Unmanaged Software Porting Guide

RELEASE UM 3.6.1

Unmanaged-PG106

Corporate Headquarters: San Jose, CA

January 24, 2018

For a comprehensive list of changes to this document, see the Revision History.

Table of Contents

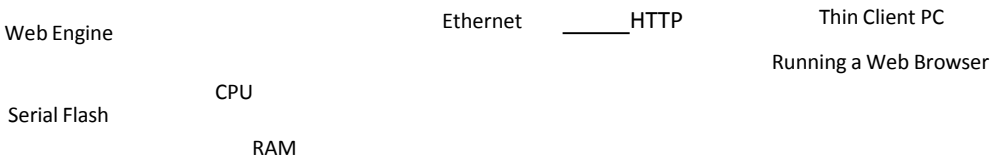
Broadcom® Unmanaged (UM) software is available in three packages:

- Unmanaged Web (UM-Web)
- Unmanaged Plus (UM+)
- Unmanaged Dumb (UM-Basic)

This document applies to all UM packages. It describes the UM software and provides instructions on how to use it. It is intended for developers who are using Broadcom StrataConnect™ chips.

The Unmanaged (UM) software is a light-weight management system on Level 2 IGMP snooping, VLAN, QoS, mirror, TRUNK, rate limit, etc. In order to provide a light-weight management service through HTTP protocol, it also supports various protocol stacks including the uIP/DHCP/Bonjour stack.

The UM software also includes an API to manipulate and configure switch features.



The UM executable binary is in external flash. The on-chip RAM/L2 cache is used for the runtime variables, the stack, and the heap. The overall memory layout solution for each chip is shown in the following sections. A generic description of each memory component is listed below.

- Loader: the memory component is an executable binary that is only included in the Unmanaged Web solution. It provides these features:
  - Verification of the firmware and launch firmware code.
  - Firmware upgrade service using the HTTP protocol.
- Firmware: an executable binary of main application.
- Persistence: a flash storage area for runtime variable storage. This area is managed by a UM software module (serializer). If a runtime variable is to be stored in this area, then serializer API registration must be called.
- Vendor Config: a flash storage area for configurations in config.um. A tool can be used insert the configuration into this storage area of the precompiled firmware image(*.image).
- Factory value: a flash storage area for programming items such as the MAC address. Factory value(s) should not be modified after consumer-end production.
- Data: a set of compile-time generated variables located in the data or bss section.

- Flash programing driver: the programing driver should be placed in a non-flash area because the system cannot execute code from flash and program the flash at the same time.
- Stack: a memory pool for the call stack.
- Heap: a memory pool for runtime memory allocation.
- DMA heap: a memory pool for runtime DMA memory allocation.
- Rx_buffer: a preallocate buffer for the packet receive of the internal processor.
- Bookkeeping: provides a message box between the loader and the firmware. It retains the variable values during a software reset. The firmware may use this area to inform the loader about which boot flow to execute (normal or firmware upgrade mode).

Note: The flash partition has been changed in this version. Do not use the um-3.2.x loader (or an earlier version) to do the web firmware upgrade with the um-3.3.x firmware, and vice versa. If this is done, it could cause an unexpected behavior, such as the exception error.

Flash Partitions and Memory Layout for Unmanaged Web (BCM5333X)

Flash Partitions and Memory Layout for Unmanaged Plus (BCM5333X)

Flash Partitions and Memory Layout for Unmanaged Web (BCM5340X)

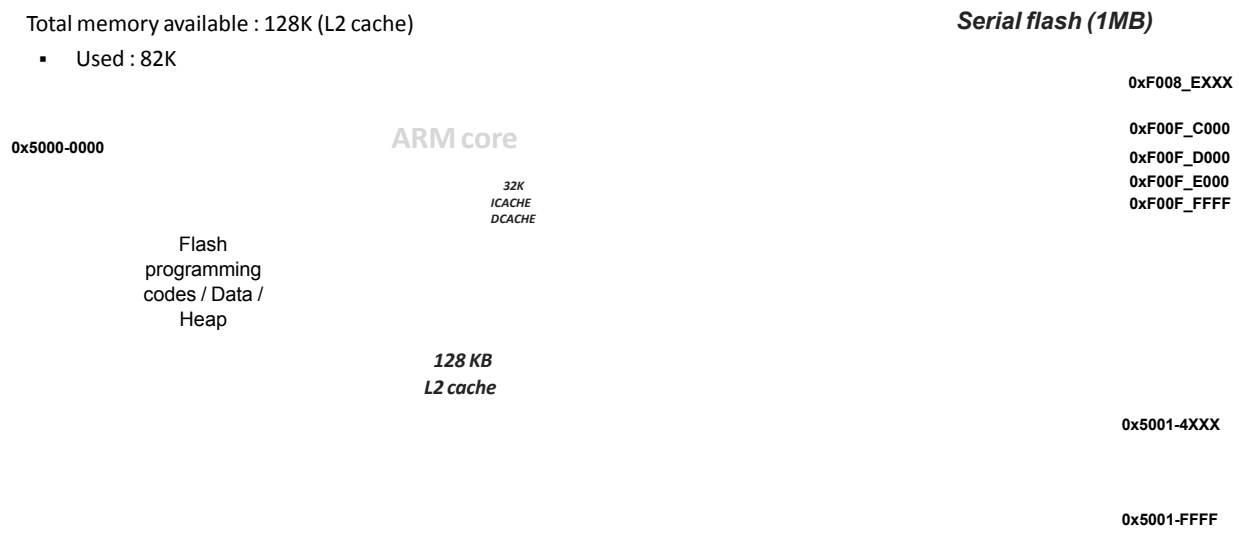
Flash Partitions and Memory Layout for Unmanaged Plus (BCM5340X)

Flash Partitions and Memory Layout for Unmanaged Web (BCM5346X)

Flash Partitions and Memory Layout for Unmanaged Plus (BCM5346X)

Flash Partitions and Memory Layout for Unmanaged Dumb (BCM5354X)

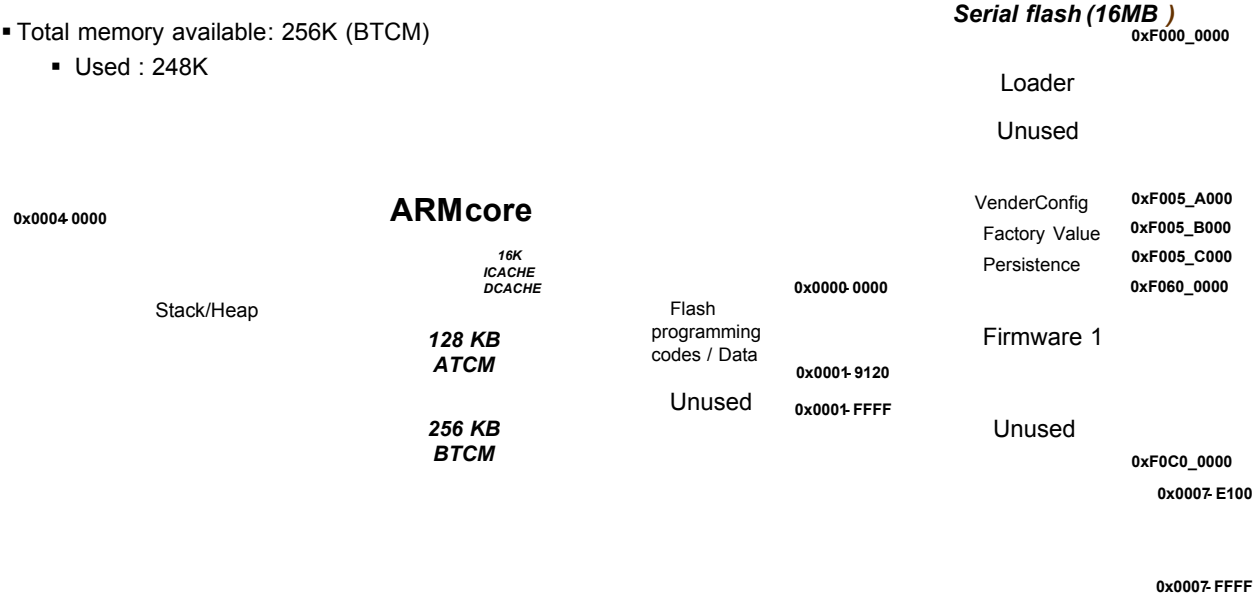
- Total memory available : 128K (L2 cache)
  - Used : 82K

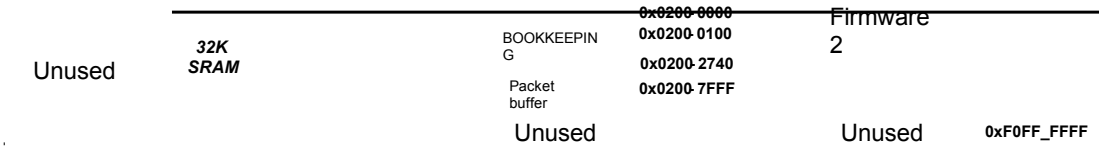



Unused
32K
SRAM
0x0200-
7FXX
0x0200-
7FFF

Flash Partitions and Memory Layout for Unmanaged Web (BCM5357X)

