ENT-AN1184-4.3 Application Note

Brief Introduction to WebStaX SW Architecture

Released

March 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 1.1

Revision 1.1 was published in March 2018. In revision 1.1, the document was updated to condense the information to an overview of the overall architecture. Detailed information is now located in UG 1068 SW Introduction to WebStaX under Linux, which can be found in the doc directory of the software installation.

Revision 1.0

Revision 1.0 was published in August 2017. It was the first publication of this document.

2 Introduction

This document offers a brief introduction to WebStaX, its design, and how to customize and integrate it. More detailed information can be found in the product specification and throughout the various user and configuration guides.

3 Component Overview

WebStaX is a collection of many different SW components, which are combined into a product that targets the end-user. This collection of SW includes both third-party and internal developed SW, and it also includes both proprietary and open-source SW.

This means that the WebStaX product includes a fair amount of SW developed for this specific purpose, but it also includes third-party SW that has been carefully selected, integrated, and in some cases patched and tested as a single product.

When new versions of WebStaX (updates) are being released, it typically updates all the components used in WebStaX, while offering a mechanism to install the upgrades. Each release of WebStaX includes a list of all third-party SW included, and specifies the versions and license terms for each third-party SW packet.

The following subsections list all the main components that constitute the WebStaX product.

3.1 Board Support Packet (BSP)

The BSP provides the following third-party components.

- Development tools needed to build the executable application file (such as cross-compiler, cmake, linker, and automake/autoconf).
- Third-party components needed on target (such as Linux kernel, libc, net-snmp, dropbear, and busybox).

Microsemi provides a BSP that is designed and optimized for Microsemi reference boards along with the WebStaX application software. The BSP is distributed both as source and binary. The sources are needed for customers who need to change the BSP, while the binary BSP can be used if no changes are required. More details can be found in UG1068, which is shipped with SDK (CEServices\docs).

3.2 MESA/Unified-API

The MESA/Unified-API is a library used to access the switch or PHY hardware. The API is included as part of the application software. Customers, that are building a product based on WebStaX variants, automatically use the API included in the WebStaX source package.

The API provides an abstraction layer so that the application does not need to be aware of the register set provided by a single chip, allowing the application to support many different switch cores. More details can be found in UG1070, which is shipped with SDK (CEServices\docs).

3.3 Application

The WebStaX product family includes four different application packages: WebStaX, SMBStaX, IStaX, and CEServices. The four packages have different feature sets and licensing terms. This document does not focus on the individual packages, but it assumes that one of the four packages is being used. When referring to MSCC-Application, application or switch application, then it is one of these four packages.

The application is a collection of SW modules that can be divided into the following categories listed in the table:

Table 1 • Collection of Software Modules in WebStaX Product Family

Modules	Description
Control modules	Allows the user to apply static configuration and/or query status. Such modules typically use the MESA library to configure the switch-HW accordingly. Examples of such modules are: port, acl, vlan, and evc.

Modules	Description
Protocol modules	Allows the user to enable/configure protocols that can interact with other network equipment. Such modules typically listen for certain network packets and transmit packets according to the specification of the protocols. Example of such modules are: gvrp, lldp, and ptp.
Interface modules	Provides management interface that can be used to configure and/or query status of switch. Example of such modules are: cli, ssh, snmp, json-rpc, and web.
Infrastructure modules	This is a set of modules that are not directly visible to the user, but provide some infrastructure facilities used by SW running on the switch. Examples of such modules are: basics (algorithms and containers), critd (mutex and dead lock detection), and trace (debug facilities).

The application SW is organized in such way that each module can be enabled/disabled in the build process, and new modules can easily be added.

Note:

Some of the application modules have internal dependencies that must be considered when enabling/disabling modules. Not all combinations have been tested, and certain combinations may lead to compilation errors.

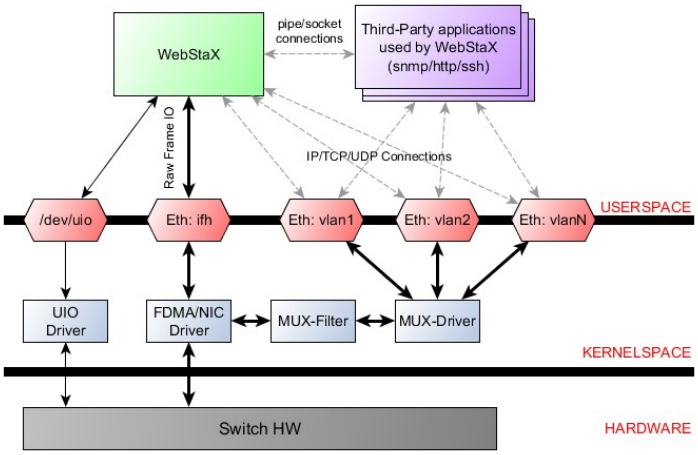
3.4 Flash Images

A flash image is a binary image that may be burned to the NOR flash using a programmer. The flash images include a partition table for the NOR flash, bootloader, and bring-up image (Linux kernel, stage1 file system, and stage2 minimal). A flash image may only be used on the specific board for which it is designed.

