FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

VxWorks BSP for Pandaboard

Development of a VxWorks based Board Support Package for the Pandaboard

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Computer Engineering

by

Florin Hillebrand
Registration Number 0925917

to the Faculty of Informatics

at the TU Wien

Advisor: Univ. Prof. Dr. Ulrich Schmid

Vienna, 19th March, 2018

Florin Hillebrand   Ulrich Schmid

Erklärung zur Verfassung der Arbeit

Florin Hillebrand
Lacknergasse 70
1170 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 19. März 2018

Florin Hillebrand

Danksagung

Zu aller erst richtet sich mein Dank an Prof. Schmid, welcher mir die Gelegenheit gegeben hat, mit einer so populären und wichtigen Software zu arbeiten. Besonders gilt der Dank für sein Vertrauen, welches er mir vom ersten Gespräch an geschenkt hat.

Was aus dieser Arbeit nicht gleich ersichtlich ist, ist das gesamte Hintergrundwissen, das ich in meiner Studienlaufzeit an der Universität aneignen konnte. Zahlreiche Professoren und Assistenten verdienen meine Anerkennung für deren pädagogischen und professionellen Einsatz, die ich leider hier nicht alle aufzählen kann. Zumindest bedanke ich mich aber bei Herrn Robert Najvirt für seine Lehrveranstaltung „Hardware Modelling“ und die Laborübungen im Rahmen des Kurses „Digital Design and Computer Architecture“, welche mir grundlegendes Verständnis eines Prozessors vermittelt haben.

Meine Familie spielte im gesamten Studium, wie auch bei dieser Arbeit, eine sehr wichtige Rolle und verdienen sich ein außerordentliches Dankeschön meinerseits. Mein Bruder Max hat mir sein Zimmer zur Verfügung gestellt, damit ich mit der Entwicklung dieses *Board Support Package (BSP)* beginnen konnte. Mein Vater Bernhard hat mich maßgeblich in meiner gesamten Laufbahn vielseitig beeinflusst und stellte mir alle Materialien für diese Arbeit zur Verfügung die ich benötigte. Ein ganz besonderer Dank geht an meine Mutter Christine, welche mich seit Beginn moralisch sowie auch finanziell unterstützt hat.

Zu guter Letzt richtet sich mein Dank auch an meine Freunde und Kollegen Juri Berlanda und Rene Kofler, für deren erfahrungsvollen und professionellen Meinungen und Beiträge.

Acknowledgements

First of all, I would really like to express my appreciations to Prof. Schmid, who gave me the opportunity to work on such a popular and important software, especially for his faith in me from our first talk on.

What is not revealed by this document, is the whole background knowledge that I could gain in the lectures at the university. There are numerous professors and assistants to who I would like to express my gratitude for their educational and professional input. I cannot name all of them here. But at least I would like to acknowledge Robert Najvirt for the lecture Hardware Modelling and the lab exercises in Digital Design and Computer Architecture, which gave me a fundamental understanding of how a processor works.

My family played a very important role in the whole time period of my studies, including this thesis. By brother Max lent me his room at home to start with the development of this *BSP* at home. My father Bernhard inspired me in so many ways and gave me all the material that I needed for this project (plastic case, cables, ecc.). My mother Christine encouraged me from the very first time and gave me so much moral and financial support.

Last but not least, I appreciate my friends and colleagues Juri Berlanda and Rene Kofler for their experienced and professional opinions and input.

Kurzfassung

Eingebettete Systeme haben in den letzten Jahrzehnt sehr an Bedeutung gewonnen: Von Waschmaschinen und Smart-Phones bis hin zu Straßenfahrzeugen, Flugzeugen und Raumfahrzeugen. Diese sich ständig entwickelnden Anwendungen haben zur Folge, dass ständig neue Computerprogramme auf immer neueren Geräten angepasst werden müssen. Diese schriftliche Arbeit zeigt die Schritte auf, welche notwendig sind, um das weit verbreitete Betriebssystem *VxWorks* auf dem populären *PandaBoard* einsetzen zu können. Das Ergebnis dieses Projekts ist eine Software-Schicht (*Board Support Package (BSP)*), welche das Betriebssystem mit der darunterliegenden Hardware verbindet und die gemeinsame Nutzung ermöglicht. Es schafft unter anderem die Grundlage dafür, dass Studenten im Rahmen der Lehrveranstaltung *Building Reliable Distributed Systems (BRDS - 182.704)* Echtzeitanwendung auf Basis von *VxWorks* mittels konventioneller Hardware entwickeln können. Auf der anderen Seite, dient dieses Dokument all denen, die sich für die Entwicklung eines *VxWorks BSP* für eine spezielle Hardware interessieren.

Abstract

The importance of embedded systems has grown rapidly in the last decade: from dish washers and smart phones, up to cars, planes and spacecraft. As a consequence of these fast evolving applications, software needs to be continuously adapted to new hardware. This paper explains the steps needed to adapt the wide spread operating system *VxWorks* to the popular *PandaBoard* hardware. The result of this project is a software layer (*Board Support Package (BSP)*) that connects the operating system with the underlying hardware. Among other applications, it also creates the basis for students, creating real-time applications based on *VxWorks* using conventional hardware in the course *Building Reliable Distributed Systems (BRDS - 182.704)*. On the other hand, this document is valuable for everyone interested in creating a *VxWorks BSP* for a specific hardware.

Contents

CHAPTER 1

Introduction

This paper documents the development of a *BSP* that connects the *PandaBoard* A4 hardware with the *Real Time Operating System (RTOS) VxWorks* 6.9 (See Figure 1.1), a combination that does not exist yet. *BSP* development is a rare activity, but gains more importance with the growing need for always evolving embedded hardware, such as adapting Android to a new smart phone or television set-top box. It is a great opportunity to work with a proprietary, industry grade *Operating System (OS)*, whose applications amongst others are: automotive, aerospace, robots and astronautics. *VxWorks* is made available by *Wind River Systems (Wind River)*, which runs a university program, allowing for every university to use its software for academic use.

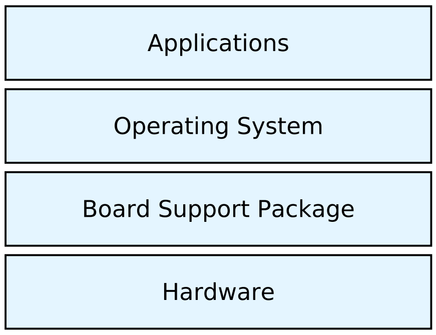

FIGURE 1.1 – Abstract layer view of a *OS*.

The scope of the practical part of this project is to allow real time applications based on *VxWorks* to run on the *PandaBoard*. This enables students from the course *Building Reliable Distributed Systems (BRDS - 182.704)*, to evaluate this OS as an *RTOS*, develop application software and compare it to others. This thesis, on the other hand, is written to give a reader an introduction into *BSP* development for *VxWorks* and to document the experiences gained during this project.

1.1 Structure of this thesis

The following section provides a high level overview about the document structure, including a short description for each chapter and how they connect together.

To gain full traceability of the project goals, requirements have been specified in the “Requirements“ Chapter 2. After this chapter, it should be clear what basic functionality should be supported by the *BSP* layer. The following “BSP architecture“ Chapter 3 introduces the reader to low-layer basics of the *OS* and the architecture of a *BSP* including its device driver management. With the requirements and a basic understanding of a *VxWorks BSP*, tasks have been identified and listed in the “Road-map“ Chapter 4. These tasks unveil their dependencies among each other and form a schedule. During the identification and the specification of the project tasks, methods have been defined to test the systems functionality; they are listed in the “Testing“ Chapter 5. During the project, numerous software and hardware tools have been used to aid in development and debugging. To gain full repeatability, all these tools have been described and collected in the “Development Environment“ Chapter 6. Chapter 7 “Pitfalls“ is a summary of the whole process and describes the problems occurred while developing. It will help the reader to reproduce steps and will help other students to extend the *BSP* with new functionality. The paper ends with the “Conclusion“ Chapter 8 that summarizes the outcome of the project. Finally, a glossary is provided to explain acronyms and domain specific terms and a bibliography which might be a good starting point for those who are working on a similar project.

CHAPTER 2

Requirements

The *PandaBoard* hardware is already used in the *Building Reliable Distributed Systems (BRDS - 182.704)* course with other *RTOSs*, like *QNX*. This is why the main requirement is to enable students to develop *VxWorks* real time applications running on the *PandaBoard*, with the optional requirement of wireless networking support.

