ENT-AN1007-4.3

Application Note

Microsemi Ethernet API Software

Released

October 2018

Contents

Tables

Figures

1 Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

Revision 1.2

Revision 1.2 was published in October 2018. In revision 1.2, the proprietary and confidential restrictions were removed. There were no changes to the technical content.

Revision 1.1

Revision 1.1 was published in January 2018. The following is a summary of changes in revision 1.1 of this document.

- Removed references and sections related to OTN mapper products from the guide.
- Replaced references to Jaguar-1 with Jaguar-2 throughout the guide.
- Updated the directory structure table. For more information, see *Directory Structure* on page 3.
- Updated the API targets table. For more information, see *Targets* on page 3.
- Updated the application example platforms table. For more information, see *Application Examples* on page 13.

Revision 1.0

Revision 1.0 was published in November 2016. It was the first publication of this document.

2 Microsemi API Software

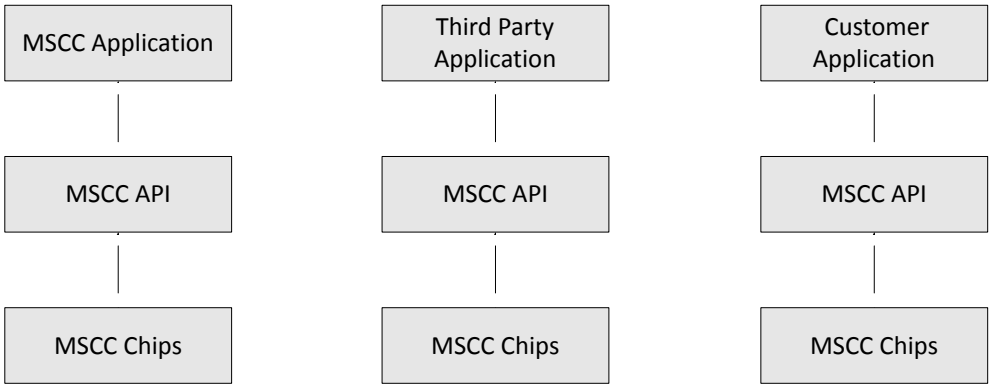
The Microsemi unified application programming interface (API) provides a comprehensive, user friendly, and robust function library that supports all Microsemi Ethernet switch and PHY. The unified API is portable to any operating system (OS) and was developed with 32-bit and 64-bit CPUs as intended targets. The driver software was developed in standard C and supports multi-instance device targets. This document explains the code structure and porting process along with application examples to assist in rapid development and system deployment.

2.1 Overview

The Microsemi API provides portable driver software for Microsemi switch and PHY mapper products. It can be used as a basis for the following application software solutions.

- Microsemi application software used for production and demonstration
- Third party application software provided by a partner company
- Customer application software developed by customers using the Microsemi API

Figure 1 • Microsemi API Applications



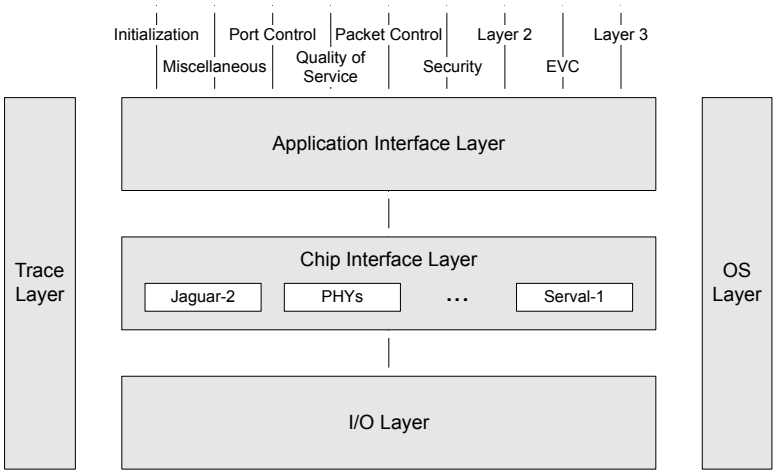
2.1.1 API Architecture

The API architecture consists of the following layers.