4 Integration with Linux

This section provides a brief overview of the system architecture and goes into some details on how WebStaX is integrated with the Linux OS. The following illustration shows the overall system architecture.

Figure 1 • Overall System Architecture



The green box-labeled WebStaX is the MSCC switch application, it can be any of the supported variants (WebStaX, SMBStaX, IStaX, or CEServices). The switch application is running as a long-lived, normal user-space process (as root), and it is interacting with the switch registers through the uio driver. The WebStaX application includes an instance of the API (linked in as a library). The application must be the exclusive owner of the API and switch registers.

> **Note:**
>
> This means that no other process is allowed to instantiate the API or alter the switch registers in HW. Each process must go through the API instance already created by the application. Other processes can communicate with the WebStaX application and indirectly access the API through the WebStaX process).

The `uio` kernel space driver is a simple kernel module that does the following two things.

- Exposes the entire register region of the switch hardware.
- Exposes all interrupts from the switch HW.

The `uio` kernel module is provided by the Linux kernel (part of the BSP) and allows user-space applications, like WebStaX, to gain access to HW registers and interrupt from user-space. This is achieved by a `mmap` of the register region from the user-space application.

4.1 Frame Flow

Besides configuring the switch registers in HW, the application also implements a number of protocols (which may influence the switch configurations). To implement these protocols, the application needs to inject frames into the switch core, and it needs to extract frames that have been redirected to the CPU (either because it was sent to the MAC address of the CPU, or because an ACL rule has captured the frame). To implement this frame-flow, the Linux kernel in the BSP provides a FDMA driver that can inject/extract to/from the CPU queue in the switching hardware.

Frames that are injected/extracted to/from the CPU queue are prefixed with an extra header that carries various side-band information related to the frame (front port, classified VLAN, ACL rule number, time stamp, and so forth). The content of the header is chip dependent, and the content is specified in the data

sheet of the switching chip. This information is needed by the application to implement most of the L2 protocols, but it also causes a problem when the frame is being processed through the Linux IP stack (as the kernel does not know inter-frame-header). To solve this, received frames are being exposed both on a Linux network interface called `ifh` (short for interface frame header) and to the `MUX-Filter` (see *Figure 1* on page 5).

The `MUX-Filter` sees all the frames being received by the CPU queue in the switching hardware. The driver decodes the frame header to see which classified VLAN a given frame belongs to, and if such an interface exists, then the switch dependent frame header is popped and the frame is being processed by the Linux IP stack. The `MUX-Filter` is configured by the user-space application using the `netlink` protocol, and this configuration channel allows the application to dynamically create and delete IP interfaces that correspond to a VLAN domain. These types of interfaces are being referred to as VLAN interfaces.

This design allows the user-space applications to implement various L2 protocols and have access to all the side-band data collected by the switch-core. It also allows existing Linux applications to do various socket operations (IP, UDP, and TCP) without changing these applications.

4.2 System Services

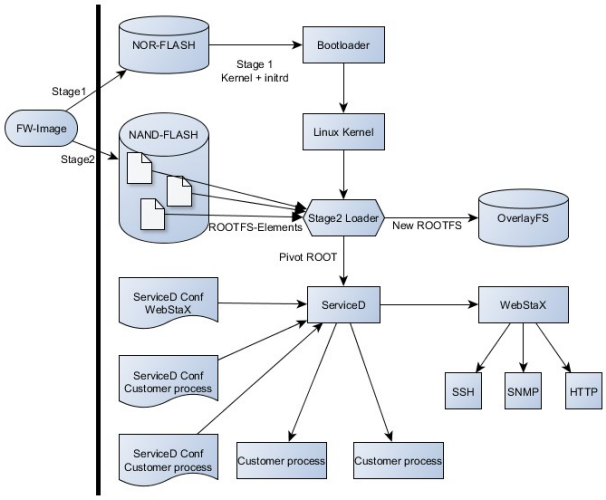
The WebStaX application listens on a number of TCP/UDP ports, and it creates a number of third-party services. The list of TCP/UDP ports and third-party services depends on the variant (WebStaX, SMBStaX, IStaX, or CEServices). One of the examples of listening ports is TCP port 23 that the application listens on in order to implement telnet. One of the examples of third-party services is `hiawatha`, which is being used as web-server and `net-snmp` as SNMP master agent.