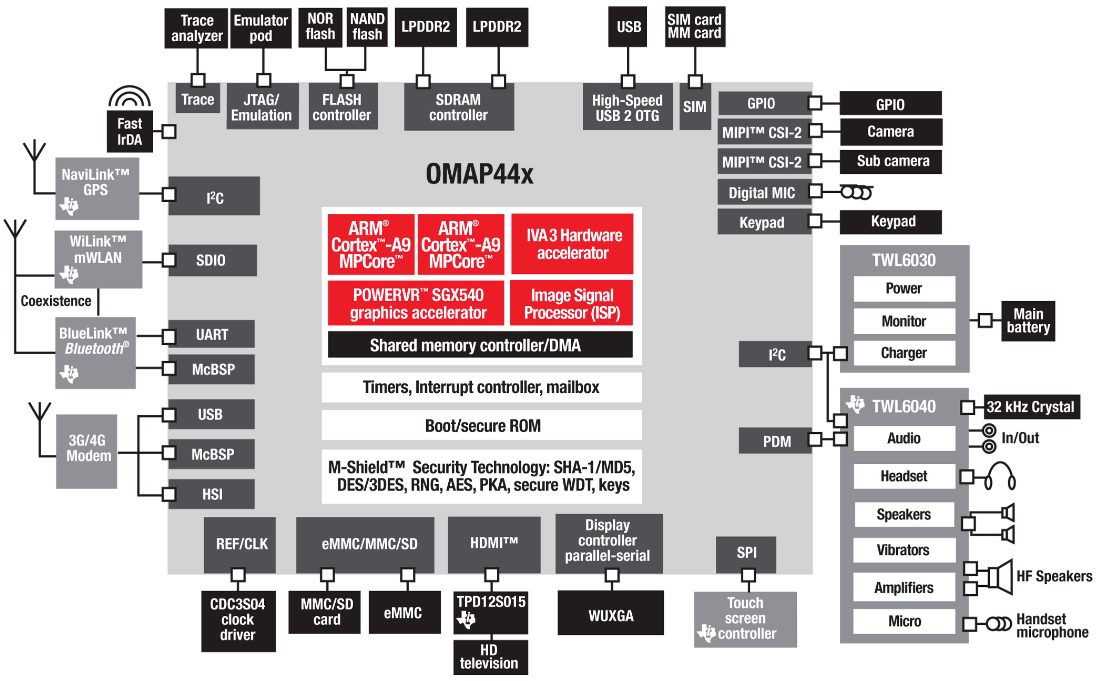

FIGURE 2.1 – OMAP4 Platform Architectural Block Diagram [18]

RQ01: Kernel

It shall be possible to start the VxWorks 6.9 kernel on one of the cores included in the on-board TI OMAP4430 processor and to use at least 1GB of the LPDDR2 memory. Further it shall support the basic functionalities of a modern multitasking processor: *Memory Management Unit (MMU)*, L2 cache, interrupts, timers and tasks.

RQ02: Timer

It shall be possible to use the VxWorks system timer for task scheduling purposes. It shall be possible to use at least one additional timer for general purpose, with possibilities to change frequency, set the timer value, get the timer value and execute user routines on events.

RQ03: Ethernet

It shall be possible to use the on-board SMSC LAN9514 USB hub and ethernet controller to communicate to the PandaBoard with the *OS* features provided by VxWorks: IP TCP/UDP stack and sockets.

RQ04: Wireless (optional)

It shall be possible to use the wireless LAN functionalities of the on-board TI TiWi transceiver module to connect to a Wireless Access Point and to communicate with others on the same network.

CHAPTER 3

BSP architecture

This chapter introduces into the basics of a VxWorks *BSP*. It explains what it is and what tasks a *BSP* covers in the big picture of this *OS*.

3.1 VxWorks BSP

According to *Wind River*, “The purpose of a *BSP* is to configure the VxWorks kernel for the specific hardware on the target board. In addition, the BSP provides an easy way to maintain portability across many different hardware configurations without having to customize the core OS, in our case VxWorks. This portability is achieved by defining a boot procedure and a set of routines that are called during the boot process for configuration, and during normal operation for specific kinds of hardware access.“ [22, Section 1.2]

The actual *BSP* is a set of assembler, C source and header files and documentation files. Every *BSP* requires a minimal set of routines and initialized variables defined in its source files. The developer can hook into different stages in the boot process and initialization phase to do proper hardware initializations. It is then the *BSP* developers task to pass the information to specific VxWorks routines used to set-up the VxWorks kernel, so that peripherals and specific features can be used from within VxWorks.

3.2 Basic configuration

All the processor internal devices, such as *Universal Asynchronous Receiver Transmitter (UART)*, interrupt controller, *Direct Memory Access (DMA)*, DDR memory controller etc. provide registers to control and monitor their functions. Each of these registers is connected through different *interconnects* mapped to the physical address space of

Module Name	Start Address (hex)	End Address (hex)	Size	Description
GPIO5	0x4805 B000	0x4805 BFFF	4KB	Module
	0x4805 C000	0x4805 CFFF	4KB	L4 interconnect
GPIO6	0x4805 D000	0x4805 DFFF	4KB	Module
	0x4805 E000	0x4805 EFFF	4KB	L4 interconnect
Reserved	0x4805 F000	0x4805 FFFF	4KB	Reserved
I2C3	0x4806 0000	0x4806 0FFF	4KB	Module
	0x4806 1000	0x4806 1FFF	4KB	L4 interconnect
Reserved	0x4805 F000	0x4805 FFFF	32KB	Reserved
UART1	0x4806 A000	0x4806 AFFF	4KB	Module
	0x4806 B000	0x4806 BFFF	4KB	L4 interconnect

Table 3.1 – Excerpt of the OMAP4430 memory map on the L4_PER interconnect.

the processor. All this information is provided by the processor's manufacturer in the so-called memory map [17, Section 2].

The basic configuration of a *BSP* comprises the definition of address offsets and sizes. This information is to tell the *OS* kernel, how the physical address space is partitioned, so that it can initialize the processors *MMU* to protect address regions which should not be accessible by an application.

3.3 VxWorks boot sequence

The *VxWorks* boot sequence starts right after the board boot sequence has completed (see Section 7.7 "Board boot sequence"). The *VxWorks boot loader* itself is designed to be executed from non-volatile ROM memory such as an EEPROM or NAND module. It prepares the execution stack, initializes main memory, unpacks and copies itself into main memory (assembly) and continues execution from there. The next phase of the *boot loader* prepares for the execution of a minimal *VxWorks* kernel. It initializes the the *MMU* and Caches, clears *Block Storage Segment (BSS)* and initializes device drivers, in order to load the complete kernel image from external locations into main memory, such as network, *Secure Digital (SD)* or USB. The *VxWorks* kernel will then re-run the most initialization steps done so far and prepares for the multi tasking kernel.

3.4 Kernel initialization

The following routines are the most crucial functions that are executed to prepare the hardware for the *boot loader* and the kernel itself. While the architecture-dependent functions usrInit() and sysHwInit2() are already provided by *Wind River*, sysHwInit() needs to be implemented by the *BSP* developer.

- usrInit(): It is the first C code that runs within a *VxWorks* image. It initializes the cache, clears the *Block Storage Segment (BSS)* and initializes vectors.

- sysHwInit(): Does early *BSP*-specific initialization. Initializes *I/O pins*, *General Purpose Inputs/Outputs (GPIOs)* and clocks. This is the place where legacy drivers are initialized (see Section 3.5 “Device drivers“).

- sysHwInit2(): This is the first code to run in the multi-threaded kernel context, as the root task. It initializes *VxBus* (see Section 3.5 “Device drivers“) and instantiates the incorporated device drivers.

3.5 Device drivers

Device drivers for VxWorks can be developed in two models, one is the legacy method and the other is using the VxBus system. The current and proper way of writing device drivers is to write them in a VxBus compliant way. However some drivers still need to be implemented in legacy mode, because they are used before the VxBus is up and running.

When writing a legacy driver, the developer creating the driver, defines the communication between the driver and the *OS*. [23, Section 2]. Those drivers are usually very *BSP* specific and their code resides within the *BSP* code and are not likely to be reused by other *BSPs*.

All other drivers follow the *VxBus* rules. There are three important terms in the driver development with VxBus: device, driver and instance. Device is the hardware. Driver is the software configuring, controlling and communicating with the hardware. Instance is the combination of a device and a driver [23, Section 2.3]. *VxBus* defines the infrastructure for device drivers in a platform-independent way. It matches up hardware with their corresponding drivers, controls access for device drivers to the hardware and integrates them into the VxWorks system. Additionally, VxWorks provides tools (see Section 3.6 “Component system“) which can be used in the development phase to include a specific driver and its dependencies. Each driver and its matching hardware is classified in a device driver class. Each class defines a set of methods used to communicate with the specific class of drivers. General classes are: Serial drivers, storage drivers, network interface drivers, Non-Volatile RAM Drivers, timer drivers, DMA controller drivers, bus controlled drivers, USB drivers, Interrupt controller drivers, multifunction drivers, remote processing element drivers, console drivers and resource drivers. The driver itself advertises the supported methods, according to the driver class in use. Those standardized methods permit the *OS* to integrate devices in its components in a platform independent way. To meet hard real time requirements VxBus defines a set of rules which a driver must apply to achieve the required performance for the overall system [23, Section 2.4]. The *BSP* does the instantiation of the drivers, by defining special INCLUDE macros and pass arguments to VxBus: register address offsets, interrupt numbers and other driver specific options.

3.6 Component system

Components included in the *BSP* with INCLUDE macros usually have few to no dependencies to other components. It is the programmers task to resolve dependencies among included components, because there are no compile-time checks. Unresolved dependencies can result in faulty, mostly unrelated behaviour, which is hard to debug.

Therefore, when developing applications for *VxWorks*, a developer works with the component meta description system, which is a collection of *Component Definition Files (CDF)* that define the components dependencies. WindRiver provides tools to set-up *VxWorks* projects based on a *BSP* with additional components and automatically resolve their dependencies. There is a graphical tool built in the *Integrated Development Environment (IDE)* and a command line tool provided for this purpose.

