapable indication. A set command will need the exact variable names:

```
set port_data_set_np neighborPropDelayThresh 400 asCapable 1
```

The command can be entered in the command line. It will need quotes to separate the commands:

```
pmc -u -t 1 “target 0010a1.ffff.947726-1” “get port_data_set_np”
```

```
pmc -u -t 1 “target 0010a1.ffff.947726-1” “set port_data_set_np  
neighborPropDelayThresh 800 asCapable 1”
```

10.4.3 phc_ctl

The phc_ctl utility is used to manipulate the clock.

It needs the clock device, either from a network device like eth0 or a real clock device like /dev/ptp0.

It supports the following parameters:

-q

This suppresses logging.

-l 4

This specifies the logging level. It is recommended to use the “-q” parameter all the time.

Next are the supported commands:

get

Return the current PTP time. Note this time will not be the same as the system time.

set [second.nanosecond]

Set the PTP time. If the time is not specified the current system time is used. As the system time will be different from the PTP time by UTC offset the program automatically adds that offset before setting the PTP time. Note that setting PTP time less than the UTC offset, which currently is 37, is illegal.

`cmp`

Return the difference between the PTP time and system time. Normally it will be the same as the UTC offset. As this program does not have UTC offset information it assumes this difference is the UTC offset and uses it to compensate the system time as set by the above command.

`adj <second.nanosecond>`

Adjust the PTP time by the specified amount. Note that the symbol of negative, a hyphen “-“, is interpreted by the program that a parameter should follow, so it cannot be used and the underscore “_” character is used instead.

`freq <ppb>`

Change the clock frequency. Like the above command the underscore “_” is used to specify a negative number.