- Application interface layer provides a C interface to the application layer. Functions are arranged in groups.
- Chip interface layer maps function parameters to chip-specific register accesses.
- I/O layer provides register access. This layer is platform dependent and is implemented outside the API.
- OS layer adapts the API to a given OS.

- Trace layer maps code trace macros for debugging purposes.

Figure 2 • Microsemi API Architecture



2.1.2 Directory Structure

The following table shows the subdirectories that organize the API source. Application example code is included in separate subdirectories.

Table 1 • Directory Structure

Directory	Description
`vtss_api/bin/mips/doxygen`	Automatically generates documentation of the public header files. The directory includes a document for each product.
`vtss_api/include`	Public (external) header files. The application must include only vtss_api.h, which includes all other required header files.
`vtss_api/base`	API implementation and private (internal) header files arranged in a number of subdirectories. The application build system (for example, makefile) should compile all C files in this directory, including subdirectories.
`vtss_api/boards`	Reference implementation of board-specific code. Used by the Linux kernel module example.
`vtss_api/appl`	Application example implementations, running mainly as userspace Linux applications.
`vtss_api/linux_support`	Example user-mode IO (UIO) Linux drivers for supporting running the switch API on external CPU systems through PCIe, and so on.

2.1.3 Targets

The API must be compiled for one or more target chips by defining one or more of the symbols in the following table. This selection has the following effects.

- The external header files will include the functions and data types supported by the selected targets.
- The application interface layer will include the functions supported by the selected targets.
- The chip interface layer will be included for the selected targets.

The VTSS_CHIP_CU_PHY target may be used when compiling the API for a PHY-only application.

When compiling the API for switch or MAC, the PHY part is always included. For example, if the VTSS_CHIP_JAGUAR_2 target is selected, PHYs connected to the MII management controller of the switch chip can be controlled using the PHY portion of the API.

Table 2 • API Targets

Target	Part Number
VTSS_CHIP_CU_PHY	VSC8211 VSC8641 VSC8221 VSC8664 VSC8224 VSC8502 VSC8234 VSC8504 VSC8244 VSC8514 VSC8512 VSC8552 VSC8522 VSC8572 VSC8538 VSC8574 VSC8558 VSC8582 VSC8601 VSC8584 VSC8634 VSC8658 VSC8575 VSC8562 VSC8564 VSC8501
VTSS_CHIP_10G_PHY	VSC8256 VSC8487-15 VSC8257 VSC8488-15 VSC8258 VSC8489 VSC8484 VSC8490 VSC8486 VSC8491 VSC8488
VTSS_CHIP_SPARX_III_11	VSC7414
VTSS_CHIP_SERVAL_LITE	VSC7416
VTSS_CHIP_SERVAL	VSC7418
VTSS_CHIP_SERVAL_T	VSC7410
VTSS_CHIP_SERVAL_TP	VSC7415
VTSS_CHIP_SERVAL_TE	VSC7430
VTSS_CHIP_SERVAL_TEP	VSC7435
VTSS_CHIP_SERVAL_2_LITE	VSC7436
VTSS_CHIP_SERVAL_TE10	VSC7437
VTSS_CHIP_SPARX_IV_34	VSC7440
VTSS_CHIP_7513	VSC7513
VTSS_CHIP_7514	VSC7514
VTSS_CHIP_SPARX_III_10_UM	VSC7420