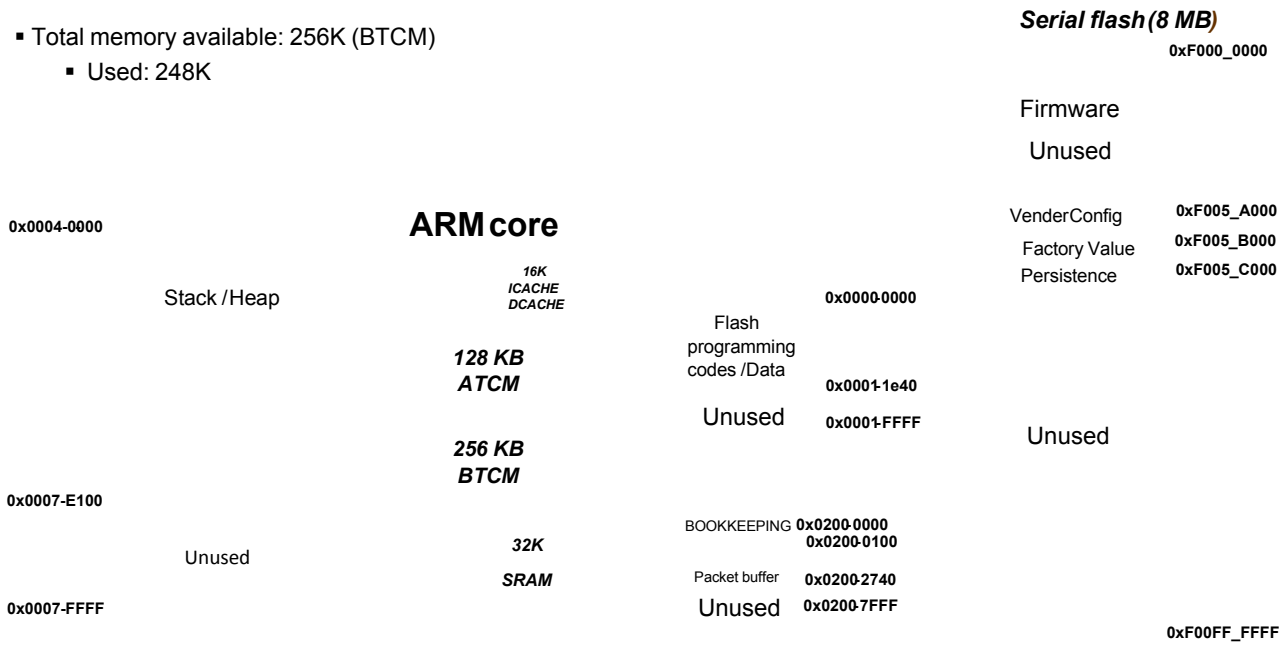
- Total memory available: 256K (BTCM)
  - Used : 248K





Flash Partitions and Memory Layout for Unmanaged Plus (BCM5357X)

- Total memory available: 256K (BTCM)
  - Used: 248K



This section describes the initialization sequence in UM software. The initialization sequence can be separated into two parts, BSP and MAIN. BSP includes core initialization (cache, MMU/MPU, timer, UART) for ARM CPU and c startup assembly code. After BSP initialization is completed, the MAIN subroutine begins. The MAIN subroutine includes the initialization items listed below. After all initializations are completed, UM software stays at an infinite `loop cli()` to process the input command and execute background handlers.

- `board_early_init()`
- `sal_init()`
- `background_init()`
- `sys_timer_init()`
- `sys_linkchange_init()`
- `net_utils_init()`
  - The subroutines above include peripheral driver and system service initialization.
- `board_init()`
  - PHY/switch initialization and a check to determine whether a firmware upgrade is required.
- `appl_init()`
  - Application execution such as CLI, DHCP, IGMP Snooping, etc.
- `persistence_init()`
  - Configuration retrieving and setting.
- `cli()`
  - A for-loop to process the input command and execute background handlers.

The boot sequence can be classified as either normal or rescue boot mode. Firmware (main application) is executed at the end of normal boot mode. The rescue boot mode is for firmware upgrade flow if the user requests a firmware upgrade.

Normal Boot Mode

Unmanaged Web includes two executable binaries: loader and firmware (main application). The loader is located at the top of SPI flash. When the boot strap is selected to boot from SPI flash the device runs the loader at first. The loader verifies the firmware. If firmware verification fails, the loader proceeds the rescue boot flow. The loader provides a web service so that the user can download the firmware image. If firmware passes the verification stage, the loader launches the firmware to execute the main application.

In comparison to Unmanaged Web, Unmanaged Plus has only one image binary: firmware (the main application). The firmware of Unmanaged Plus is located at the top of flash. The firmware is executed first and directly.

At the beginning of the firmware/loader, the code:

- Initializes the processor system.
- Configures the L2Cache as RAM (L2CRAM) for the runtime writable variables and stack.
- Initializes a switch system such as SBUS, EP/IP, LED, SerDes, external PHY, etc.
- After initialization, the system enters an infinity loop to detect events such as link status changes or reception of UART character commands.

Dual Image Boot Mode

The difference between the normal and dual image boot modes is that the latter has a duplicate of the firmware image. The reason for the dual image is to prevent side effects if the firmware upgrade flow fails. Because of the firmware duplication, the loader uses an additional step to determine if the image is up-to-date so that it can launch the correct firmware image.

Rescue Boot Mode

Rescue boot is a boot flow for firmware upgrade and is only available for the Unmanaged Web build option. The loader will proceed to the rescue boot flow if one of following conditions exists:

- Firmware verification fails.
- The boot mode in the bookkeeping area is marked as rescue instead of normal boot.

After normal boot, the user can use the web page to inform the system to download a firmware upgrade. The system changes the boot mode in the bookkeeping area to be rescue boot, and then triggers a reboot.

After the reboot, the loader checks the value of the boot mode, and then launches the rescue boot flow. At the end of the rescue boot flow, the system provides a web service. The user can use the Bonjour service for discovery of the system and then establish a HTTP connection to operations and upload firmware to set the boot mode to normal.

How to Use RealView JTAG ICE

Broadcom StrataConnect chips include an internal processor system called iProc. The processor can be controlled to debug the firmware using JTAG and RealView ICE. The UM binary runs in SPI flash. ICE cannot download the firmware directly. The firmware image should be programmed to flash by the programmer or the firmware upgrade flow. After an ICE connection is established, the code can traced step-by-step.

JTAG must be enabled through the strap pins of chip before using it. Table 1 lists a strap pin setting example for the BCM5340X.

Table 1: JTAG Strap Pin Setting for the BCM5340X

	pad_jtce1	*pad_jtce0*
ARM JTAG mode	0	1
Normal mode	0	0

Connect the Target Board

To connect ICE with the target board using a JTAG cable:

1. Power up the target board and start RealView ICE.
2. On the PC, launch **RealView Debugger** to connect ICE through an Ethernet or USB connection.
3. From the RealView Debugger menu click T**arget > Connect to Target…** to display the Connect to Target window (Figure 11).
4. In the Connect to Target window (Figure 11) select **RealView ICE** and click **Add** to display the RVConfig window.

Establish a Connection

Note: If ICE is connected through the Ethernet, enter the corresponding **IP address** (Figure 12) before clicking Connect.

In the **RVConfig window** click **Connect** (Figure 12). After the ICE connection is established, the RVConfig window will update to show the JTAG devices that were found (Figure 13 on page 17).

Enter IP address if connected via Ethernet

Configure the Connection

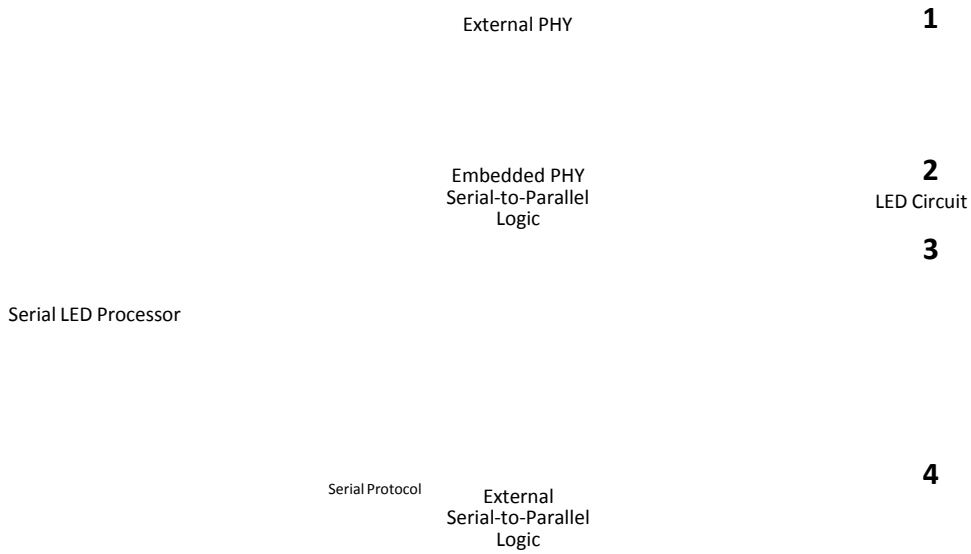
After a connection is established (see"Connect the Target Board" on page 15) follow the steps below.
In the RVConfig window (Figure 13 on page 17):

1. In the **Debug System group**: Set the clock speed to **20 KHz**
2. In the **Auto Configure group**:
   a. Verify that *Use adaptive clock if detected* is ***NOT*** selected.
   b. Click **Auto Configure** to start the JTAG device scan: be patient, it will take a few minutes. When the scan is complete each block on the JTAG scan chain is shown in the RVConfig window (Figure 13 on page 17). Go through the JTAG scan chain to find the main processor block (Cortex-A9 in the example in Figure 13 on page 17) and then close the RVConfig window.
3. **Save** the configuration and note the name of the saved file (RVI_25_0.rvc in the example in Figure 13).

4. From the **RealView Debugger main window** click T**arget > Connect to Target** to display the rvdebug.brd - Connect to Target window.
5. In the **rvdebug.brd - Connect to Target** window (Figure 14) **right-click the desired target** and, from the shortcut menu, select **Connect**.