CHAPTER 4

Road-map

With the knowledge of the requirements of the project, tasks have been identified and isolated to reach the projects goals. Tasks are organized in milestones for a better overview of the current development state. A graphical representation of the defined milestones is depicted in Figure 4.1. The individual milestones are described in the following sections.

4.0.1 milestone: preparation

Reading: There are three important documents that need to be studied before starting any practical work. One is the processor manual, also called Technical Reference Manual [17] from *Texas Instruments (TI)*. It is the most comprehensive document with all technical details of the processor, but not including information regarding third-party components. The second document is the *PandaBoard* manual [15], which describes all *Integrated Circuits (IC)* and their connections on the development board. The third and most important document in *VxWorks BSP* development is the VxWorks *BSP* Development Guide [22], describing the fundamentals of a *BSP* and the *VxWorks* kernel.

Reference BSP: The first task in the development of a VxWorks *BSP* is the evaluation of the provided reference *BSPs*. The one fitting best to the processor, will be used for further development. All the given reference *BSPs* are compared with respect to the following criteria:

- Processor architecture
- Single core/Dual core
- Register addresses
- Integrated modules and their functions

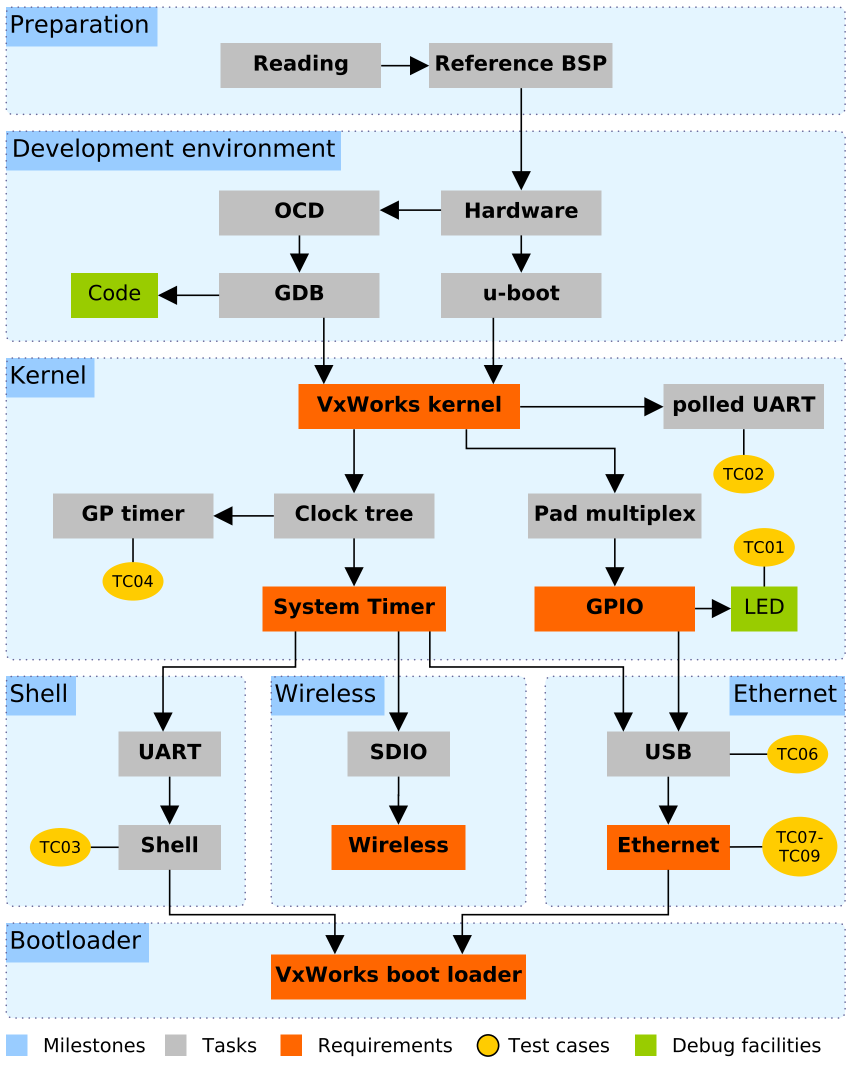

Figure 4.1 – Dependency graph of the project tasks

WindRiver delivers *VxWorks* with reference *BSPs* for the TI AM437X and the TI OMAP3, beside many others.

4.0.2 milestone: Development environment

Hardware: Design and assemble a stable development environment, to reduce environmental factors that may cause floating cables or breaking connections.

OCD: Configure the *Open source On-Chip-Debug (OpenOCD)* software to support the physical *Joint Test Action Group (JTAG)* adapter connected to the *PandaBoard*.

GDB: Set-up *GNU debugger (GDB)* to read binaries produced by the *VxWorks* compiler to inspect code running on the target processor.

u-boot: Adapt the universal *boot loader* to the board's environment to load the kernel image from network, in order to postpone the portation of the official *VxWorks boot loader*.

4.0.3 milestone: Kernel

VxWorks kernel: Complete the port of the VxWorks kernel, to get a reliable environment for device drivers development.

Polled UART: Implement routines that send text to the workstation connected via the *UART*, without having a full fledged *UART* driver.

Pad multiplex: Write a legacy driver (see Section 3.5 "Device drivers") to switch between the different input/output capabilities offered by a single *I/O pin* on the processor.

GPIO: Port an existing and compatible GPIO legacy driver to control individual *I/O pins* functionalities.

Clock tree: The *clock tree* is the clock signal distribution network on the processor, which feeds the individual processor modules with their required clocks. This task addresses the complex *clock tree* by implementing a legacy driver (see Section 3.5 "Device drivers") to enable and configure the required clocks.

GP timer: Port the general purpose timer driver from a reference *BSP*.

System timer: Set-up the first general purpose timer as the kernel's system timer to enable task scheduling.

4.0.4 milestone: Shell

UART: Port the standardized serial driver with interrupt support to enable kernel logging and kernel shell.

Shell: Enable kernel shell and familiarize with it [21].

4.0.5 milestone: Ethernet

USB: Port the USB host driver and implement the required routines for proper initialization.

Ethernet: Port the USB ethernet driver.

4.0.6 milestone: Wireless

SDIO: Port the *Secure Digital Input Output (SDIO)* driver to enable communication to the external wireless LAN module.

Wireless: Port the wireless LAN driver.

4.0.7 milestone: Boot loader

VxWorks boot loader: Ensure the board can boot with the official *VxWorks* boot loader, with support for serial communication and ethernet.

CHAPTER 5

Testing

Testing is performed continuously and more testing possibilities are added with each new device driver that accesses specific hardware to be tested. With increasing number of usable hardware drivers, also the complexity of test cases increases.

In the early stages of the *BSP* development, during the implementation of the *GPIO* driver, a *Light-emitting diode (LED)* can be used to communicate with the developer. The *PandaBoard* features two *LEDs* that can be controlled via *GPIO* and are used to signal that the processor came to a specific line in the code. The low complexity of the *GPIO* driver makes it easy to use the *LEDs* in the very beginning of the boot process, which allows just assembly code to run.

When the basic initialization part is completed and C code can be executed, *UART* communication gets implemented in polled mode. This way text messages can be sent to the console prior to using a regular driver. When the regular *UART* driver is ready, the built-in *VxWorks* command line interpreter can be used, which grants the use of many Unix like programs. There is also a C interpreter that allows to access C functions and variables defined in the code.

Wind River provides a *BSP* validation test suite, which allows a developer to get a quick understanding of the state of the *BSP* [25, Section 1.2.1]. Due to its configuration complexity, it is not used in the early *BSP* development stages but only at the end of the work (see Section 6.9 “BSP Validation Test Suite“).

5.1 Test cases

The following are test cases derived from the requirements (see Chapter 2 “Requirements“). The results of manual execution have been collected in the appendix B (see Chapter 10 “Appendix B: Test results from manual execution“).