Target	Part Number
VTSS_CHIP_SPARX_III_17_UM	VSC7421
VTSS_CHIP_SPARX_III_25_UM	VSC7422
VTSS_CHIP_CARACAL_LITE	VSC7423
VTSS_CHIP_SPARX_III_10	VSC7424
VTSS_CHIP_SPARX_III_10	VSC7424
VTSS_CHIP_SPARX_III_18	VSC7425
VTSS_CHIP_SPARX_III_24	VSC7426
VTSS_CHIP_SPARX_III_26	VSC7427
VTSS_CHIP_CARACAL_1	VSC7428
VTSS_CHIP_CARACAL_2	VSC7429
VTSS_CHIP_SERVAL_2	VSC7438
VTSS_CHIP_SPARX_IV_52	VSC7442
VTSS_CHIP_SPARX_IV_44	VSC7444
VTSS_CHIP_SPARX_IV_80	VSC7448
VTSS_CHIP_SPARX_IV_90	VSC7449
VTSS_CHIP_LYNX_2	VSC7464
VTSS_CHIP_JAGUAR_2	VSC7468

2.1.4 Instance References

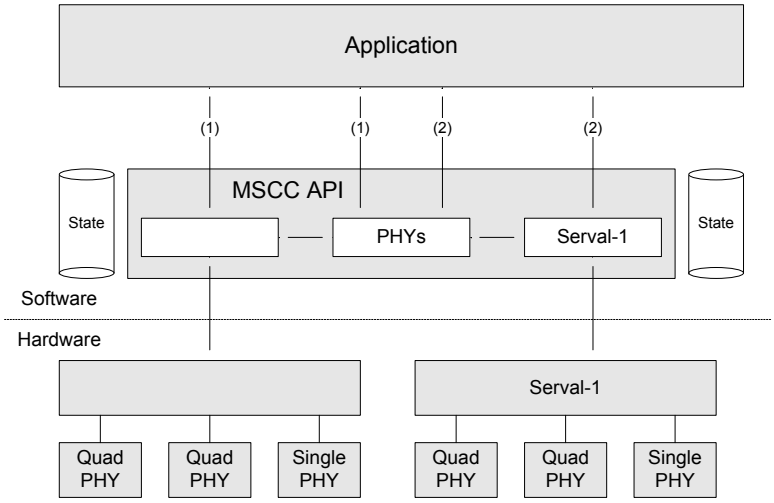
The API supports control of multiple target instances from one application. To facilitate this, the API provides a create function, `vtss_inst_create`, which must be called for each target instance during initialization. The create function returns an instance reference that must be used for subsequent API calls on the same target.

For applications controlling a single target instance, the create function may be called with a NULL instance reference pointer and subsequent API calls on the target may use a NULL instance reference.

The following illustration shows how the instance references are used to select target instances for an application controlling two Serval-1 switches, both of which control multiple PHYs. The API internally

associates a state block with each created target. In the illustration, the interface call references are indicated using numbers.

Figure 3 • Instance References



The first instance created is the left Serval-1 target. Subsequent API function calls using the returned instance reference are directed to the left Serval-1 chip for switch and PHY control.

The second instance created is the right Serval-1 target. Subsequent API function calls using the returned instance reference are directed to the right Serval-1 chip for switch and PHY control.

2.1.5 Initialization

The following initialization sequence must be used for each target instance controlled by the application. The application example includes initialization code demonstrating this sequence. Note the differences between switch targets as compared to PHY-only targets.

1. Create a target instance using `vtss_init_get` and `vtss_init_set` .
2. Initialize a target instance using `vtss_init_conf_get` and `vtss_init_conf_set`.
   - For switch targets, register read/write callback functions must be provided by the application.
   - For PHY-only targets, MII management read/write callback functions must be provided by the application.
3. Set up the port map table using `vtss_port_map_set` for switch targets.

When the sequence has been completed, the application can start controlling ports, quality of service (QoS), and other functions on the target.

2.1.6 API Protection

The API functions are non-reentrant due to an internal state in the targets and the API. To protect API accesses for multi-threaded applications, the following two callback functions must be implemented by the application using a semaphore.

- `vtss_callout_lock` is called by the API when entering an API function.
- `vtss_callout_unlock` is called by the API when exiting an API function. For single-threaded applications, these functions may be left empty.