Control Schemes

The LEDs may be controlled in several ways, depending on the LED behavior requirements and the board design. Figure 15 shows four LED control schemes.



1. The control signals come from the external PHY. The LED relative registers of the external PHY can be controlled through the external MDIO bus.
2. The control signals come from the ball-out of the StrataConnect chip and the chip setting selects the LED control signal of the embedded PHY as output. Therefore, the embedded PHY controls LED behavior. The LED relative registers of the embedded PHY can be controlled through the internal MDIO bus.
3. The control signals come from the serial LED microprocessor, pass through the internal serial-to-parallel logic, chip multiplexer, and the ball-out. The serial LED microprocessor provides more flexible and compatible LED behavior. No matter which PHY chip is selected and used, the LED behavior will not change. The specialized LED not supported by PHY may be achieved by LED microprocessor software.
4. The control signal of the LED circuit comes from the serial LED microprocessor and is passed through external serial-to-parallel logic. This scheme may provide more pin outs to control more LEDs in case of a limited number of switch chip ball-outs.

BCM953334K Case Study

The Broadcom BCM953334K is a 24-port demo switch set that includes the StrataConnect BCM53334 switch chip and an external PHY (BCM54282). This board covers schemes 1 and 3 (Figure 16). The first 16 LEDs are controlled by the BCM53334 serial LED processor. The remaining eight LEDs are controlled by the external PHY.

For details on the external PHY LED control refer to:

- Function, `boad_phy_init_callback(…)` in `$UM/systems/bcm95333x/src/board_init.c`. (LED1_SEL and LED2_SEL are set for *link* and *activity* separately in `boad_phy_init_callback(…)`. LED3_SEL and LED4_SEL are not used by the reference board)
- The BCM54282 programming reference.

For details on the serial LED processor refer to:

- Function `bcm5333x_load_led_program(void)` in `src/driver/soc/hr2switch.c`.
- LED uCode generated by LED assembly code `hr2_embedded_16x1g_1.hex` or array `ledproc_led` in `hr2_embedded_16x1g_1.c`.

The LED assembly code (`tools/led/example/hr2_embedded_16x1g_1.asm`) supports two LEDs per port: one for link (left) and the other for activity (right). The description below summarizes the implementation of the LED assembly code.

- Left and right LEDs OFF – no link
- Left LED green – link is up
- Right LED blinking green – both RX and TX
- Left and right LEDs alternately blink Green – loop detected.

The GENERIC-SERIAL-LED mechanism is available starting with UM 3.6.1, which supports the BCM5357X device. GENERIC-SERIAL-LED provides a unified interface to control the serial LED processor and the serial LED circuit on different boards. When using GENERIC-SERIAL-LED, you usually do not need to know how to write the assembly code for the board LED processor because for most cases, GENERIC-SERIAL-LED already includes a generic LED assembly code. Additionally, you do not need to know how the port status is remapped on different chips because GENERIC-SERIAL-LED handles the port status remapping with its own remapping rule. If the generic serial LED assembly code cannot meet some board requirement, you can still use the generic code as a base to create another new LED assembly code.

In general, the features of GENERIC-SERIAL-LED are as follows:

- Supports up to three LEDs per port.
- Supports up to four user-defined LED colors (up to four kinds of color).
- Supports four kinds of generic LED behaviors such as LINK, ACTIVITY, BLINK, and FORCE_ON for each LED.
- Supports a configurable timer for flexible BLINK periods and ACT extension time.

Application Interface

GENERIC-SERIAL-LED has a user interface that includes:

- A set of board APIs for initializing or changing broad parameters of serial LED scan chain.

  For example, `board_ledup_init(...)` is for passing the LED scan chain information into UM-GENERIC-SERIAL-LED then UM-GENERIC-SERIAL-LED will take care the initialization the serial LED processor and the remap table setting.
- The LED scan chain information is only associated with how the board is designed. A structure called *board_serial_led_t* is used to describe all the LED scan chain on each board and listed below.

```
typedef struct {

   /*
       Physical ports list on the scan chain for each LED processor based on board design
        For example:
        LED0:   { 2, 3, 4, 5, 10, 11, 12, 13, 35, 34, 33, 32 }
   */
  uint8 *pport_seq[LEDUP_NUMBER];

   /* Total port count for each LED scan chain */
  uint8 port_count[LEDUP_NUMBER];

   /*
       Fixed total bits for each LED scan chain,
        If this field is zero, system will fill this field as "port_count * leds_per_port *
        bits_per_led"
        For some special board design, you need to send more dump bits out.
   */
  uint8 fix_bits_total[LEDUP_NUMBER];
```

```
    /* Number of leds per port: maximum LED number per port = 3 */
    uint8 leds_per_port;

    /* Number of bits for each single LED*/
    uint8 bits_per_led;

    /* Number of color user could specify */
    uint8 num_of_color;

    /*
          Scan chain bits for controlling color for each LED,
          The number of available color is specified by num_of_color
          The number of available bits for each color is specified by bits_per_led
    */
    uint8 bits_color[4];

    /* Bits to turn off LED */
    uint8 bits_color_off;

    /* For 1~3 LEDs, each mode is two bits wide,
          in case of third LED is active, the SW link is not supported */
    ledup_mode_t led_mode[3];

    /* ledup blink period, uint=30ms */
    uint8 led_blink_period;

    /*
        For the extension of the LED-on time when LED indicate tx_rx activity
        unit=30ms
    */
    uint8 led_tx_rx_extension:4;
  }   board_serial_led_t;
```

How to Use GENERIC-SERIAL-LED

First, populate the *board_serial_led_t* structure according to board design. The following example shows how to populate the *board_serial_led_t* structure for the BCM956174R board.

```
/* Ex: BCM956174R */

/* Scan Chain on LED processor 0 */
const uint8 BCM956174R_ledup0[] =
{
   57, 56, 55, 54, 53, 52, 51, 50, /* QTC0.0 QTC0.1 */
   49, 48, 47, 46, 45, 44, 43, 42, /* QTC1.1 QTC1.1 */
   33, 32, 31, 30, 29, 28, 27, 26  /* QTC1.2 QTC1.3 */
};

/* Scan Chain on LED processor 1 */
const uint8 BCM956174R_ledup1[] =
{
   86,              /* CLPORT0 */
   82,              /* XLPORT6 */
   77, 76, 75, 74,  /* XLPORT4 */
};

const ledup_ctrl_t BCM956174R_ledup_ctrl = {
  .pport_seq = { (uint8 *)BCM956174R_ledup0, (uint8 *)BCM956174R_ledup1 },
  .port_count = { sizeof(BCM956174R_ledup0), sizeof(BCM956174R_ledup1) },
  .leds_per_port = 2,
  .bits_per_led  = 1,
  .num_of_color = 1,
  .led_blink_period = 50,     /* 30 ms * 50 = 1.5s */
  .led_tx_rx_extension = 2,  /* 2 *30ms = 60ms */
  .bits_color = {0, 0, 0, 0},
  .bits_color_off = 1,
  .fix_bits_total = {0, 0},
  .led_mode = {LED_MODE_LINK, LED_MODE_TXRX, 0},
};
```

After populating the structure, you can use board APIs to initialize the serial LED processor like the following:

```
board_ledup_init(&BCM956174R_ledup_ctrl)
```

BCM5333X Platform

The following subsections describe the front panel port numbers for the BCM953334K and BCM953394K.

- To enable auto-negotiation/KR mode on 10G ports, change the value of **CFG_CONFIG_10G_PORT_AN** to **1** in `config_loader.h / config_umweb.h / config_umplus.h / config_umdumb.h`.
- To enable AN-CL73 mode on 1G ports, change the value of **CFG_CONFIG_1G_PORT_AN** to **1** in `config_loader.h / config_umweb.h / config_umplus.h / config_umdumb.h`.

Beginning with the UM 3.2.2 release, the default option for the chip is option 1 for all SKUs.

To apply a different option, change the value of **CFG_CONFIG_OPTION** in `config_loader.h /config_umweb.h/ config_umplus.h/config_umdumb.h`.

Note: Use a vendor config to enable AN on 10G ports or AN-CL73 mode on 1G ports without revising the values of **CFG_CONFIG_10G_PORT_AN**/**CFG_CONFIG_1G_PORT_AN** and rebuilding the image.

Port Configuration

BCM953334K

Table 2 lists the front panel port numbers for the BCM953334K. Ports 1–24 are 1G copper mode with auto-negotiation.

2	4	6	8	10	12	14	16	18	20	22	24
1	3	5	7	9	11	13	15	17	19	21	23

Table 3 lists the front panel port numbers for the BCM953394K.

- Ports 1, 3, 5, 7, 9, and 11 are 1G copper mode with auto-negotiation.
- Ports 25–28 are 10G fiber mode with forced speed.
- Ports 29–32 are 1G fiber mode with auto-negotiation, clause 37.
- Ports 2, 4, 6, 8, 10, and 12 are unused.

2	4	6	8	10	12									
1	3	5	7	9	11		25	26	27	28	29	30	31	32

BCM5333X Port Mapping

The chips of the BCM5333X platform have two views of the port numbers: the logical port number view and the physical port number view. The mapping for the logical number to the physical number is in hr2switch.c.