TC01: GPIO - LEDs

Execution	Turn both *LEDs* on the *PandaBoard* on and off.
Expected behaviour	Check if the LEDs light are switched on and off correctly.
Covered requirement	RQ01: Kernel

TC02: Polled serial

Execution	Send a string over the serial line to the workstation by manually writing to the UART buffer.
Expected behaviour	Check if the string is received correctly on the workstation.
Covered requirement	RQ01: Kernel

TC03: Shell

Execution	Run a VxWorks shell on the serial line with interrupts enabled. Set the host serial configuration to: 115200 8n1 with neither hardware nor software flow-control.
Expected behaviour	Check if sending commands and receiving output is possible.
Covered requirement	RQ01: Kernel

TC04: Timer simple

Execution	Enable the system timer and enable/disable a LED in a 60Hz interval.
Expected behaviour	Verify the frequency with an oscilloscope or a frequency counter.
Covered requirement	RQ02: Timer

TC05: Timer complex

Execution	Enable an additional timer to toggle a GPIO line on an Expansion Connector of the PandaBoard with a 60Hz, 200Hz, 1KHz, 100KHz frequency.
Expected behaviour	Verify the frequencies of all intervals with an oscilloscope or a frequency counter.
Covered requirement	RQ02: Timer

TC06: USB - Low Speed Keyboard

Execution	Enable USB (INCLUDE_USB) and use the usbTool keyboard test utility to capture keys typed on a USB keyboard.
Expected behaviour	Check that the keys typed on the USB keyboard are displayed on the target shell.
Covered requirement	RQ03: Ethernet

TC07: LAN - UDP

Execution	Send a text inside a UDP packet to the workstation on a specific port.
Expected behaviour	Check if the text is displayed correctly on the workstation.
Covered requirement	RQ03: Ethernet

TC08: LAN - TCP

Execution	Send a text inside a TCP stream to the workstation on a specific port.
Expected behaviour	Check if the text is displayed correctly on the workstation.
Covered requirement	RQ03: Ethernet

TC09: LAN - DHCP

Execution	Let the target device retrieve IP configurations from a DHCP server.
Expected behaviour	Check if the target device gets an IP address assigned.
Covered requirement	RQ03: Ethernet

TC10: WLAN - Connect

Execution	Connect to a WLAN network created by an wireless access point.
Expected behaviour	Check if the target device is discoverable by the wireless access point.
Covered requirement	RQ04: Wireless (optional)

TC11: WLAN - UDP

Execution	Connect to a WLAN network and send a text inside a UDP packet to the workstation.
Expected behaviour	Check if the text is displayed correctly.
Covered requirement	RQ04: Wireless (optional)

TC12: WLAN - DHCP

Execution	Connect to a WLAN network and retrieve IP configuration from DHCP.
Expected behaviour	Check if an IP address has been assigned to the target device
Covered requirement	RQ04: Wireless (optional)

CHAPTER 6

Development Environment

A set of powerful tools and infrastructure is used to ease in the development process. This chapter describes the used hardware and the software tools.

6.1 Hardware

To minimize faults introduced by relocating hardware and plugging cables, it is important to have all the components and utilities always in place and plugs not getting moved so that faults of this kind don't occur during development. A hard plastic case from an old drilling machine was the most suitable case, which can hold all the hardware needed beside the workstation computer and protect the individual components from physical damage. All components are fixed to the ground plate of the plastic case, so that they cannot move around when moving the case.

The main components include:

- PandaBoard
- Serial to USB converter to connect to the workstation
- *JTAG* adapter
- Router, Ethernet switch and wireless access point
- Power distribution for all the power supplies

The router is used to statically assign IP-addresses to the *PandaBoard* and to the workstation, for FTP boot to work correctly. This ensures a consistent network environment.

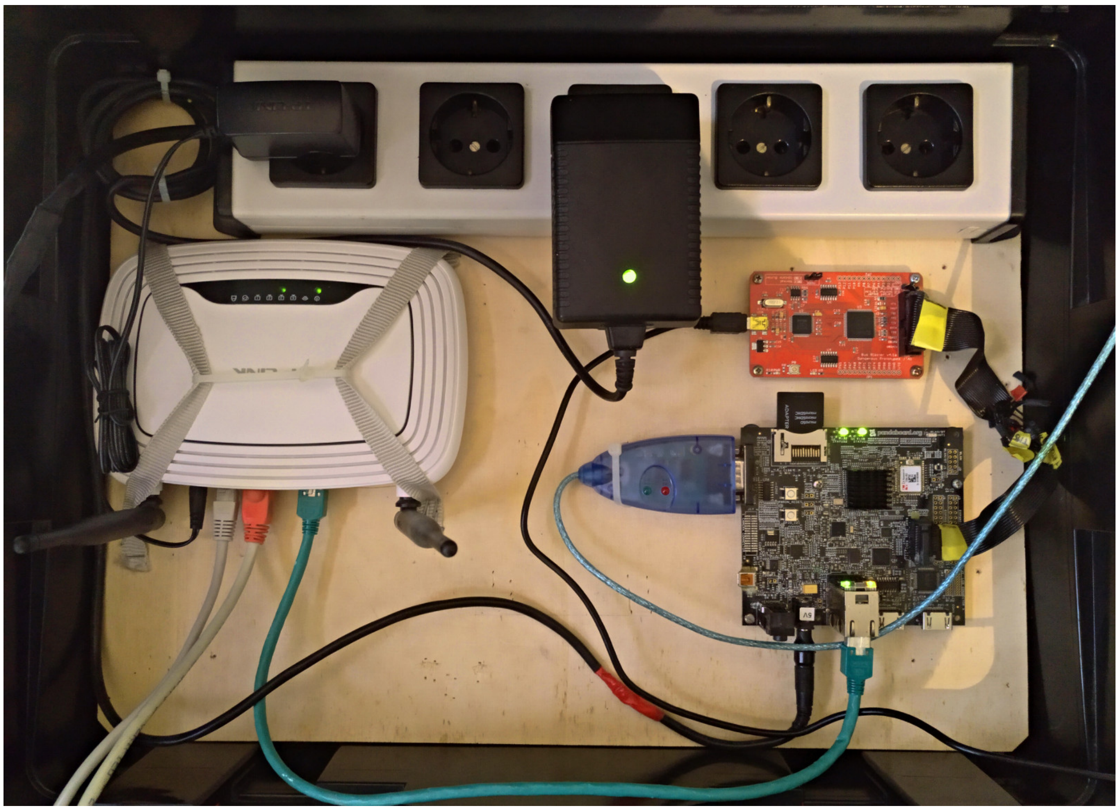

Figure 6.1 – Inside view of the case containing router, JTAG adapter, PandaBoard and power supplies.

6.2 Version control system

A version control system for the source code is is essential for modern software development. *GIT* is used for this purpose.

6.3 Wind River tools

Wind River provides graphical and command line tools to manage software components, compile code and debug. *Workbench* is *Wind River*'s graphical *IDE* based on *Eclipse* with an integrated code editor, wizards for new projects, graphical component management and multiple debug features. If one wants to use its own editor, the command line program *vxprj* can be used to create projects and manage components.

6.4 GNU compiler tools

Generally *VxWorks* supports the GNU compiler collection and the in-house DIAB compiler from WindRiver.

For this project it seemed appropriate to choose the GNU compiler tools, because of their huge community and for their support for the *JTAG* adapter mentioned in the Development Environment chapter. This combination lets the *BSP* developer disassemble code produced by the compiler, program binary code directly on to the processor's RAM, to set breakpoints, to step through code lines and to inspect memory.

6.5 TI tools

TI offers two quite useful tools on their discontinued product website. One of the tools is targeted to the complex clock tree inside the processor and is called Clock Tree Tool (CTT). It shows the clock lines in a graph and the connected components. It is quite handy when it comes to debug clock related issues. The second tool focuses on the processor's pad configuration and is called Pad Configuration Tool (PCT).

6.6 OCD

Mostly all *System-on-a-chip (SoC)* architectures and many other *ICs* offer at least a *JTAG* port to inspect memory contents, program counter and examine stack traces. *PandaBoard* features a *JTAG* port which connects to the Cortex-A9 CPUs, the Cortex-M3 CPUs and the *Digital Signal Processor (DSP)*. A *JTAG* connection to the workstation combined with GNU debugger becomes a powerful and indispensable chain of tools.

The *BusBlaster*, a *JTAG* adapter from the low-cost tools supplier Dangerous Prototypes, was chosen for its compatibility with *OpenOCD*. The manufacturer also offers helpful documentation to configure *OpenOCD* for the *PandaBoard* (see Section 7.4 “OpenOCD configuration“).

6.7 Polled serial

Since U-Boot is used to load the VxWorks kernel, the *UART* is already initialized for the communication to the workstation. By the given pre-initialized components, performed by *U-Boot*, it is possible to directly write to the UART output registers to send simple text messages to the workstation. This method is called „early printk“.

When the kernel boots, the *UART* gets reinitialized and its *VxBus* driver gets loaded. From this point on, kernel functions are used to print debug messages to the console.

6.8 VxWorks shell

When a functional kernel is running a *VxWorks* shell can be used over a serial or a network connection to do immediate modifications or instrumentations in the *OS*. *VxWorks* offers a wide range of commands, that can be individually enabled as components (see Section 7.8 “Component system“).

6.9 BSP Validation Test Suite

Wind River offers multiple tests, that can either be run from command line or within the *Workbench*. It is a collection of scripts that run on the development machine and download a special testing kernel onto the device. This requires a serial communication interface and a working ethernet connection. The test results of the current state of the BSP have been collected in the source code repository [1].

CHAPTER 7

Pitfalls

7.1 Initial readings

Wind River provides useful manuals to get an understanding of how a *VxWorks BSP* is supposed to work and how devices interact with the kernel: [22] [23] [24].

7.2 NO_SOURCE license

The WindRiver university program gives students and professors the opportunity to teach, graduate and to research based on their products. The type of license gathered from WindRiver in the context of their academic program is a special *NO_SOURCE license*. This license includes sources for reference *BSPs* and sources for device drivers, WindRivers *IDE*, tools, but no sources for the kernel itself. It turned out that kernel sources are very important when it comes to develop the lowest layers of this *OS*: when the processor executes kernel code, which gets only shipped in binary form, a developer just has its disassembled code to debug. Also *Workbench* doesn't support *On-Chip-Debugging (OCD)* functionality under this kind of license. Another drawback is that the documentation of a library often refers to its source code, which as mentioned is not available [4].

7.3 WindRiver Support Network

The WindRiver support network is an on-line portal, which is a platform for a knowledge base, ticket support and forums.