2.1.7 OS Layer

The OS layer implements timer functions required by the API.

An OS-specific header file is needed for each OS type. The API supports Linux and eCos, but can be extended to support any OS with basic timer functionality.

The OS-specific header defines the following entities.

- `vtss_mtimer_t(typedef)` is a data type used to implement timers.
- `VTSS_NSLEEP(nsec)` is a macro to sleep for nsec nanoseconds.
- `VTSS_MSLEEP(msec)` is a macro to sleep for msec microseconds.
- `VTSS_MTIMER_START(timer, msec)` is a macro to start a timer by initializing the timeout to msec microseconds, where timer is a variable declared of the type `vtss_mtimer_t`. If the timer has already been started, it must be restarted with the new timeout value.
- `VTSS_MTIMER_TIMEOUT(timer)` is a macro used to determine if the timer has expired, where timer is a variable of declared type `vtss_mtimer_t`.

2.1.8 Trace Layer

The API code includes trace macros, in the style of the C function printf, for debugging and troubleshooting. The application must implement the following callback functions for mapping the trace macros.

- `vtss_callout_trace_printf` prints or logs a trace message.
- `vtss_callout_trace_hex_dump` prints or logs a hexadecimal data dump.

The trace macros are organized in groups with levels that may be changed at runtime. By default, only the error trace is enabled.

The trace system may be disabled by setting `VTSS_OPT_TRACE` to zero at compile-time. If the trace is disabled, the function must not be implemented.

2.1.9 Recommended API Calling Sequence

The following general sequence is recommended if using the PHY API. It will help prevent the passing of structures from the application to the PHY API with uninitialized or incorrectly initialized variables. In general, the PHY API does not perform boundary checking of elements within a given structure passed into the API.

1. Declare the structure to be passed into the PHY API.
2. Initialize all parameters in the declared structure to zero using memset or an equivalent OS function.
3. If the PHY API has a `get` function, execute the `get` function call so that the structure that was previously declared and passed in gets set to the existing values in the PHY API (for example, `vtss_init_conf_get`).
4. Modify only the values within the structure where a change is desired.
5. If the PHY API has `set` functions, execute the `set` function call so that the structure passed in gets set in the hardware configuration registers by the PHY API (for example, `vtss_init_conf_set`).

The issues caused by uninitialized elements or incorrect default settings for elements within any given data structure passed into the PHY API can be minimized by performing the previous sequence.

2.1.10 Checklist for API Configuration

The following items may need additional attention when porting the PHY API. Some initializations can be done on the compilation line.

- Initialize `VTSS_OPT_PORT_COUNT` macro (for example, -DVTSS_OPT_PORT_COUNT=4).
- Initialize OS type (for example, -DVTSS_OPSYS_LINUX=1).
- Initialize PHY chip type (for example, -DVTSS_CHIP_CU_PHY).
- Initialize the `VTSS_MSLEEP` macro properly.

2.1.11 Sample API Demo Applications

Demo applications for the PHY API can be found in the `vtss_api/appl` directory.

`vtss_appl_cu_phy.c` is a sample demo program for 1G PHY.

2.2 Product Families

This section briefly describes the API functionality for each product family.

2.2.1 1G PHY Family

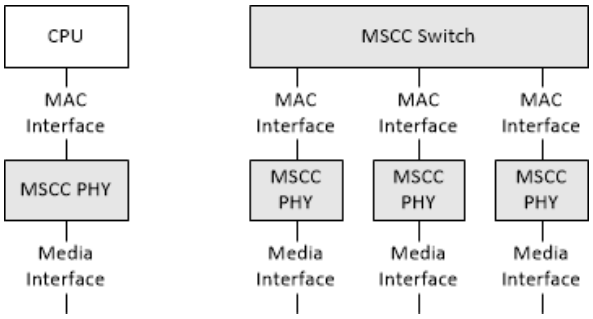
The 1G PHY family includes products listed for `VTSS_CHIP_CU_PHY` in *Table 1* on page 4 .