The UM software provides the user port number. In most cases, the user port number is the same as the front panel port number. It starts at 1-basis and extends to the maximum port number supported in the hardware. It is used for the parameters of the board's APIs also. However, in some Broadcom SVK boards, the number of front panels may not be enumerated in order, and some numbers might be skipped. For example, the BCM953394K front panel port number is shown as 1–12 and 25–32, and ports 2, 4, 6, 8, 10, and 12 are unused. In such case, only the necessary front panel ports (in this case, 1, 3, 5, 7, 9, 11, and 25–32) are mapped to the user ports (1–14) in the UM software (see Table 5). In principle, the order of user ports is the same as the order of front panel ports only when the multipurpose SVK is used to verify the UM software. In the actual or customer's product, the unused ports are typically removed and are not exposed to the front panel. The mapping for the user number to the logical number is located in the functions `board_uport_to_lport()` and `board_lport_to_uport()` of board.c.

Table 4 lists the BCM953334K port mapping and Table 5 lists the BCM953394K port mapping.

Table 4: BCM953334K Port Mapping

Port Type	***Port Number***																							
Front Panel Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
User Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Logical Number	9	8	7	6	5	4	3	2	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Physical Number	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Table 5: BCM953394K Port Mapping

Port Type	***Port Number***													
Front Panel Number	1	3	5	7	9	11	25	26	27	28	29	30	31	32
User Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Logical Number	2	6	10	14	18	22	26	27	28	29	21	23	24	25
Physical Number	2	6	10	14	18	22	26	27	28	29	30	31	32	33

PHY Address

The PHY address (_phy_addr) for the BCM953334K and BCM953394K is defined in bcm956150k_miim_ext.c. The chip physical port number is used as its index. To change the mapping of _phy_addr in bcm956150k_miim_ext.c, revise the PHY address based on board specifications.

PHY Device

The PHY library supports the PHY driver for the BCM54282 and BCM54880E for the BCM5333X platform.

BCM5340X Platform

The following subsections describe the front panel port numbers for the BCM953406K and BCM53456K.

- To enable auto-negotiation/KR mode on 10G ports, change the value of **CFG_CONFIG_10G_PORT_AN** to **1** in `config_loader.h / config_umweb.h / config_umplus.h / config_umdumb.h`.
- To enable AN-CL73 mode on 1G ports, change the value of **CFG_CONFIG_1G_PORT_AN** to **1** in `config_loader.h / config_umweb.h / config_umplus.h / config_umdumb.h`.

Beginning with the UM 3.2.2 release, the default option for the chip is option 1 for all SKUs.

To apply a different option, change the value of **CFG_CONFIG_OPTION** in `config_loader.h /config_umweb.h/ config_umplus.h/config_umdumb.h`

Note: Use a vendor config to enable AN on 10G ports or AN-CL73 mode on 1G ports without revising the values of **CFG_CONFIG_10G_PORT_AN/CFG_CONFIG_1G_PORT_AN** and rebuilding the image.

Port Configuration

BCM953406K

Table 6 lists the front panel port numbers for the BCM953406K. Ports 1–12 are 1G fiber mode with auto-negotiation, clause 37. Ports 13–24 are 10G fiber mode with forced speed.

1	3	5	7	9	11	13	15	17	19	21	23
2	4	6	8	10	12	14	16	18	20	22	24

Table 7 lists the front panel port numbers for the BCM953456K.

- The left 16 ports are 1G copper mode with auto-negotiation.
- The right ports 1–8 are 1G fiber mode with auto-negotiation, clause 37.
- Right ports 9, 11, 13, and 15 are 10G fiber mode with forced speed.
- Right ports 10, 12, 14, 16, and 17–20 are unused.

Table 7: BCM953456K Front Panel Port Numbers

2	4	6	8	10	12	14	16		1	3	5	7	9	11	13	15	17	19
1	3	5	7	9	11	13	15		2	4	6	8	10	12	14	16	18	20

BCM5340X Port Mapping

The BCM53406 chip has two views of the port numbers: the logical port number view and the physical port number view. The mapping for the logical number to the physical number is in ghswitch.c.

The UM software provides the user port number. The mapping for the user number to the logical number is in the functions `board_uport_to_lport()` and `board_lport_to_uport()` of board.c.

Table 8 lists the BCM953406K port mapping and Table 9 lists the BCM953456K port mapping.

Table 8: BCM953406K Port Mapping

Port Type	*Port Number*																							
Front Panel Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
User Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Logical Number	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Physical Number	2	3	4	5	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37

Table 9: BCM953456K Port Mapping

Port Type	*Port Number*																													
Front Panel Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	1	2	3	4	5	6	7	8	9	11	13	15		
User Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28		
Logical Number	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29		
Physical Number	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	28	30	32		

PHY Address

The PHY address (_phy_addr) of the BCM953456K is defined in bcm95340xk_miim_ext.c. The chip physical port number is used as its index. To change the mapping in inbcm95340xk_miim_ext.c, revise the PHY address based on board specifications.

PHY Device

The PHY library supports the PHY driver for the following PHY devices on the BCM5340X platform: BCM54290, BCM54292, and BCM84848.

BCM5346X Platform

The following subsections describe the front panel port numbers for the BCM956270K with BCM53460 SKU.

- To enable auto-negotiation/KR mode on 10G ports, change the value of **CFG_CONFIG_10G_PORT_AN** to **1** in `config_loader.h / config_umweb.h / config_umplus.h / config_umdumb.h`.
- To enable AN-CL73 mode on 1G ports, change the value of **CFG_CONFIG_1G_PORT_AN** to **1** in `config_loader.h / config_umweb.h / config_umplus.h / config_umdumb.h`.

Note:
1. VIPER ports do not support AN-CL73 mode.
2. Use vendor config to enable AN on 10 G ports or AN-CL73 mode on 1G ports without revising the value of **CFG_CONFIG_10G_PORT_AN/CFG_CONFIG_1G_PORT_AN** and rebuilding the image.

Port Configuration

Ports 1–4 are 1G fiber mode with auto-negotiation, clause 37. Ports 5–12 are 10G fiber mode with forced speed. If the SKU is BCM53461, Ports 9-12 are 1G fiber mode with auto-negotiation by default.

Uart1	VIPER	TSC0	TSC1	MDC2/ MDIO2
Uart0	Ports 1–4	Ports 5–8	Ports 9–12	MDC1/ MDIO1

Below is the port mapping of BCM53460/BCM53461. The mapping for the user port number to the chip port number (logical port number or physical port number in UM software) is in the functions `board_uport_to_lport()` and `board_lport_to_uport()` of board.c.

Table 11 lists the BCM956270K port mapping.

Table 11: Board BCM956270K Port Mapping

Port Type	***Port Number***											
Front Panel Port Number	1	2	3	4	5	6	7	8	9	10	11	12
User Port Number	1	2	3	4	5	6	7	8	9	10	11	12
Chip Port Number	1	2	3	4	5	6	7	8	9	10	11	12

PHY Address

The PHY address (_phy_addr) of the BCM956270K is defined in bcm95346xk_miim_ext.c. The chip physical port number is used as its index. To change the mapping in bcm95346xk_miim_ext.c, revise the PHY address based on board specifications.

PHY Device

None.

BCM5354X Platform

The following subsections describe the port configuration and port mapping for the BCM953547K and BCM953549K devices. The BCM953547K supports up to 24 embedded GPHY ports and 8 SGMII ports. The BCM953549K supports up to 8 embedded GPHY ports and 8 SGMII ports.

Port Configuration

BCM953547K/BCM953549K

Figure 17 shows the front-panel port numbers for the BCM953547K. Figure 18 shows the front-panel port numbers for the BCM953549K. Left ports 1 through 24 of the BCM953547K are 1G copper mode with auto-negotiation. Right ports 1 through 8 are SGMII ports. QGPHY5 (left ports 21 through 24) of BCM953547K and SGMII 0 (right port 1 through 4) are alternative ports. Likewise, QGPHY5 (left ports 5 through 8) of BCM953549K and SGMII 0 (right port 1 through 4) are alternative ports.

Use a strap to select the usage of QGPHY5 and SGMII 0 as described in Table 12. For example with option 1, if GPHY_SGMII_SEL[0] and GPHY_SGMII_SEL[1] are set to 0, QGPHY 5 is disabled, and SGMII 0 is enabled. With this setting, the real front-port numbering becomes that shown in Figure 19: "Option 1 Front Panel Port Numbers," on page 30. As another example (option 6), if GPHY_SGMII_SEL[0] and GPHY_SGMII_SEL[1] are set to 1, QGPHY 5 is enabled, and SGMII 0 is disabled. With option 6, the real front ports numbering for option 6 is shown in Figure 24: "Option 6 Front Panel Port Numbers," on page 30.

Some of the options, like option 3 and option 4, partially enable port 1 and port 2 of QGPHY 5 and port 3 and port 4 of SGMII 0. For the detailed strap setting toward each option and the enabled ports, see Table 12.

Note: The BCM53547 and BCM53548 devices can be used on the BCM953547K reference board, while the BCM53549 device is for the BCM953549K reference board only.

Table 12: QGPHY 5 and SGMII 0 Strap Settings

Option	GPHY_SGMII_SEL[1]	GPHY_SGMII_SEL[0]	QGPHY 5				SGMII 0			
			1	2	3	4	1	2	3	4
Option 1/7/13	0	0	–	–	–	–	Sa	S	S	S
Option 2/8/14	1	1	S	S	S	S	–	–	–	–
Option 3/9/15	0	1	S	S	–	–	–	–	S	S
Option 4/10/16	0	1	S	S	–	–	–	–	S	S
Option 5/11/17	0	0	–	–	–	–	S	S	S	S
Option 6/12/18	1	1	S	S	S	S	–	–	–	–

a. S: Switch Port

BCM5354X Port Mapping

The BCM5354X has two views of the port numbers: the logical port number view and the physical port number view. The mapping from the logical number to the physical number is in src/driver/soc/bcm5354x/wh2portconf.c.