The knowledge base contains documentation, tutorials, defects and security considerations for all WindRiver products and tools. Documentation is provided directly on the website or can be downloaded as PDF files.

Like many other companies, also WindRiver provides individual customer support for its software products. With the university program license, one gets an enterprise support license, which allows to create support tickets that are processed by a professional employee. While developing this *BSP* several issues were discovered and submitted: [7] [8] [6] [4] [9] [10] [12].

7.4 OpenOCD configuration

OCD combined with *GDB* is the software used to communicate directly with the processor using *JTAG*. *OCD* alone is not sufficient because it will not be aware of processor exceptions, function stacks a.s.o. It is the task of *GDB* to handle processor exceptions while stepping through code or read in symbol names to be used instead of plain addresses.

There is some helpful documentation on *Dangerous Prototypes (DP)* site [3] to configure OpenOCD [14] with this type of *JTAG* adapter. All other project specific configurations are collected in files in the project repository and described in the respective README file [5].

7.5 Debugging symbols

Recompiling the libraries provided by WindRiver with debugging options allows the compiler to include debugging information into the resulting binary image. This debugging information can then be parsed by *GDB* to show human readable names of functions and show the exact location in the code. Each library uses different preprocessor macros to control debugging, but not all libraries are recompilable (see Section 7.2 “NO_SOURCE license“).

7.6 Creating a bootable SD-Card

The processors internal boot ROM supports only a specific partition structure for the *SD* card [11, see also Section A.1 „Create a bootable SD card“]. Newer partitioning tools on *Linux* don’t allow this specific structure to be created. Thus it is convenient to use an adapted *SD* card image for the *PandaBoard*. PandaBoard offers a validation image, which can be downloaded from their website [16]. It already contains the *U-Boot boot loader*, the Linux kernel and a root file system.

7.7 Board boot sequence

The *PandaBoard* is limited to boot from *SD*, because the size of ROM memory is not sufficient to fit a *OS boot loader* [15, Section 2.6]. Therefore the processor’s internal, pre-programmed *Boot ROM* (shown in Figure 2.1 as „Boot/secure ROM“) directly loads the *X-Loader* from *SD* (MLO file) into the internal SRAM (48KB [17, 27.4.2.2]). The

X-Loader prepares DDR RAM and loads the real *boot loader* into it and executes it. This is where either *U-Boot* or the *VxWorks boot loader* comes in to play [19].

The initial VxWorks *boot loader* had no support for communication devices and thus it was impossible to load the kernel image, except via *JTAG*. Therefore, it was reasonable to replace the VxWorks *boot loader* with *U-Boot*, which is known to work on the *PandaBoard*. One of the advantages of *U-Boot* is the availability of basic device drivers, such as USB and the *Local Area Network (LAN)* driver, to facilitate the use of the USB LAN device to transfer a *Standalone-VxWorks image* into RAM. Newer *VxWorks* 7 kernels have support for the device trees and can be booted using the same procedure as they are used to boot Linux.

7.8 Component system

BSP development is performed in a BSP/bootloader project. All drivers and components used in the *BSP* are enabled by defining INCLUDE macros. Application development is done in a *VxWorks Image Project (VIP)* project, where components are included with dedicated tools. Those tools take the component definitions of the *BSP* and resolve dependencies between components. In a BSP/bootloader project, the developer needs to take care of resolving the dependencies. Otherwise compilation can result in undefined reference errors, crashes of running tasks or the CPU jumping to unknown locations.

7.9 UART flow control

Hardware flow control needs to be disabled when doing early printk using a serial terminal, because the driver that handles flow control is not yet loaded.

7.10 UART interrupt number

The interrupt ID numbers cannot directly be taken from the documentation [17, Section 17.3.1] and transferred to the *BSP*. The reason for this is that the register layout for the interrupt controller in a multi-core set-up differs from a processor with just one core [2, Section 2.2.1]. Multi-core processors are supported in *VxWorks* by using the *Symmetric Multi Processing (SMP)* functionalities of the kernel. To make interrupts compatible when not using *SMP*, every interrupt number from the processor's reference manual needs to be shifted by adding the decimal number 32. A support case has been created to cross-check this with a WindRiver employee [7].

7.11 Power management

The TWL6030 chip (see the TWL6030 block shown in Figure 2.1) is responsible to drive all the power rails on the board, including USB and *SD* card power. This *IC* is connected through an *Inter-Integrated Circuit (I^2C)* interface to the processor. The processors internal logic uses this interface to control the external *Switching Mode Power Supply (SMPS)* inside the TWL6030, to react on changes to the different power domains. Which means that there is no need for a *I^2C* driver for a functional power management. But such a driver is needed when using other functionality of the TWL6030 (e.g. sensing the *SD* card slot).

The *BSP* doesn't include support for the processor's power domains and thus there is no idle task in the *OS* that puts the processor in low power consumption mode. The processor is thus running at full power all the time, which results in a high power dissipation (see Section 7.14 "Temperature too high").

7.12 USB via ULPI

The USB controller connected via ULPI needs a special initialization sequence. Information has been collected from the *U-Boot* implementation and *Linux* implementation of the driver. Additionally a *TI* support case helped [13].

7.13 Ethernet driver

The Ethernet driver is a special case of driver, because the *IC* is connected to the processors USB bus. No other reference *BSP* uses USB ethernet devices, thus it was impossible to get a working example for a driver instantiation. VxWorks provides drivers for the USB host controller and defines different interfaces for the communication with USB network devices and ships with a USB LAN library. The problem was that there was not enough documentation available in order to create such a driver from scratch. Therefore contacting the support was needed to proceed: even though the given *NO_SOURCE license* doesn't include code for such drivers, the *Wind River* support team provided the required code [6]. This code was then adapted to the LAN9514 *IC* according to the existing *U-Boot* LAN driver for the same *IC*.

7.14 Temperature too high

The link status of the Ethernet *PHY* is queried using *Media Independent Interface (MII)* operations over USB. The *VxWorks USB ethernet driver layer (usb2End)* layer provides this functionality and lets the user specify link status register offset and mask to query the status. Different USB Ethernet manufactures provide different ways to issue *MII* operations for the *PHY*. Due to this fact, the *usb2End* couldn't gather the link status successfully and caused the internal link status routine to be executed in a high priority kernel task in a loop. Since the processor and its components are always running at full power (see Section 7.11 "Power management"), the dominant status link task caused the processor to heat up to its maximum allowed temperature and issued itself a reset. Applying a heat-sink on the processor provided more time to debug the problem and implementing the appropriate link status routine fixed it.

7.15 Running DHCP client on the USB ethernet interface

Running the DHCPC client directly doesn't work. No output is returned from the command, no IP-address gets assigned to the interface and no frames are leaving the physical interface. The customer support clarified that DHCP needs to be enabled via the *ifconfig* command line tool and cannot be executed directly [10] .

7.16 USB ethernet not always detected correctly

Sometimes the USB Ethernet/hub LAN9514 is not detected correctly, which leads to unusable network functionality in *VxWorks*. This is a known problem which has been submitted to *Wind River*, but has marked as "Not to be Fixed" [27].

7.17 TCP retransmission errors and ACK DUP

The USB Ethernet driver was not fully functional and lost frames, which resulted in retransmission of frames that arrived out of sequence. To debug the issue, the base driver library would need to be recompiled with DEBUG flags, but the *NO_SOURCE license* doesn't allow it (see Section 7.2 “NO_SOURCE license“). By adding early printk routines to the driver code, the problem could be analysed: The driver was setting a turbo mode on the Ethernet device, which lets the device send multiple ethernet frames in one USB packet. Handling multiple Ethernet frames in one packet is possible, but handling Ethernet frames that are split into two USB packets is not supported by the underlying USB base ethernet driver. An additional buffer would be needed to collect parts of a frame separately. Therefore the turbo mode has been disabled, so each Ethernet frame gets sent in one USB packet.

This problem was only seen with TCP communication, since UDP frames would not exceed the maximum USB packet size.

7.18 WDB

The *Wind River Debug Bridge (WDB)* is required to take advantage of the *BSP Validation Test Suite (VTS)* and *Workbench* debugging techniques. *WDB* can communicate over a second serial interface or over Ethernet. The board does not provide a second RS-232 port (although other *UARTs* are exposed over pin headers on the *PandaBoard*), so there was the need to run the *WDB* over the given USB Ethernet. Running *WDB* over a USB enabled Ethernet device leads to different restrictions: *WDB* can only be used for task debugging and not for system debugging and the bootline needs to contain the flags to discover an IP-Address from a DHCP server [12].

7.19 Wireless driver

The driver for the *TI* TiWi wireless LAN and Bluetooth module connected to the *SDIO* port of the processor is for sure the most difficult part of this project (see the corresponding “WiLink mWLAN“ block in Figure 2.1). There are a number of reasons, why porting that driver is unrealistic. The most obvious reason is that the *NO_SOURCE license* doesn’t cover wireless applications. WindRiver sells their wireless driver as a separate product, which exceeds this project’s budget [9]. There exists a driver for Linux, but porting it is unrealistic: 44 C source files, 52 header files with a sum of 28906 lines of code. Which would additionally depend on other Linux libraries like the MAC80211 layer.