The 1G PHY API functions and structures are described within the chip API documentation included in the API package `/doc/vtss_serval.pdf`.

2.2.1.1 PHY Applications

The API supports a number of Microsemi PHY products. Each chip may include multiple PHYs (ports). For each target instance, the individual PHYs are identified using a port number. The following illustration shows two PHY applications.

Figure 4 • PHY Applications



One single PHY connects to a MAC inside a CPU—both the MAC interface and the MII management interface of the PHY connect to the CPU. Three octal PHYs connect to a VTSS switch chip—both the MAC interfaces and the MII management interfaces of the PHYs connect to the switch chip.

The API contains examples of both application types described in *Application Examples* on page 13.

2.2.1.2 PHY Initialization

The following steps perform PHY initialization.

1. Initialize the chip by calling the `vtss_phy_pre_reset` function. This will perform initialization needed for the entire chip (for example, load the internal 8051 CPU). The `vtss_phy_pre_reset` function must be called with the lowest PHY (port) number within the chip.
2. Initialize each PHY within the chip by calling the `vtss_phy_reset` function for each PHY (port).
3. Startup the chip by calling the `vtss_phy_post_reset` function. This performs chip setup, which is needed after the first time all PHYs within the chip have been reset (for example, set the coma pin). The `vtss_phy_post_reset` function can be called with any PHY (port) number within the chip.

2.2.1.3 PHY Control After Initialization

After the main initialization, the normal PHY control sequence for each port is as follows:

1. Configure the PHY using `vtss_phy_conf_set`.
2. Periodically poll the PHY status to detect link change events:
   - For PHY-only applications, use `vtss_phy_status_get`.
   - For switch applications, use `vtss_port_status_get`.
3. If link state events are detected, the application must take appropriate action. If auto-negotiation is enabled, the switch must be configured normally on link-up events, such as speed, duplex, and flow control.

2.2.1.4 PHY Debug Print Functions

The following two functions assist the debug process.

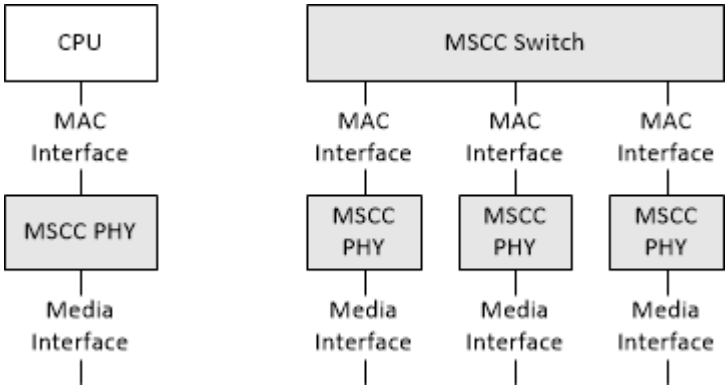
- `vtss_phy_debug_stat_print` prints the PHY statistics.
- `vtss_phy_debug_phyinfo_print` prints the internal PHY information.

2.2.2 10G PHY Family

The 10G PHY family includes products listed for `VTSS_CHIP_10G_PHY` in *Table 1* on page 4.

The individual PHYs are identified using a port number. The following illustration shows two PHY applications.

Figure 5 • 10G PHY Applications



One single 10G PHY connects to a XAUI interface of a MAC inside a CPU—both the MAC interface and the MDIO management interface of the PHY connect to the CPU. Multiple 10G PHYs connect to a VTSS switch chip—both the MAC interfaces and the MDIO management interfaces of the PHYs connect to the switch chip.