The Unmanaged software uses the user port number as an index of the port for the front-panel display and in API parameters. The mapping from the user port number to the logical port number is in the functions `board_uport_to_lport()` and `board_lport_to_uport()` in systems/bcm95354x/src/board.c.

The following tables list all the port mappings of BCM53547 options 1 through 6, BCM53548 options 7 through 12, and BCM53549 option 13 through 18. The figures in this section show the real port numbers on the front panel for each option.

Table 13: BCM53547 Options 1 through 6 Port Mapping

Port Type	***Port Number***																											
User Port Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
Logical Port Number	13	12	11	10	9	8	7	6	5	4	3	2	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
Physical Port Number	13	12	11	10	9	8	7	6	5	4	3	2	18	19	20	21	22	23	24	25	26	27	28	29	34	35	36	37

2 1G	4 1G	6 1G	8 1G	10 1G	12 1G	14 1G	16 1G	18 1G	20 1G		
1 1G	3 1G	5 1G	7 1G	9 1G	11 1G	13 1G	15 1G	17 1G	19 1G		

2 1G	4 1G	6 1G	8 1G	10 1G	12 1G	14 1G	16 1G	18 1G	20 1G	22 1G	24 1G
1 1G	3 1G	5 1G	7 1G	9 1G	11 1G	13 1G	15 1G	17 1G	19 1G	21 1G	23 1G

2 1G	4 1G	6 1G	8 1G	10 1G	12 1G	14 1G	16 1G	18 1G	20 1G	22 1G	
1 1G	3 1G	5 1G	7 1G	9 1G	11 1G	13 1G	15 1G	17 1G	19 1G	21 1G	

2 1G	4 1G	6 1G	8 1G	10 1G	12 1G	14 1G	16 1G	18 1G	20 1G	22 1G	
1 1G	3 1G	5 1G	7 1G	9 1G	11 1G	13 1G	15 1G	17 1G	19 1G	21 1G	

21 1G	23 1G	25 1G	27 1G
22 1G	24 1G	26 1G	28 1G

2 1G	4 1G	6 1G	8 1G	10 1G	12 1G	14 1G	16 1G	18 1G	20 1G	22 1G	24 1G
1 1G	3 1G	5 1G	7 1G	9 1G	11 1G	13 1G	15 1G	17 1G	19 1G	21 1G	23 1G

Table 14: BCM53548 Option 7 Port Mapping

Port Type	***Port Number***																			
User Port Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Logical Port Number	5	4	3	2	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Physical Port Number	5	4	3	2	18	19	20	21	22	23	24	25	26	27	28	29	34	35	36	37

Table 15: BCM53548 Option 8 Through 12 Port Mapping

Port Type	***Port Number***																			
User Port Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Logical Port Number	5	4	3	2	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
Physical Port Number	5	4	3	2	18	19	20	21	22	23	24	25	26	27	28	29	34	35	36	37

				2 1G	4 1G	6 1G	8 1G	10 1G	12 1G		
				1 1G	3 1G	5 1G	7 1G	9 1G	11 1G		

				2 1G	4 1G	6 1G	8 1G	10 1G	12 1G	14 1G	16 1G
				1 1G	3 1G	5 1G	7 1G	9 1G	11 1G	13 1G	15 1G

				2 1G	4 1G	6 1G	8 1G	10 1G	12 1G	14 1G	
				1 1G	3 1G	5 1G	7 1G	9 1G	11 1G	13 1G	

				2 1G	4 1G	6 1G	8 1G	10 1G	12 1G	14 1G	
				1 1G	3 1G	5 1G	7 1G	9 1G	11 1G	13 1G	

				2 1G	4 1G	6 1G	8 1G	10 1G	12 1G		
				1 1G	3 1G	5 1G	7 1G	9 1G	11 1G		

				2 1G	4 1G	6 1G	8 1G	10 1G	12 1G	14 1G	16 1G
				1 1G	3 1G	5 1G	7 1G	9 1G	11 1G	13 1G	15 1G

Table 16: BCM53549 Option 13-18 Port Mapping

Type	***Port Type***											
User Port Number	1	2	3	4	5	6	7	8	9	10	11	12
Logical Port Number	2	3	4	5	6	7	8	9	10	11	12	13
Physical Port Number	18	19	20	21	26	27	28	29	34	35	36	37

2 1G	4 1G		
1 1G	3 1G		

2 1G	4 1G	6 1G	8 1G
1 1G	3 1G	5 1G	7 1G

2 1G	4 1G	6 1G	
1 1G	3 1G	5 1G	

2 1G	4 1G	6 1G	
1 1G	3 1G	5 1G	

2 1G	4 1G		
1 1G	3 1G		

2 1G	4 1G	6 1G	8 1G
1 1G	3 1G	5 1G	7 1G

PHY Parameters

The BCM953547K and BCM953549K reference boards do not contain any external PHY chip on board. Unmanaged 3.6.x software enables the SDK PHY drivers on the BCM5354X platform. The parameters of the PHY drivers can be assigned by a vendor configuration mechanism, which is similar to the SDK property mechanism config.bcm. For example, *phy_port_primary_and_offset* must be configured to indicate the primary port and offset. Example configuration files are available in the folder $UM/systems/bcm95354x/vendor_config/.

PHY Device

None.

BCM5357X Platform

The following subsections describe the port configuration and port mapping for the BCM953570K.

Port Configuration

BCM53570 contains four kinds of port macros, including SGMIIPX4, QTC, TSCE and TSCF. Unlike before, the front panel of BCM953570K does not have the front port numbers printed on it. Instead, only the index numbers of each port macro instance are printed on the panel. Figure 37 below shows the layout of the BCM953570K front panel.

UM 3.6.1 supports BCM53570 option5_0 with 52 × 2.5G ports, 2 × 40G port plus 4 × 25G ports. SGMII0 ~ SGMII5 provide 24 × 2.5G capabilities, QTC 0 and QTC 1 provide 8 × 2.5G capabilities, and TSCE0, TSCE1, TSCE2, TSCE3 and TSCE5 provide 20 × 2.5G capabilities. TSCE4 and TSCE6 provide 2 × 40G capabilities. TSCF0 provides 4 × 25G capabilities. Other SKUs (port options) are in a preview stage for the UM 3.6.1 release.

Table 17: Port Mapping of BCM53570 Option 5_0

Front Panel	***Port Number***

BCM53570 Port Mapping

BCM53570 has two views of the port numbers, the logical port number view and the physical port number view. The mapping from the logical number to the physical number is in gh2portconf.c.

The UM software use the user port number to be an index of port for web page displaying and parameter of APIs. The mapping from the user number to the logical number is in the functions `board_uport_to_lport()` and `board_lport_to_uport()` of board.c. Table 18 below illustrates the port mapping of BCM53570 option 5_0.

Table 18: BCM53570 Option 5_0 Port Mapping

Port Type	***Port Number***																												
User Port Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
Logical Port Number	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Physical Port Number	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Port Type	***Port Number***																												
User Port Number	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58
Logical Port Number	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
Physical Port Number	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59

PHY Address and Parameters

Even though BCM953570K does not contain any external PHY chip on board, there are MDC/MDIO buses and QSFP cages allowed the connection with the external PHY boards, such as the BCM9Q_SGMIIPHY board.

UM 3.6.1 starts to enable SDK PHY drivers on the BCM53570 platform. The parameters of the PHY drivers can be assigned by a vendor configuration mechanism, which is similar to the SDK property mechanism config.bcm. For example, the config variable *port_phy_addr* could be used for assignment of the MDIO address of the external PHY. Additionally, to make the PHY work properly, some extra parameters should be given. The way to set these extra parameters depends on the implementation of the selected PHY drivers. Taking the BCM54290 as an example, the extra parameter, *phy_port_primary_and_offset*, must be configured to indicate the primary port and offset. For details regarding the BCM54290 setting, refer to BCM54290 SDK driver and $UM/systems/ bcm95357x/vendor_config/config.um.953570K_op5.

Note: Some vendor config examples about BCM5357X SKUs are placed in the folder $(UM)/systems/bcm95357x/vendor_config for reference.

PHY Device

The PHY library supports the PHY driver for the following PHY devices on the BCM5357X platform: BCM54290.

UM provides a UART console to facilitate the entry debug commands.

The commands and descriptions are listed below.

R/W Memory Mapped Register or Device Memory

d dumps the memory-mapped register or device memory. The internal SRAM is mapped to 0x1B000-0000 to 0x1B00-7FFF on the BCM5333X platform.

Example: Write and read SRAM on the BCM5333X:

```
CMD> e - Edit contents of memory
Address: 0x1B000000
    Press ESC or Ctrl-C to quit editing.
    1B000000 [EE]: 0xEF

CMD> d - Dump memory
Address: 0x1B000000
    Length: 0x1

    1B000000: EF
```

Read/Write the SBUS Register

The SBUS register can be accessed indirectly through a memory mapped register. A block number is defined at the top of the chip specification and is required before the SBUS register is accessed.

Example: Write and read SBUS register IARB_TDM_CONTROL:

```
CMD> s - Set value of switch register
Block: 0xa
Addr: 0x02000900
Value: 0x10
Done

CMD> g - Get value of switch register
Block: 0xa
Addr: 0x02000900
Value: 0x00000010
```

Read/Write SBUS Table

SBUS table access is similar to SBUS register access: the difference is the SBUS command op-code. A block number and index are needed before SBUS table access. The index is the offset address of the desired entry.