CHAPTER 8

Conclusion

The outcome of this project is a combination of a popular hardware platform based on the *PandaBoard* with the proprietary industrial grade *RTOS VxWorks* (used in many professional applications, like the Mars rover Curiosity [26]), communicating over GPIO, serial line, USB and Ethernet. In the whole development process, valuable knowledge was gained to debug the *PandaBoard* with cheap hardware (*BusBlaster* < 50 €) and open-source tools, even though information is not spread out on the Internet as for other open-source software.

With some additional effort, *SD* card support could be integrated, so that the *VxWorks boot loader* can load the kernel image directly from *SD*, without the need for *U-Boot*.

After reading several documents and reviewing numerous source code lines from *TI* and the *Linux* kernel, the effort for the portation of the wireless LAN driver for the *PandaBoard* was estimated as way too high for the context of this thesis.

Due to the nature of the *NO_SOURCE license* (see Section 7.2 "NO_SOURCE license"), important kernel/driver debugging functionality could not be used, because they require the recompilation of the libraries. This indicates that Wind River's university program targets application developers, but not *BSP* developers. Yet, the goal for this project was accomplished, since even without wireless LAN functionality, the platform can be used to teach real time applications based on *VxWorks* without the need for expensive hardware. In addition, this paper is valuable when porting *VxWorks* to other hardware platforms like the *RaspberryPI* or the *BeagleBoard*.

CHAPTER 9

Appendix A: VxWorks PandaBoard BSP manual

DESCRIPTION

TI OMAP4430 PANDA

User manual for the VxWorks 6.9 Board Support Package for the PandaBoard A4.

INTRODUCTION

This reference entry provides board-specific information necessary to run VxWorks for the PandaBoard. These boards feature the Texas Instruments OMAP4430 processor with two ARM Cortex-A9 cores (only one core supported by this BSP).

The PandaBoard can be booted from SD card only. Since SD is not yet supported by this BSP, there is the need to first run the universal boot loader u-boot in order to get the VxWorks kernel over FTP on the device.

VxWorks bootrom is supported by this BSP, but it cannot load the kernel from SD card, so it is useless when developing applications.

The tested and working boot sequences are:

MLO > u-boot > vxWorks.bin (SD, TFTP)
MLO > u-boot > bootrom.bin (SD, TFTP) > vxWorks (TFTP)
MLO > bootrom.bin > vxWorks (TFTP)

The explanation of the individual files involved in the boot process are listed in the following table.

File	Description
MLO	Second-stage boot loader
u-boot	Universal boot loader
bootrom.bin	VxWorks boot loader
vxWorks.bin	VxWorks kernel as standalone binary image
vxWorks	VxWorks kernel as ELF image

The instructions of how to prepare the boot loader and kernel are described in the following "Boot Process" section.

NOTE

This BSP is in development state and needs to be reviewed by a professional BSP developer.

The sections "VxWorks boot loader methods" and "Creating an active partition on a SD card from Windows" are entirely written by Wind River and have been copied from the reference BSP.

BOOT PROCESS

Boot settings for the PandaBoard

The boot sequence is fixed to boot from SD card on the PandaBoard, so there are no switches that need to be set.

Below is a listing of the available boot sequences and the corresponding sections to follow.

MLO > u-boot > vxWorks.bin (SD)

1) Create a bootable SD card
2) Compile u-boot and copy to SD card
3) Create a serial console connection
5) Compile a VxWorks standalone kernel image
8.1) Run the VxWorks kernel with u-boot (SD)

MLO > u-boot > vxWorks.bin (TFTP)

1) Create a bootable SD card
2) Compile u-boot and copy to SD card
3) Create a serial console connection
5) Compile a VxWorks standalone kernel image
8.2) Run the VxWorks kernel with u-boot (TFTP)

MLO > u-boot > bootrom.bin (SD) > vxWorks (TFTP)

1) Create a bootable SD card
2) Compile u-boot and copy to SD card
3) Create a serial console connection
4) Compile and run the native VxWorks bootloader
5) Compile a VxWorks standalone kernel image
7.1) Run the VxWorks bootloader with u-boot (SD)
9) Load the VxWorks kernel image with the native VxWorks bootloader via TFTP

MLO > u-boot > bootrom.bin (TFTP) > vxWorks (TFTP)

1) Create a bootable SD card
2) Compile u-boot and copy to SD card

3) Create a serial console connection
4) Compile and run the native VxWorks bootloader
5) Compile a VxWorks standalone kernel image
7.2) Run the VxWorks bootloader with u-boot (TFTP)
9) Load the VxWorks kernel image with the native VxWorks bootloader via TFTP

MLO > bootrom.bin > vxWorks (TFTP)

1) Create a bootable SD card
2) Compile u-boot and copy to SD card
3) Create a serial console connection
4) Compile and run the native VxWorks bootloader
5) Compile a VxWorks standalone kernel image
6) Run the VxWorks bootloader without u-boot
9) Load the VxWorks kernel image with the native VxWorks bootloader via TFT

1) Create a bootable SD card

The boot sequence on the PandaBoard is fixed to boot from SD card. Therefore a bootable SD card is needed. Since it is not always easy to create such a SD card from scratch, it is best to start with the validation image provided by TI or Pandaboard. If it cannot be found do the following steps (Assuming you are using a UNIX based workstation. Otherwise follow the steps described in the "Creating an active partition on a SD card from Windows" section at the end of this manual.

```
# Create the partition table
echo "start=63,size=36MB,type=c,bootable" | sudo sfdisk /dev/sdc

# Assuming your SD card is at /dev/sdc. This creates
# a bootable WIN95 OSR2 FAT32, LBA-mapped partition
# /dev/sdc1 starting at sector 63 with the smallest
# possible FAT32 partition of 32MB.

# Create a FAT32 partition called VXBOOT:

sudo mkfs.fat -F 32 -n "VXBOOT" /dev/sdc1

# Mount the SD card
sudo mount /dev/sdc1 /mnt/sdcard
```

2) Compile u-boot and copy to SD card

```
# Get the cross compile toolchain
# Download the cross compile toolchain with
# gcc 4.9.4 from linaro for your machine and
# unpack it and add it to your PATH variable

wget https://releases.linaro.org/components/toolchain/binaries/latest-4/ar
tar -xf gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabi.tar.xz
cd gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabi
export PATH=$PATH:`pwd`/bin

# Get the u-boot sources from linaro and checkout the 2011.11.2 tag
git clone https://git.linaro.org/boot/u-boot-linaro-stable.git
git checkout -b vxWorks 2011.11.2

# Configure and build
export ARCH=arm
export CROSS_COMPILE=arm-linux-gnueabi-
make distclean
make omap4_panda_config
make

# Copy the MLO and bootloader to the mounted SD card
cp MLO u-boot.img /mnt/sdcard
sync
sudo umount /mnt/sdcard
```

3) Create a serial console connection with the PandaBoard

Connect your workstation's serial port or use a USB to serial adapter and connect it to the PandaBoards RS-232 serial port.

Run a serial communication program at 115200 8N1 with no hardware flow control

```
minicom -b 115200 /dev/ttyS0
```

4) Compile the native VxWorks bootloader

```
# Compile the bootloader with the following
# command inside the BSP directory
cd ti_omap4430_panda
make bootrom.bin
```

```
# Copy the bootloader to the TFTP environment
# to later load it via TFTP
cp bootrom.bin /tftpboot

# Copy the bootloader to the SD card
cp bootrom.bin /mnt/sdcard
```

5) Compile a VxWorks standalone kernel image

It is possible to compile the VxWorks standalone image directly in the BSP directory, but it will not feature any shell or peripheral drivers. So it is best to create a VxWorks VIP project and enable some basic kernel components.

To perform this, create a VIP project with Wind River Workbench or Wind River command line tools and add the following components:

```
INCLUDE_SHELL
INCLUDE_SHELL_BANNER
INCLUDE_USB_KEYBOARD
INCLUDE_USB_SHOW
INCLUDE_IFCONFIG
INCLUDE_IPIFCONFIG_CMD
INCLUDE_IPCOM_SHELL_CMD
INCLUDE_IPPING_CMD
INCLUDE_PING
INCLUDE_IPTELNETS
INCLUDE_IPDHCPC
```

Now compile a VxWorks standalone binary image by issuing

```
# This will generate the files: vxWorks and vxWorks.bin
make vxWorks.bin

# Copy the ELF kernel image to the TFTP environment
cp vxWorks /tftpboot

# Copy the standalone kernel image to the
# TFTP environment and SD card

cp vxWorks.bin /tftpboot
cp vxWorks.bin /mnt/sdcard
```

6) Run the VxWorks bootloader without u-boot

The bootrom.bin binary image contains a header usually read by the MLO (u-boot) to know where the image should be copied to. Which is only used if the u-boot.img file is overwritten by bootrom.bin.

Run the following commands to copy the bootloader on the SD card

```
mount /dev/sdc1 /mnt/sdcard
rm /mnt/sdard/u-boot.img
cp bootrom.bin /mnt/sdcard/u-boot.img
umount /mnt/sdcard
```

Stick the SD card back in to the PandaBoard and boot. This will directly start the native VxWorks bootloader, which is limited to load the VxWorks kernel from TFTP. So be sure to have the VxWorks kernel image inside your workstations TFTP environment.

7) Run the VxWorks bootloader with u-boot (SD/TFTP)

The VxWorks bootloader can either be loaded from SD card or from TFTP. Perform one of the following steps.