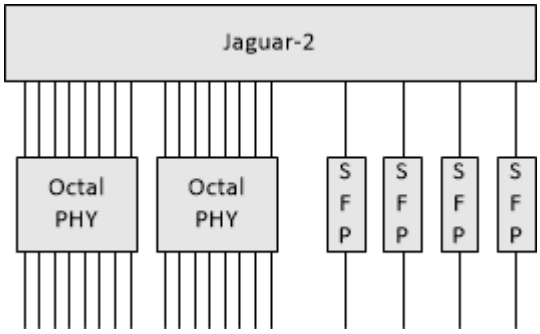
The API contains examples of both application types as described in *Application Examples* on page 13. The normal 10G PHY control sequence for each port is to set the PHY operating mode using

`vtss_phy_10g_phy_mode`. This function detects, resets, and sets the operating mode of the PHY type.

2.2.3 Switch Families

The following illustration shows an example of a switch based on the Jaguar-2 product. The system is a Layer 2 switch with 16 copper ports and four SFP ports.

Figure 6 • VSC7468 Jaguar-2 Application



The following steps are the initialization sequence for switch families.

1. Create a target using `vtss_inst_get` and `vtss_inst_create`.
2. Initialize a target instance using `vtss_init_conf_get` and `vtss_init_conf_set`.
3. Set up the port map table using `vtss_port_map_set`.
4. Configure the switch ports using `vtss_port_conf_get` and `vtss_port_conf_set`. If PHYs are present, reset and configure PHYs.

2.3 Function Groups

The following sections list the most important API functions. The header files include detailed descriptions on the data types and functions while the documentation directory includes a document for each product family. These resources can be used to study the API for a specific product.

The following general definitions are listed in `vtss_api/include/vtss/api/options.h`.

- Feature defines. The selection of one or more targets at compile time causes a number of symbols (VTSS_FEATURE_*) to be defined. These are used to indicate which functions and data fields are available.
- Options. Default values are assigned to compile time options (VTSS_OPT_*). These default values may be overridden when building the API.

2.3.1 Initialization

The following functions are defined in `include/vtss_init_api.h`.

- Target creation
- Target initialization
- Restart functions for targets supporting warm start

Target creation and initialization must be called before other target functions.

2.3.2 Miscellaneous

The following miscellaneous functions are defined in `include/vtss_misc_api.h`.

- Trace configuration and callback functions
- Debug print functions (for showing API internal information)

- API lock/unlock callback functions
- Register read/write (for debugging switch targets)
- Chip ID access (for switch targets)
- GPIO control (for switch targets)
- Interrupt control (for switch targets)

2.3.3 Port Control

The following port control functions for switch targets are defined in `include/vtss_port_api.h`.

- Port map (must be called after initializing switch targets)
- 10G PHY counters
- MMD management for 10G PHYs
- Auto-negotiation, IEEE 802.3 clause 37
- Port configuration (speed, duplex, flow control, and so on)
- Port status
- Port statistics

The following PHY control functions are defined in `include/vtss_phy_api.h`.

- PHY reset
- PHY configuration and status
- PHY power configuration and status
- Recovered clock configuration
- PHY register read/write functions
- VeriPHY (cable diagnostics)

The following 10G PHY control functions are defined in `include/vtss_phy_10g_api.h`.

- 10G PHY operating mode and status
- 10G PHY sublayer status
- 10G PHY reset
- 10G PHY power on/off
- 10G PHY loopback

2.3.4 Quality of Service

The following QoS control functions for switch targets are defined in `include/vtss_qos_api.h`.

- QCL: QoS classification rules
- Bandwidth control (policing and shaping)
- Egress scheduler

2.3.5 Packet Control

The following packet control functions for switch targets are defined in `include/vtss_packet_api.h`.

- CPU Rx packet registration and CPU queue mappings
- CPU Rx functions
- CPU Tx functions

The following frame DMA functions, which can only be used from the internal CPU of the switch, are defined in `include/vtss_fdma_api.h`.

- FDMA initialization and channel configuration

- CPU Rx functions (frame extraction)
- CPU Tx functions (frame injection)
- FDMA statistics

2.3.6 Security

The following security control functions for switch targets are defined in `include/vtss_security_api.h`.