Example: Write and read SBUS table L2_USER_ENTRY:

```
CMD> r - Dump table
Block: 0xA
Address: 0x1C100000
    Index: 0x0
Length: 0x6
Value:  0x84000003  0x00000000  0x80000040  0xFFC06030  0x000807FF  0x00000004

CMD> w - Edit contents of table
Block: 0xA
Address: 0x1C100000
    Index: 0x0
Length: 0x6
DWord [00:0x84000003]: 0x84000000
DWord [01:0x00000000]: 0x3
DWord [02:0x80000040]: 0x
DWord [03:0xffc06030]: 0x
DWord [04:0x000807ff]: 0x
DWord [05:0x00000004]: 0x
Done

CMD> r - Dump table
Block: 0xA
Address: 0x1C100000
    Index: 0x0
Length: 0x6
Value:  0x84000000  0x00000003  0x80000040  0xFFC06030  0x000807FF  0x00000004
```

Read/Write PHY Register

The PHY register can be accessed with a given user port number as shown below. See "Port and PHY Configuration" on page 23 for the user port number. Ports containing a SerDes inside may provide an option for selecting the SerDes type.

```
CMD> p - Get value of PHY register
User port: 1
 0: bcm56150 addr=0x1 on bus 0
Enter your choice: [0]
Reg address: 0x4
PHY(Port 1) : Reg:0x4  Value:0x000005E1

CMD> q - Set value of PHY register
User port: 1
 0: bcm56150 addr=0x1 on bus 0
Enter your choice: [0]
reg address: 0x4
data: 0x1e1
PHY(Port 1) : Reg:0x4  Value:0x000001E1

CMD> p - Get value of PHY register
User port: 1
 0: bcm56150 addr=0x1 on bus 0
Enter your choice: [0]
Reg address: 0x4
PHY(Port 1) : Reg:0x4  Value:0x000001E1

CMD> p - Get value of PHY register
User port: 18
 0: bcm5428(9)2 addr=0x15 on bus 0
 1: bcmi_qsgmii_serdes addr=0x11 on bus 1
Enter your choice: [0]
Reg address: 0x4
PHY(Port 18) : Reg:0x4  Value:0x000005E1
```

Snake Test Command

The Unmanaged software provides the following snake test modes.

1. Internal MAC loopback
2. Internal PHY loopback
3. External loopback
4. Port pair loopback
5. Snake settings for packet generator device.

> **Note:** Reboot the system is required to recover the settings from the snake test once the test is complete.

The BCM5333X platform is used to describe the details of each mode in the sections below.

Internal MAC/PHY Loopback

For mode 1 and mode 2, the Unmanaged software provides the snake test with internal MAC or PHY loopback. Remove the cable from all testing ports. In this mode, the user can assign any user port number for the minimum and maximum port number before testing. The total testing ports (maximum port number – minimum port number) must be larger than or equal to two. The user can input the duration time for the snake test, but the duration time must be in multiples of 10 seconds. The default duration is 60 seconds.

At the beginning of the test, the CPU sets the path for the snake test, which includes both the VLAN and loopback modes. Next, the CPU sends two packets to the minimum and maximum port. The packets then go through the snake path for the set duration time. At the end of the test, the CPU attempts to receive these two packets and cuts the path. The CPU compares the packet content and statistics for each testing port. Finally, the test shows either `Passed` or `Failed` as the result.

```
CMD> k - Snake Test
Run mode:
0: Internal MAC loopback
- Must disconnect cable from all testing ports
- Total testing ports must be larger than or equal to 2
1: Internal PHY loopback
- Must disconnect cable from all testing ports
- Total testing ports must be larger than or equal to 2
2: External PHY loopback
- Must connect loopback cable from all testing ports
- Total testing ports must be larger than or equal to 2
3: Port pair loopback
- Min port must start from an odd number, such as 1
- Total testing ports must be larger than or equal to 2
- Total testing ports must be the multiple of 2
4: Do snake setting for packet generator device
- Min port must start from an odd number, such as 1
- Total testing ports must be larger than or equal to 4
- Total testing ports must be the multiple of 2
Enter your choice: [0]
Enter min port: [1]
Enter max port: [24] 4
Enter duration (seconds) - must be the multiple of 10: [60]

lport 2 (P:9), speed = 1000, duplex = 1, tx_pause = 0, rx_pause = 0 an=0 bcm56150

lport 3 (P:8), speed = 1000, duplex = 1, tx_pause = 0, rx_pause = 0 an=0 bcm56150

lport 4 (P:7), speed = 1000, duplex = 1, tx_pause = 0, rx_pause = 0 an=0 bcm56150

lport 5 (P:6), speed = 1000, duplex = 1, tx_pause = 0, rx_pause = 0 an=0 bcm56150

Snake Test: Start circular test on all testing ports for 60 seconds.
Snake Test: Time elapsed 10 seconds.
Snake Test: Time elapsed 20 seconds.
Snake Test: Time elapsed 30 seconds.
Snake Test: Time elapsed 40 seconds.
Snake Test: Time elapsed 50 seconds.
Snake Test: Time elapsed 60 seconds.
Snake Test: 2 packets received by CPU.

Snake Test: Statistics for each port
```

```
Port 01    TX 254692 pkt  17319056 byte,  RX  254692 pkt  17319056 byte
Port 02    TX 254692 pkt  17319056 byte,  RX  254692 pkt  17319056 byte
Port 03    TX 254692 pkt  17319056 byte,  RX  254692 pkt  17319056 byte
Port 04    TX 254693 pkt  17319124 byte,  RX  254693 pkt  17319124 byte
Snake Test: Passed.
Please reboot system to recover setting from snake test !
```

Port Pair Loopback

For mode 3, Unmanaged software provides the snake test via the port pair cable, such as (1, 2), (3, 4) and (5, 6) in the following example. Connect the cable with port 1 and port 2, port 3 and port 4, and port 5 and port 6 in following example. In this mode, the user can assign any user port number for the minimum and maximum port number before testing. The total testing ports (maximum port number – minimum port number) must be in multiples of 2. The minimum port number must start from an odd number, such as 1 or 3. The user can also input the duration time for snake test, but the duration time must be in multiples of 10 seconds. The default duration is 60 seconds.

At the beginning of the test, the CPU sets the path for the snake test, which includes VLAN mode after connecting the external cable. The CPU then sends two packets to the minimum and maximum ports. The packets go through the snake path for the set duration time. At the end of test, the CPU tries to receive these two packets and cuts the path. The CPU then compares the packet content and statistics for each testing port. Finally, the test shows either `Passed` or `Failed` as the result.

```
CMD> k - Snake Test
Run mode:
0: Internal MAC loopback
   - Must disconnect cable from all testing ports
   - Total testing ports must be larger than or equal to 2
1: Internal PHY loopback
   - Must disconnect cable from all testing ports
   - Total testing ports must be larger than or equal to 2
2: External PHY loopback
   - Must connect loopback cable from all testing ports
   - Total testing ports must be larger than or equal to 2
3: Port pair loopback
   - Min port must start from an odd number, such as 1
   - Total testing ports must be larger than or equal to 2
   - Total testing ports must be the multiple of 2
4: Do snake setting for packet generator device
   - Min port must start from an odd number, such as 1
   - Total testing ports must be larger than or equal to 4
   - Total testing ports must be the multiple of 2
Enter your choice: [0] 3
Enter min port: [1]
Enter max port: [24] 6
Enter duration (seconds) - must be the multiple of 10: [60] 20
Please connect cable for port pair (1, 2) (3, 4) (5, 6).

Snake Test : Port 1 is link down, please connect cable to it !

lport 6 (P:5), speed = 1000, duplex = 1, tx_pause = 1, rx_pause = 1 an=1 bcm56150

lport 7 (P:4), speed = 1000, duplex = 1, tx_pause = 1, rx_pause = 1 an=1 bcm56150
```

```
lport 8 (P:3), speed = 1000, duplex = 1, tx_pause = 1, rx_pause = 1 an=1 bcm56150

lport 9 (P:2), speed = 1000, duplex = 1, tx_pause = 1, rx_pause = 1 an=1 bcm56150

Snake Test : Port 5 is link down, please connect cable to it !

lport 4 (P:7), speed = 1000, duplex = 1, tx_pause = 1, rx_pause = 1 an=1 bcm56150

lport 5 (P:6), speed = 1000, duplex = 1, tx_pause = 1, rx_pause = 1 an=1 bcm56150

Snake Test : Start circular test on all testing ports for 20 seconds.

Snake Test : Time elapsed 10 seconds.

Snake Test : Time elapsed 20 seconds.

Snake Test : 2 packets received by CPU.

Snake Test : Statistics for each port
Port  01    TX 1619212 pkt  110099960 byte,  RX  1482648 pkt  100820064 byte.
Port  02    TX 1482648 pkt  100820064 byte,  RX  1619212 pkt  110099960 byte.
Port  03    TX 1617598 pkt  109996664 byte,  RX  1482648 pkt  100820064 byte.
Port  04    TX 1482648 pkt  100820064 byte,  RX  1617598 pkt  109996664 byte.
Port  05    TX 1617598 pkt  109996664 byte,  RX  1484268 pkt  100923744 byte.
Port  06    TX 1484268 pkt  100923744 byte,  RX  1617598 pkt  109996664 byte.

Snake Test : Passed.

Please reboot system to recover setting from snake test !
```

Snake Test Settings for the Packet Generator Device

For mode 4, the Unmanaged software provides the snake path without sending the packet from the CPU. In this mode, the user must connect the ports with an external cable. As shown in the following example, connect a cable for each port pair (2, 3) (4, 5). In this mode, the user can assign any user port number for the minimum and maximum port number before testing. The total testing ports (maximum port number – minimum port number) must be larger than or equal to 4 and in multiples of 2. The minimum port number must start from an odd number, such as 1 or 3. The user can also input the duration time for the snake test, but the duration time must be in multiples of 10 seconds. The default duration is 60 seconds.