The external services needed by the WebStaX application are automatically started by the application itself. The application also offers configuration hooks that can stop a given service, if the user does not wish to use it.

4.3 Boot Sequence

The boot sequence of a WebStaX system differs from general purpose Linux systems because it starts by booting from NOR, and when the kernel is up, it mounts the NAND flash as its root file system. The system also uses a custom `init` process called `ServiceD`. The following illustration shows the boot process of a WebStaX system.

Figure 2 • Boot Process



4.3.1 The Modular Image Format

The image format used in WebStaX is called modular firmware images (mfi), and it is designed such that more file-system images can be appended. When the system is booted, the union of all the appended filesystems is presented as the root-file system.

When booting, the `init` process iterates through each section in the mfi files and mounts each root file system element on top of each other by using the `OverlayFS` facilities in the Linux kernel. Once this process is completed, the final root-file system is ready, and the boot process continues from the freshly prepared root file system.

Note:

The mfi format allows the different root file system elements to be placed in either NOR or NAND flash.

4.3.2 ServiceD as init Process

When the final root file system is ready, the system initializes all the services that need to be running. The ServiceD application is used to perform this task. The ServiceD process reads its configuration files (see ServiceD Conf WebStaX, and ServiceD Conf Customer process in the *Figure 1* on page 6), creates, and monitors the configured services. In a vanilla WebStaX system, there is only one service called `switch_app`, which represents the WebStaX application. When the application is started, it automatically starts the set of services it depends on.

Note:

ServiceD is not the same as systemd. ServiceD is the init process developed by Microsemi. See ENT-AN1163 for more details.

5 Software Customization Options

WebStax offers a wide range of facilities to allow various customizations. The following is a list of regularly used customization facilities.

5.1 WebStaX Application Customization

The WebStaX application consists of a number of modules. The building system allows to disable certain modules, if needed (inter module dependency exists and needs to be considered when doing so).

Customers can also add modules of their own and integrate them as part of the existing product. This allows to extend the existing facilities and still provide the end-user with a unified stream line set of interfaces.

5.2 Interface/Management Customization

All configuration, status, and control in the WebStaX application is exposed as a JSON-RPC interface. This JSON-RPC interface can either be accessed through a HTTP(S) connection (allowing remote access/control), or by using an IPC pipe (to control WebStaX from a local process).

This JSON-RPC interface allows to create other interfaces and/or network management systems.

5.3 GUI Adjustments/Replacement

The WebStaX application includes a web, front-end written in html/java-script. This interface can be adjusted with other colors, logos, and so forth. Also, the entire web interface can be replaced with an alternative GUI that uses the JSON-RPC interface.

5.4 Third Party Daemons

Third-party daemons can be added to the `mfi` images and can be started by the ServiceD-init process. Such daemons can either be open-source or proprietary. Third-party daemons can use the existing L3 interfaces to communicate with the outside world, and/or us the JSON-IPC facilities to access the switch facilities through the WebStaX application.

6 HW Integration Options

The MSCC switch chips include an internal CPU that can be used to run the switch application. But, it is also possible to do a board design that uses an external CPU instead.

Customers have to choose whether they want to use the internal CPU or if they prefer an external CPU. Arguments for choosing an external CPU exist, if more CPU resources are needed, or that an alternative CPU architecture is required.

Customers that choose to do a project with an external CPU, must also provide the BSP for the given project. The MSCC source BSP can be adjusted to support most CPU architectures, or a custom made BSP can be designed from scratch.

The preferred way of reaching the registers from an external CPU is by using PCI-e. The alternative option is SPI.

6.1 Frame Flow with an External CPU

Projects using an external CPU need to decide how to implement the frame-flow between the switch-core and the host CPU. There are two options: either use PCI- Express or dedicate one of the switch ports for the purpose.

7 Alternatives to WebStaX

Customers that does not want to use WebStaX for specific reasons are welcome to use an alternative SW-Stack. Such projects should integrate the Unified-API/MESA with the alternative SW-Stack and may optionally be using the BSP.

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-04465