7.1) Run the VxWorks bootloader with u-boot (SD)

```
# Load the VxWorks bootloader from SD card and start it.
# This assumes that bootrom.bin has been copied
# to the SD card before.

# Run the following commands in the u-boot shell:

mmc rescan 0
fatload mmc 0:1 BFBFC000 bootrom.bin
go BFC00000
```

7.1) Run the VxWorks bootloader with u-boot (TFTP)

```
# This assumes that bootrom.bin has been copied to the TFTP environment be

# Run the following commands in the u-boot shell
# and replace DEVICE_IP with the PandaBoards IP
# address and replace WORKSTATION_IP with your
# workstations IP address.
```

```
set ipaddr DEVICE_IP
set serverip WORKSTATION_IP
usb start
tftpboot BFBFC000 bootrom.bin
go BFC00000
```

When using u-boot, one can load either the VxWorks bootloader or the VxWorks kernel image directly into RAM and start it from there.

8) Run the VxWorks kernel with u-boot (SD/TFTP)

With u-boot the VxWorks kernel can be loaded from SD card or from TFTP. Choose one of the following steps and run them in the u-boot shell.

8.1) Load the VxWorks kernel from SD card and start it

```
# This assumes that vxWorks.bin has been
# copied to the SD card before.

mmc rescan 0
fatload mmc 0:1 80100000 vxWorks.bin
go 80100000
```

8.2) Load the VxWorks kernel from TFTP and start it

```
# This assumes that vxWorks.bin has been copied
# to the TFTP environment before.

set ipaddr DEVICE_IP
set serverip WORKSTATION_IP
usb start
tftpboot 80100000 vxWorks.bin
go 80100000
```

9) Load the VxWorks kernel image with the native VxWorks bootloader via TFTP

When the VxWorks bootloader is running and the usb network interface has been enumerated, you can setup the boot parameters to load the vxWorks ELF image from your workstation host via TFTP.

Change boot parameters: When inside the VxWorks bootloader you can press **c** to change the boot parameters.

Boot the VxWorks kernel: This assumes that vxWorks ELF binary has been copied to the TFTP environment before.

When the boot parameters have been set correctly, boot the kernel by pressing **@**

VxWorks boot loader methods

The boot methods are affected by the boot parameters. If no password is specified, RSH (remote shell) protocol is used. If a password is specified, FTP protocol is used, or, if the flag is set to 0x80, TFTP protocol is used.

FEATURES

This section describes all features of the board, supported or not. It documents all configurations of the board and the interaction between features and configuration items.

List of Hardware Features

Hardware Interface	Controller	Driver/Component	Status
UART:0	on-chip	**vxbNs16550Sio.c**	SUPPORTED
INTC	on-chip	**vxbArmGenIntCtlr.c**	SUPPORTED
TIMER	on-chip	**vxbOmap35xxTimer.c**	SUPPORTED
GPIO	on-chip	**sysGpio.c**	SUPPORTED
EDMA	on-chip	**vxbTiEdma3.c**	INCOMPLETE
USB ETHERNET	on-chip	**usb2Lan9514.c**	SUPPORTED
CLOCK	on-chip	**omap44xxClock.c**	SUPPORTED
PINMUX	on-chip	**omap44xxPadConf.c**	SUPPORTED
MMCHS0/MMCHS1	on-chip	**vxbTiMmchsCtrl.c**	SUPPORTED
USB-Host	on-chip	**vxbPlbUsbEhci.c**	SUPPORTED
TWL6030	on-board	n/a	UNSUPPORTED
I2C	on-chip	n/a	UNSUPPORTED
USB-OTG	on-chip	n/a	UNSUPPORTED
NAND FLASH	on-chip	n/a	UNSUPPORTED
MMCHS2	on-chip	n/a	UNSUPPORTED
SPI:0	on-chip	n/a	UNSUPPORTED
SPI FLASH	on-chip	n/a	UNSUPPORTED
DCAN:1	on-chip	n/a	UNSUPPORTED
LCD	on-chip	n/a	UNSUPPORTED
MAILBOX	on-chip	n/a	UNSUPPORTED
SPINLOCK	on-chip	n/a	UNSUPPORTED

Hardware Interface	Controller	Driver/Component	Status
SGX	on-chip	n/a	UNSUPPORTED
HDMI	on-chip	n/a	UNSUPPORTED
McBSP	on-chip	n/a	UNSUPPORTED
McASP	on-chip	n/a	UNSUPPORTED

List of features (ARM Cortex-A9 CPU)

Module	Status
FPU(VFPv3)	SUPPORTED
L2 cache	SUPPORTED
MMU	SUPPORTED

HARDWARE DETAILS

This section documents the hardware elements on the PandaBoard and the use of the supported drivers.

Verified Hardware

This BSP has been verified on the PandaBoard.

Verified board information are listed below:

```
PandaBoard OMAP4430        | PandaBoard Rev. A3
```

Memory Map

This is the memory mapping and shows how memory is partitioned when booting up the board. Sections that are aligned to the right are used in the boot process and are overwritten by the kernel. This mapping doesn't show the mapping for u-boot when it executes.

```
+-------------+ <- 0xBFFFFFFF End of DRAM address space (Q2 1GB)
|             |
| bootrom.bin |
|             |
| vxWorks.bin |
|             |
|    +--------|
|    | u-boot |
+----+--------+ <- 0xBFC00000
|    | u-boot |
|    | header |
+    +--------+ <- 0xBFBFC000 - u-boot image
|             |
|             |
|             |
|    Memory   |
|     pool    |
|             |
|             |
|             |
|    +--------+
|    |  .bss  |
|    +--------+
|    | .data  |
|    +--------+
|    | .text  |
+----+--------+ <- 0x81000000 Boot image
|     .bss    |
+-------------+
|    .data    |
+-------------+
|    .text    |
+-------------+ <- 0x80100000 System image
| / / / / / / |
+-------------+
|Exception msg|
+-------------+ <- 0x80001200
|             |
+-------------+ <- 0x80001100
| / / / / / / |
+-------------+ <- 0x80000000 Start of DRAM address space (Q2 1GB)
```

Serial Configuration (UART)

The universal asynchronous receiver/transmitter serial interface is compatible with the industry standard TL16C550/15C750.

UART 3 is used as the serial console

No special configuration needed. Default configuration is:
Baud Rate: 115200
Data: 8 bit
Parity: None
Stop: 1 bit
Flow Control: None

The baud rate is set to 115200 by default, other UARTs are not enabled by default.

Timers

There are 12 full featured general(timer1 ~ timer11), timer0 is used as the system clock timer and thus not available as general purpose. Use VxWorks system libraries to make use of the timers.

Timer support is enabled by the following component (enabled by default):

```
#define DRV_TIMER_OMAP35XX
```

or

```
vxprj component add DRV_TIMER_OMAP35XX
```

L2 Cache

This release of BSP supports L2 cache (PL310). Define **INCLUDE_L2_-CACHE** in **config.h** to enable L2 cache support. L2 cache support is enabled by default.

L2 Cache support is enabled by the following component (enabled by default):

```
#define INCLUDE_L2_CACHE
```

or

```
vxprj component add INCLUDE_L2_CACHE
```

GPIO

General purpose input/output is supported in this release of BSP. To use GPIO, make sure the following component is defined in **config.h** before you create the VIP, or add it in your VIP dynamically.

GPIO support is enabled by the following component (enabled by default):

```
#define INCLUDE_OMAP4430_GPIO
```

or

```
vxprj component add INCLUDE_OMAP4430_GPIO
```

Documentation for the GPIO module can be found inside the header of the file "**sysGpio.c**".

GPIO pin usage on the PandaBoard (**pandaboard.h**):

module	pad	function
GPIO	**GPIO_7**	LED D1
GPIO	**GPIO_8**	LED D1
GPIO	**GPIO_121**	S2 push button
GPIO	**GPIO_1**	USB Hub power
GPIO	**GPIO_62**	USB Hub reset

USB Host

The PandaBoard features a USB3320C USB Phy connected to USBB1 via ULPI, which is directly connected to the LAN9514 USB hub and Ethernet IC.

While in the created VIP, components for EHCI need to be enabled. To use different USB devices, USB class drivers need to be enabled too. Please refer to USB programmer guide for more details to enable different kinds of USB class driver.

Enable USB Host mode controller (Enabled by default in this BSP)

```
#define INCLUDE_USB
#define INCLUDE_EHCI_INIT
#define INCLUDE_HCD_BUS
```

or

```
vxprj component add INCLUDE_USB
vxprj component add INCLUDE_EHCI_INIT
vxprj component add INCLUDE_HCD_BUS
```

USB Ethernet

Ethernet is supported by the LAN9514 IC, connected to the USB hub

Enable the USB Hub/Ethernet driver (Enabled by default in this BSP)

```
#define INCLUDE_USB_GEN2_LAN9514
#define INCLUDE_USB_GEN2_INIT
```

or

```
vxprj component add INCLUDE_USB_GEN2_LAN9514
vxprj component add INCLUDE_USB_GEN2_INIT
```

The USB OTG feature is not supported and the MAC address is currently hard coded. To change the MAC address for the current build, change the following lines in the **usb2Lan9514.h** header file:

```
#define HARDCODED_MAC0  0x2A
#define HARDCODED_MAC1  0x40
#define HARDCODED_MAC2  0x86
#define HARDCODED_MAC3  0xB5
#define HARDCODED_MAC4  0x3B
#define HARDCODED_MAC5  0x0A
```

SPECIAL CONSIDERATIONS

Preemptive Interrupts

Preemptive interrupts are supported by the interrupt driver so **INT_MODE** can be defined as **INT_NON_PREEMPT_MODEL** or **INT_PRE-EMPT_MODEL** in **config.h**.