In the beginning of the test, the CPU sets the path for the snake test, which includes VLAN mode after connecting an external cable. The user can then inject a packet to the minimum and maximum port number from the packet generator device and check whether the statistics are correct from the packet generator device.

```
CMD> k - Snake Test
Run mode :
0: Internal MAC loopback
   - Must disconnect cable from all testing ports
   - Total testing ports must be larger than or equal to 2
1: Internal PHY loopback
   - Must disconnect cable from all testing ports
   - Total testing ports must be larger than or equal to 2
2: External PHY loopback
```

```
- Must connect loopback cable from all testing ports
- Total testing ports must be larger than or equal to 2
3: Port pair loopback
- Min port must start from an odd number, such as 1
- Total testing ports must be larger than or equal to 2
- Total testing ports must be the multiple of 2
4: Do snake setting for packet generator device
- Min port must start from an odd number, such as 1
- Total testing ports must be larger than or equal to 4
- Total testing ports must be the multiple of 2
Enter your choice: [0] 4
Enter min port: [1] 1
Enter max port: [24] 6
Please connect cable for port pair (2, 3) (4, 5).

Please connect cable with packet generator device for ports 1 and 6.

Do sanke setting for packet generator device.

Snake Test : Port 1 is link down, please connect cable to it !

lport 4 (P:7), speed = 1000, duplex = 1, tx_pause = 1, rx_pause = 1 an=1 bcm56150

lport 5 (P:6), speed = 1000, duplex = 1, tx_pause = 1, rx_pause = 1 an=1 bcm56150

lport 6 (P:5), speed = 1000, duplex = 1, tx_pause = 1, rx_pause = 1 an=1 bcm56150

lport 7 (P:4), speed = 1000, duplex = 1, tx_pause = 1, rx_pause = 1 an=1 bcm56150

lport 8 (P:3), speed = 1000, duplex = 1, tx_pause = 1, rx_pause = 1 an=1 bcm56150

lport 9 (P:2), speed = 1000, duplex = 1, tx_pause = 1, rx_pause = 1 an=1 bcm56150

User can inject packet to port 1 and 6 from packet generator device now !

Please reboot system to recover setting from snake test !
```

NVRAM Command

The `nvram` command is used to set/get/list/remove/commit the configuration of `vendor config`. It is for debug purpose only. The `nvram` command is under the `Flash` command as shown below.

```
CMD> F - Flash utilities
  f - Write factory mac address
  n - Write the serial number
  d - Dump the mac address and serial number
  s - Set nvram variable
  g - Get nvram variable
  l - list all nvram variable
  r - Remove nvram variable
  c - Commit nvram variable bindings
Enter your choice:
```

Example

For example, if user needs to add a `config "qtc_interface=2"` to set the QTC ports to SGMII mode on the BCM953434K board. The user can use the `F->s` command to set it.

```
CMD> F - Flash utilities
  f - Write factory mac address
  n - Write the serial number
  d - Dump the mac address and serial number
  s - Set nvram variable
  g - Get nvram variable
  l - list all nvram variable
  r - Remove nvram variable
  c - Commit nvram variable bindings
Enter your choice: s
Name: qtc_interface
Value: 2
```

Use the `F->l` command to list it.

```
CMD> F - Flash utilities
  f - Write factory mac address
  n - Write the serial number
  d - Dump the mac address and serial number
  s - Set nvram variable
  g - Get nvram variable
  l - list all nvram variable
  r - Remove nvram variable
  c - Commit nvram variable bindings
Enter your choice: l
qtc_interface=2
```

Use the `F->c` command to save the `nvram` variable into flash. If a user executes a `commit nvram` command, the system will erase the persistence section automatically because it may cause changes in the persistence size.

```
CMD> F - Flash utilities
   f - Write factory mac address
   n - Write the serial number
   d - Dump the mac address and serial number
   s - Set nvram variable
   g - Get nvram variable
   l - list all nvram variable
   r - Remove nvram variable
   c - Commit nvram variable bindings
Enter your choice: c
The nvram variables you commit may cause persistence size changed.
System will erase the persistence section automatically.
Are you sure to continue (y or n): y

Please reboot the device to allow the commit to take effect.
```

With the SPI flash autoprobe feature, a single unmanaged image may support multiple predefined SPI flashes. The method of autoprobing the SPI flash type is based on the JEDEC ID of the flash. The process consists of reading the JEDEC ID from the external flash when system boots-up and determining if this ID can match the ID pattern of the desired flash driver in the flash supporting table. The condition of ID matching is shown below:

```
(flash_driver's ID & ID mask == external flash ID & ID mask)
```

If the ID cannot match any drivers after the autoprobe, the flash driver will be set as `Unknown Flash` driver which does not allow any write operation to flash. The flash autoprobe mechanism is enabled by default.It can be disabled by specifying a particular flash driver, otherwise the autoprobe will be performed during every boot up period.

Supported Flash Table

The supported flash table, `flash_supporting_table`, includes all supported flash drivers and is placed at `$UM/src/drivers/flash/flash_table.h`. Each flash driver is defined by a structure, named `flash_dev_t`, which includes the ID pattern for ID matching, programming functions, and other details. The developer can check the `flash_table.h` to see if the desired flash is supported. Beginning with um-3.3.0, the Unmanaged software supports five flashes as listed in Table 19.

Table 19: Supported Flash Table

Supported Flash Name	*Manufacturer*	*Comment*
MX25L128	MACRONIX (MXIC)	–
MX25L256	MACRONIX (MXIC)	–
W25Q64	WINBOND	–
N25Q256	MICRON	–
N25Q512	MICRON	It is not supported in the BCM5333X platform.

Adding a New Flash into the Table

To add a new SPI flash, the data sheet of the new flash is required. After reading the data sheet, fill out each field of `flash_dev_t` and put it into the `flash_supporting_table`.

Note: The JEDEC ID matching is executed from top to bottom of the flash supporting table. The last flash drivers in the supported flash table is always `Unknown Flash`, keep the `Unknown Flash` at the bottom of the table.

The fields and descriptions of `flash_dev_t` are shown below.

```
typedef struct flash_dev_s {
   uint8 jedec_id[4];                          /* JEDEC ID for flash probe */
   uint8 jedec_id_mask[4];                     /* JEDEC ID mask to mask unwanted field of JEDEC
   ID */
   const struct flash_dev_funs   *funs;        /* Function pointers */
   uint32                        flags;        /* Device characteristics */
   hsaddr_t                      start;        /* First address */
   hsaddr_t                      end;          /* Last address */
   uint32                        num_block_infos;  /* Number of entries */
   const flash_block_info_t      *block_info;  /* Info about one block size */
   const char *                  name;
   const void                    *priv;        /* Devices private data */
} flash_dev_t;
```

- `Jedec_id`: This is the ID of the JEDEC ID of flash. It can be found in the SPI flash data sheet.
- `Jedec_id_mask`: This is used to ignore less meaningful bits in the JEDEC ID.
- `Funs` and `flags`: This implements the programming procedures. The flags specify special information such as `start`, `end`, `num_block_infos` and `block_info`. They specify the starting and ending address of the flash, as well as each erase block size. A SPI flash may have more than one type of `block_info`, so it also defines how many types of `block_info`.
- `Name`: This is the flash or driver name It will be shown as system boot up.
- `Priv`: This is reserved for a specific data structure of flash driver.

Verifying a New Flash Driver

After adding a new SPI flash into the supported table, check the boot up log to see if the flash name is correct and a heavy regression test of flash programming is recommended.

An example boot up log is shown below to demonstrate the successful result of flash autoprobing.

```
HURRICANE3 loader-3.3.0
……
Flash detected: W25Q64CV
devid = 0x8434, revid = 0x1
```

An example boot-up log is shown below to demonstrate a failed case of flash autoprobing.

```
HURRICANE3 loader-3.3.0
……
Flash detected: Unknown Flash
ID: 20 ba 20 10
Please check the flash supporting table
```

The failure of a flash autoprobe is regarded as a fatal error. This should be fixed as soon as possible.

Configure VLANs by Board APIs

For the Unmanaged Web, the default setting for the VLAN type is IEEE 802.1Q. The VLAN type can be changed from IEEE 802.1Q to port-based VLAN by calling `board_vlan_type_set(VT_PORT_BASED)`. The VLAN type can also be changed from the port-based to IEEE 802.1Q by calling `board_vlan_type_set(VT_DOT1Q)`. Changing the VLAN type loses all the VLAN-related configurations:

- `sys_error_t board_vlan_type_set(vlan_type_t type)` API is used to set the VLAN type. The type can be VT_PORT_BASED or VT_DOT1Q.
- `sys_error_t board_vlan_type_get(vlan_type_t *type)` API is used to get the current VLAN type. The type can be VT_PORT_BASED or VT_DOT1Q.
- `uint16 board_vlan_count(void)` API is used to get the numbers of the existing VLANs.

IEEE 802.1Q VLAN

Use `board_vlan_create()` to create an IEEE 802.1Q VLAN. This API sets the valid bit and spanning tree group ID to one in the VLAN table and the egress VLAN table.

Use `board_vlan_destroy()` to destroy a VLAN for a given VLAN ID. It removes the entries from both the VLAN table and the egress VLAN table.

To modify a member of an existing VLAN, call `board_qvlan_port_set()` for a given VLAN ID. If the VLAN with this ID does not exist, it returns SYS_ERR_NOT_FOUND. If the VLAN exists, this API modifies the member port list and the tagged member port list into VLAN and egress VLAN tables.