- 802.1X state
- Access control list

2.3.7 Layer 2

The following Layer 2 functions for switch targets are defined in `vtss_l2_api.h`.

- MAC address table
- Learning mode
- RSTP and MSTP state
- Virtual LAN (VLAN) membership and port configuration
- VCL: advanced VLAN classification
- VLAN translation
- Port isolation
- Private VLANs
- Link aggregation
- Port mirroring
- IPv4 multicast control
- IPv6 multicast control
- Port protection switching
- Ring protection switching
- Port forwarding state
- VStaX stacking configuration

2.3.8 Ethernet Virtual Connection

EVC functions are defined in `vtss_evc_api.h`.

2.3.9 Synchronization

Synchronization functions are defined in `vtss_sync_api.h`.

2.3.10 Time Stamping

Time stamping functions at the MAC layer for the switch families are defined in `vtss_ts_api.h`.

The time stamping control functions at the PHY layer for devices that support IEEE 1588v2 (such as the VSC8487-15, VSC8488-15, VSC8574, and VSC8492) are defined in `vtss_phy_ts_api.h`.

- One-step and two-step time stamping feature
- Tx time stamp FIFO interface
- Set/Get/Synchronize time of day
- Adjust clock rate

2.4 Application Examples

Application examples that demonstrate API use are included in the `vtss_api/appl` directory. The applications can be built to run the Microsemi platforms shown in the following table.

The application can be built with Linux CMake or by compiling the following files with the listed definitions.

If using the API running Linux on the internal CPU of a switch chipset, also see *Porting Guide* on page 14.

Table 3 • Application Example Platforms

Platform	Defines	Application C Files
Switch application for Serval-2 reference board controlled through PCIe from a Linux PC.	VTSS_CHIP_SERVAL_2 VTSS_OPSYS_LINUX VTSS_OPT_VCORE_III=0 BOARD_SERVAL2_REF	appl/vtss_appl.c appl/vtss_appl_trace.c appl/vtss_appl_cli.c appl/vtss_appl_switch_cli.c appl/vtss_version.c appl/vtss_appl_board_generic.c appl/vtss_appl_board_generic_uio.c boards/board_probe.c boards/port_custom_jr2.c

2.4.1 Switch Application Functionality

The following basic switch application functions are defined in `appl/vtss_appl.c`.

- Trace system integration
- Initialization
- Port map setup
- Port reset and configuration
- Port status polling and configuration based on auto-negotiation

A command line interface (CLI) for system configuration and monitoring is included in `appl/vtss_appl_cli.c`.

2.4.2 PHY Application Functionality

PHYs on the target board are initialized and their use is demonstrated in `appl/vtss_appl_cu_phy.c`.

The following steps set up the ATOM12 PHY device from a PC (either Linux or Windows using CyWin) for use on the Microsemi Atom12 evaluation board.

1. Compile the API with the ATOM12 evaluation hardware platform and Linux OS files.

   Example:

```
gcc -g -o vtss_api.exe  -I include \
appl/vtss_appl_board_atom12_eval.c \
appl/vtss_appl_cu_phy.c \
`find base/phy/ -name "*.c"`
-DVTSS_OPSYS_LINUX=1 \
-DVTSS_OPT_PORT_COUNT=12 \
-DVTSS_CHIP_CU_PHY
```

```
vtss_api.exe 10.10.132.59.
```

2. Specify the Rabbit board's IP address, which is set as an argument when running the API program, because the Atom12 evaluation board uses a Microsemi add-on Rabbit CPU board that communicates with the PC, using a socket connection, and PHY.

   Example:

   ```
   vtss_api.exe 10.10.132.59.
   ```

2.5 Linux Support

The following sections discuss the Linux support.

2.5.1 External CPU Configuration

Example UIO drivers that provide register and interrupt access to run the API are included in `linux_support/misc`. This directory also includes the example application described in *Application Examples* on page 13.