The `board_qvlan_port_get()` is used to get members for a given VLAN ID. If the VLAN with this ID does not exist, it returns SYS_ERR_NOT_FOUND.

```
board_vlan_create (uint16 vlan_id)
board_vlan_destroy (uint16 vlan_id)
board_qvlan_port_set (uint16 vlan_id, uint8 *uplist, uint8 *tag_uplist)
   uplist is the member port list
   tag_uplist is tagged port list
board_qvlan_port_get (uint16 vlan_id, uint8 *uplist, uint8 *tag_uplist)
   uplist is the member port list
   tag_uplist is tagged port list
```

Port-Based VLAN

Use `board_vlan_create()` to create a port-based VLAN and `board_vlan_destroy()` to remove a port-based VLAN. The `board_vlan_destroy()` API cleans the ports in the egress mask table for a given VLAN ID.

To modify the members of an existing VLAN, call `board_pvlan_port_set()` for a given VLAN ID. If the VLAN with this ID does not exist, it returns SYS_ERR_NOT_FOUND. If it does exist, this API modifies the member port list in the egress mask table.

The board_pvlan_port_get() is used to get members for a given VLAN ID from the egress mask table. If a VLAN with this ID does not exist, it returns SYS_ERR_NOT_FOUND.

```
board_vlan_create (uint16 vlan_id)
board_vlan_destroy(uint16 vlan_id)
board_pvlan_port_set(uint16 vlan_id, uint8 *uplist)
   uplist is the member port list
board_pvlan_port_get(uint16 vlan_id, uint8 *uplist)
   uplist is the member port list
```

Case Study 1—Create a 802.1Q VLAN

This example creates an IEEE 802.1Q VLAN with port 1 (untag member), port 2 (untag member), port 3 (tag member), and port 4 (tag member) on the BCM5333X platform.

Example:

```
int vid =2;
uint8 uplist[3] = {0x0F, 0x0, 0x0}, tag_uplist[3] = {0x0C, 0x0, 0x0};
board_vlan_create(vid);
board_qvlan_port_set(vid, uplist, tag_uplist);
```

Case Study 2—Create a Port-based VLAN

This example creates a port-based VLAN with port 1–port 4 on the BCM5333X platform.

Example:

```
int vid =2;
uint8 uplist[3] = {0x0F, 0x0, 0x0};
board_vlan_create(vid);
board_pvlan_port_set(vid, uplist);
```

Configuring L2 Entries by L2 Board APIs

UM provides L2 Board APIs to add/delete/lookup/get the addresses of L2 Entry table.

```
board_l2_addr_add(board_l2_addr_t *l2addr);
board_l2_addr_delete(board_l2_addr_t *l2addr);
board_l2_addr_delete_by_port(uint16 uport);
board_l2_addr_delete_by_vlan(uint16 vid);
board_l2_addr_delete_by_trunk(uint8 lagid);
board_l2_addr_delete_all(void);
board_l2_addr_lookup(board_l2_addr_t *l2addr, uint16 *index);
board_l2_addr_get_first(board_l2_addr_t *l2addr, uint16 *index);
board_l2_addr_get_next(board_l2_addr_t *l2addr, uint16 *index);
board_l2_addr_get_last(board_l2_addr_t *l2addr, uint16 *index);
```

board_l2_addr_t are defined as below.

```
typedef struct board_l2_addr_s {
  uint16 flags;   /* BOARD_L2_xxx flags. */
  uint8  mac[6];  /* 802.3 MAC address. */
  uint16 vid;     /* VLAN identifier. */
  uint16 uport;   /* user port number. */
  uint16 mcidx;   /* L2MC pointer. */
  uint8  tgid;    /* Trunk ID value. */
  uint8  is_trunk;  /* Trunk bit. */
} board_l2_addr_t;
```

Case Study 1—Create an L2 Entry

To add an L2 entry with MAC address 00-00- 00-01- 02-03 and vid = 1 at user port #1. Users need to assign `mac/vid/uport` fields of board_l2_addr_t, then invoke `board_l2_addr_add` as below.

```
board_l2_addr_t l2addr;
l2addr.mac[0] = 0x00;
l2addr.mac[1] = 0x00;
l2addr.mac[2] = 0x00;
l2addr.mac[3] = 0x01;
l2addr.mac[4] = 0x02;
l2addr.mac[5] = 0x03;
l2addr.vid = 1;
l2addr.uport = 1;
board_l2_addr_add(&amp;l2addr);
```

If there are any L2 entry matched `mac/vid/uport` fields already existing, the function returns `SYS_ERR_EXISTS`, otherwise it returns `SYS_OK`.

Register a Link Change Callback

If a function needs to be called every time a link change happens, then a function needs to be registered to the link-change handler. For example, create a `board_customer_linkchange()` function first and register this function to the link-change handler by calling `sys_register_linkchange()` as shown in the following code:

```
BOOL sys_register_linkchange(SYS_LINKCHANGE_FUNC func, void *arg)

   void board_customer_linkchange(uint16 uport, BOOL link, void *arg)
   {
      ......
      if (link) {
      ......
      } else {
      ......
      }
   }
   sys_register_linkchange(board_customer_linkchange, NULL);
```

Revision	*Date*	*Change Description*
	01/24/18	**Updated:** • Table 17: "Port Mapping of BCM53570 Option 5_0," on page 34 • "BCM53570 Port Mapping" on page 35 **Added:** • "Flash Partitions and Memory Layout for Unmanaged Dumb (BCM5354X)" on page 10 • "GENERIC-SERIAL-LED" on page 20 • "BCM5354X Platform" on page 28 to "Port and PHY Configuration" on page 23
	09/20/17	**Updated:** • "Configuring L2 Entries by L2 Board APIs" on page 37 **Added:** • "Flash Partitions and Memory Layout for Unmanaged Web (BCM5357X)" on page 9 • "Flash Partitions and Memory Layout for Unmanaged Plus (BCM5357X)" on page 10 • "BCM5357X Platform" on page 23 to "Port and PHY Configuration" **Removed:** • from "Memory Layout" on page 5 – Flash Partitions and Memory Layout for Unmanaged Web (BCM5343X) – Flash Partitions and Memory Layout for Unmanaged Plus (BCM5343X)

Revision	*Date*	*Change Description*
	05/27/16	**Updated:** • "Purpose and Audience" on page 10 • Figure 2: "Flash Partitions and Memory Layout for Unmanaged Web (BCM5333X)," on page 12 • Figure 3: "Flash Partitions and Memory Layout for Unmanaged Plus (BCM5333X)," on page 13 • Figure 4: "Flash Partitions and Memory Layout for Unmanaged Web (BCM5340X)," on page 13 • Figure 5: "Flash Partitions and Memory Layout for Unmanaged Plus (BCM5340X)," on page 14 • Figure 6: "Flash Partitions and Memory Layout for Unmanaged Web (BCM5343X)," on page 14 • Figure 7: "Flash Partitions and Memory Layout for Unmanaged Plus (BCM5343X)," on page 15 • "BCM5333X Platform" on page 24 • "BCM5340X Platform" on page 26 • "BCM5343X Platform" on page 28 • "Snake Test Command" on page 33 • "Internal MAC/PHY Loopback" on page 34 • "Port Pair Loopback" on page 35 • "Snake Test Settings for the Packet Generator Device" on page 36 • "Case Study 1—Create a 802.1Q VLAN" on page 42 • "Case Study 2—Create a Port-based VLAN" on page 42 **Added:** • Figure 8: "Flash Partitions and Memory Layout for Unmanaged Web (BCM5346X)," on page 15 • Figure 9: "Flash Partitions and Memory Layout for Unmanaged Plus (BCM5346X)," on page 16 • "BCM5346X Platform" on page 29
	01/27/16	**Updated:** • "BCM5343X Platform" on page 24 **Added:** • "Programming Reference" on page 35 **Removed:** • Minimum Storage Requirements

Revision	*Date*	*Change Description*
	07/30/15	**Updated:** • "Memory Layout: Introduction" on page 9 • Figure 2: "Flash Partitions and Memory Layout for UM-Web (BCM5333X)," on page 10 • Figure 3: "Flash Partitions and Memory Layout for UM+ (BCM5333X)," on page 10 • Figure 4: "Flash Partitions and Memory Layout for UM-Web (BCM5340X)," on page 11 • Figure 5: "Flash Partitions and Memory Layout for UM+ (BCM5340X)," on page 11 • "BCM953334K Case Study" on page 20 • "BCM5333X Platform" on page 21 • "BCM5340X Platform" on page 23 **Added:** • Figure 6: "Flash Partitions and Memory Layout for UM-Web (BCM5343X)," on page 12 • Figure 7: "Flash Partitions and Memory Layout for UM+ (BCM5343X)," on page 12 • "BCM5343X Platform" on page 25 • "Snake Test Command" on page 30 • "NVRAM Command" on page 34 • "SPI Flash Driver Autoprobing" on page 36 • "Vendor Configuration" on page 39
	02/25/15	**Updated:** • Figure 2: "Flash Partitions and Memory Layout for UM-Web (BCM5333X)," on page 9 • Figure 3: "Flash Partitions and Memory Layout for UM+ (BCM5333X)," on page 9 • Figure 4: "Flash Partitions and Memory Layout for UM-Web (BCM5340X)," on page 10 • Figure 5: "Flash Partitions/Memory Layout for UM+ (BCM5340X)," on page 10 • "Initialization Sequence" on page 11 • "Port and PHY Configuration" on page 19 • "BCM5333X Port Mapping" on page 20 • "BCM5340X Port Mapping" on page 22 • "Minimum Storage Requirements" on page 23 **Added:** • "Initialization Sequence" on page 11 • "Read/Write PHY Register" on page 26
	12/18/14	Initial release.

Corporate Headquarters: San Jose, CA

Unmanaged-PG106 January 24, 2018