The drivers can run the API on a Linux system with PCIe connections (for example, from a x86 Ubuntu 14.04 or later LTS system).

Other systems configurations can be constructed similarly—replacing PCIe with other bus options available on the target platforms.

2.5.2 Internal CPU Configuration

If you are using the internal processor to run Linux, you may wish to use the Yocto-based BSP that Microsemi provides. Refer to Application note AN1125, available from your local Microsemi representative, for further description of the BSP.

Once you have the basic BSP compiling, you should add the `meta-vtss-switch` layer, available at `https://github.com/vtss/meta-vtss-switch`. You will also need to obtain a suitable API release tarball, which should be placed in recipes-applications/vtss-api/files/. You will need release 4.60a or later.

After downloading the `meta-vtss-switch` layer and placing the tar archive, you must add the layer to the `<build-directory>/conf/bblayers.conf` as in the following example.

```
BBLAYERS ?= " \
  /home/<user>/project/poky/meta \
  /home/<user>/project/poky/meta-yocto \
  /home/<user>/project/poky/meta-yocto-bsp \
  /home/<user>/project/poky/meta-vtss-vcoreiii \
  /home/<user>/project/poky/meta-vtss-switch \
  "
```

With the new layer added, you can now build (bitbake) the `vtss-core-minimal` target, and produce a flash image containing the Microsemi Unified Switch API shared library as well as the `vtss_miniapp` example application.

Refer to AN1125 and https://www.yoctoproject.org/ for more information for more information on Yocto and how to deploy the image on a target system.

2.6 Porting Guide

This section summarizes the required steps to port the API to a given platform.

2.6.1 Board Support Package

For a given CPU system and OS, a BSP must be developed. Microsemi currently provides the following BSPs.

- VCore-III/eCos for SparX-III/Caracal/Jaguar reference systems
- VCore-III/Linux for SparX-III/Caracal/Jaguar reference systems

2.6.2 Build System

Create a build system to compile the OS, application, and Microsemi API. All C-files in the `vtss_api/base` directory, including subdirectories, must be compiled. Also set the appropriate OS (`VTSS_OPSYS_ECOS`, `VTSS_OPSYS_LINUX` or `VTSS_OS_CUSTOM`) and target (`VTSS_CHIP_*`) definitions for the platform.

The file `vtss_api/CMakeLists.txt` is available to support CMake.

2.6.3 OS Layer

When compiling unsupported OS types, the makefile must define the `VTSS_OS_CUSTOM` C preprocessor symbol. By doing so, `vtss_os_custom.h` will be included to implement the OS timer functions, as described in *OS Layer* on page 6. This file is reserved for OS porting and not present in the distributed source.

2.6.4 Register Access

Register read/write functions must be implemented for switch targets. MII management read/write functions must be implemented for PHY targets.

2.6.5 API Protection

The `vtss_callout_lock` and `vtss_callout_unlock` functions must be implemented. For single-threaded applications, these functions may be left empty.

2.6.6 Trace Layer

If the application wants to use the API trace, the `vtss_callout_trace_printf` and `vtss_callout_trace_hex_dump` functions must be implemented. Alternatively, set `VTSS_OPT_TRACE=0` in the build system to exclude the trace.

The API is split into different groupings—the `vtss_trace_conf_set` function can enable the trace individually for each group.

2.6.7 Application

The application code must be developed, possibly using the Microsemi application example code for reference. Include the file `vtss_api/include/vtss_api.h` to access the Microsemi API. For switch targets, ensure correct port map setup.

Microsemi Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA

Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996
Email: sales.support@microsemi.com
www.microsemi.com

Microsemi, a wholly owned subsidiary of Microchip Technology Inc. (Nasdaq: MCHP), offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions; security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, California, and has approximately 4,800 employees globally. Learn more at **www.microsemi.com**.

VPPD-04303