Make Targets

The make targets are listed as the names of object-format files. Other images not listed here may not be tested.

bootrom
bootrom.bin
VxWorks (with **vxWorks.sym**)
VxWorks.st

BSP Bootloaders and Bootroms

Bootloader/Bootrom	Status
Uboot	shipped
bootrom	SUPPORTED
bootrom_uncmp	SUPPORTED
vxWorks_rom	UNSUPPORTED
vxWorks_romCompress	UNSUPPORTED
vxWorks_romResident	UNSUPPORTED
bootrom_romResident	UNSUPPORTED

Creating an active partition on a SD card from Windows

This section describes how to create an active partition on a SD card from Windows by using DISKPART utility on Windows 7.

The DISKPART utility will be used to correct disks that do not have a partition set to active and format them with a FAT or FAT32 file-system.

a. Insert a SD card to host and find the SD card name.

b. In a DOS command shell run the following commands to ensure SD card is Configured with an active partition.

   Execute **diskpart.exe** as administrator, and then list the disks.

NOTE: Commands below are indicated after the ‘DISKPART>" prompt.

```
DISKPART> list disk

  Disk ###  Status         Size     Free     Dyn  Gpt
  --------  -------------  -------  -------  ---  ---
  Disk 0    Online          465 GB  1024 KB
  Disk 1    Online         1907 MB      0 B
```

Observe SD card number in the displayed list (SD card is Disk 1) and select disk 1

```
DISKPART> select disk 1

Disk 1 is now the selected disk.
```

List disks again to make sure you have the correct drive selected, indicated by the "*" character.

```
DISKPART> list disk

  Disk ###  Status         Size     Free     Dyn  Gpt
  --------  -------------  -------  -------  ---  ---
  Disk 0    Online          465 GB  1024 KB
* Disk 1    Online         1907 MB      0 B
```

Clean the disk, create primary partition and make it active as indicated below.

```
DISKPART> clean

DiskPart succeeded in cleaning the disk.

DISKPART> create partition primary

DiskPart succeeded in creating the specified partition.

DISKPART> select partition 1
```

```
Partition 1 is now the selected partition.

DISKPART> active

DiskPart marked the current partition as active.
```

Now format the disk as fat or fat32. It will take about 2 minutes,
depending on the disk size.

when FAT16 file-system is desired:

```
DISKPART> format fs=fat

  100 percent completed

Diskpart successfully formatted the volume.
```

when FAT32 file-system is desired:

```
DISKPART> format fs=fat32

  100 percent completed

Diskpart successfully formatted the volume.
```

Assign a drive letter and exit.

```
DISKPART> assign

DiskPart successfully assigned the drive letter or mount point.

DISKPART> exit
```

Note the drive letter that is assigned to the SD card.

Known Issues

1) The MAC address of the USB ethernet is hard coded. Since the USB Ethernet IC doesn't feature an EEPROM, the MAC address should be calculated out of hardware/ system signatures.

2) TWL6030 power management IC is not supported and thus no sleep modes are supported. This leads to a high working temperature of the processor. Mount a heat sink on top of the processor package to better transfer the heat to the environment.

BOARD LAYOUT

```
+-------+-----------+----------------------------------+
|       |  SD/MMC   |    |-|     |-|                   |
|       |   CARD    |    +-+     +-+              +-+  |
+--+    |           |  STATUS   STATUS            | |  |
|  |    +-----------+   LED2     LED1       WLAN/BT| |  |
|  |                                              | |  |
|  | RS232 (P4)                                   +-+  |
|  |                                                   |
|  |        +--+                              +-+ +-+  |
+--+        |  | PWRON_RESET                  | | | |  |
|           +--+                       +-+    | | | |  |
|                                      | |    | | | |  |
|           +--+                  JTAG | |    | | | |  |
|           |  | GPIO_121              | |    | | | |  |
|           +--+                       +-+    | | | |  |
|                                             | | | |  |
|                                             | | | |  |
|                                             | | | |  |
|                                             +-+ +-+  |
|                                              E   E   |
+--+                                           X   X   |
|| | USB OTG                                   P   P   |
+--+                 USB/Ethernet              B   A   |
|      MIC/           +------+                         |
|    LINE OUT  POWER  |      |                         |
|     +---+     IN    |      |   HDMI+1080p    DVI+D   |
|     |   |    +-+    |      |   +-------+   +-------+ |
|     |   |    | |    |      |   |       |   |       | |
+-----+---+----+-+----+------+---+-------+---+-------+-+
```

BIBLIOGRAPHY

ARMv7-A Architecture Reference Manual

OMAP4430 Multimedia Device Silicon Revision 2.x Technical Reference Manual

OMAP4 PandaBoard System Reference Manual

SEE ALSO

VxWorks Programmer's Guide: Configuration, *VxWorks Programmer's Guide: Architecture Appendix*

CHAPTER 10

Appendix B: Test results from manual execution

ID	**Test case**	**Result**
TC01	TC01: GPIO - LEDs	PASSED
TC02	TC02: Polled serial	PASSED
TC03	TC03: Shell	PASSED
TC04	TC04: Timer simple	PASSED
TC05	TC05: Timer complex	PASSED
TC06	TC06: USB - Low Speed Keyboard	PASSED
TC07	TC07: LAN - UDP	PASSED
TC08	TC08: LAN - TCP	PASSED
TC09	TC09: LAN - DHCP	PASSED
TC10	TC10: WLAN - Connect	SKIPPED
TC11	TC11: WLAN - UDP	SKIPPED
TC12	TC12: WLAN - DHCP	SKIPPED

Table 10.1 – Test results from manual execution

List of Figures

List of Tables

Glossary

BeagleBoard BeagleBoard is a cheap development hardware based on an ARM processor. 27

board A board is a printed circuit board containing the processor and peripherals. 54, 55

boot loader The boot loader is small software part that is needed to start an operating system. 6, 11, 22, 23, 27, 56

Boot ROM The Boot ROM code is fixed inside the processor and cannot be altered. It reads some boot pins and decides what to boot. 22

BRDS Building Reliable Distributed Systems (BRDS - 182.704) is a course at the Technical University of Vienna that introduces students into distributed algorithms in embedded systems. [20]. ix, xi, 1, 3

BSP A Board Support Package (BSP) is part of an operating system that connects to the hardware. v, vii, ix, xi, 1, 2, 5–11, 13, 19, 21–24, 27, 56

BSS A Block Storage Segment (BSS) is part of the data segment containing statically-allocated variables.. 6

BusBlaster BusBlaster is a cheap OpenHardware JTAG to USB adapter. 19, 27, 53

CDF Component Definition Files (CDF) are meta files for *OS* components and drivers. 8

clock tree The clock tree defines the distribution of clocks with different frequencies to synchronize coupled modules on the processor. 11

DMA The Direct Memory Access (DMA) is used in the processor to transfer data between devices without producing load. 5

DP Dangerous Prototypes (DP) is the company that produces the *BusBlaster JTAG* USB adapter. 22

DSP A Digital Signal Processor (DSP) is a specific family of processors used to process digital signals. 19

Eclipse Eclipse is a flexible *IDE* written in Java.. 18

GDB The GNU debugger (GDB) is a software to inspect a running processor and therefore used to debug software. 11, 22

GIT GIT is a popular source version control system, formerly developed for the *Linux* source tree. 18

GPIO General Purpose Input/Output (GPIO) is a hardware system to use external I/O pins of the processor for general purpose. e.g. turn a *LED* on and off. 7, 13

I/O pin A I/O pin is a contact point of an *IC* which gets soldered on a printed circuit board and usually offers different functions to be exposed to the processors outside world. 7, 11

I^{2}C Inter-Integrated Circuit (I^{2}C) is a synchronous, multi-master, multi-slave serial bus connecting peripheral *ICs* to the processor. 23, 24

IC An Integrated Circuit (IC) is a collection of electronic circuits on a silicon chip. 9, 19, 23, 24, 54, 55

IDE An Integrated Development Environment (IDE) is a software and a collection of tools to program and to debug software. 8, 18, 21, 54, 56

ifconfig ifconfig is a command line tool in UNIX *OSs* to configure IP network interfaces. 25

interconnect Local bus on the processor to connect to internal peripheral devices. 5

JTAG Joint Test Action Group (JTAG) is a standard to test printed circuit boards and inspect the *ICs* on a *board*. 10, 17, 19, 22, 23, 53, 55

LAN A Local Area Network (LAN) is a local network of computers. 23

LED A Light-emitting diode (LED) is a semiconductor light source. 13, 14, 54

Linux Linux is a popular operating system kernel used in many server systems and embedded systems. 22, 24, 27, 54

MAC The Media Access Control (MAC) layer defined in layer 2 of the ISO/OSI model and controls the access to the physical layer. 54

MII The Media Independent Interface (MII) is used to connect the *Media Access Control (MAC)* with the *PHY*. 24

MMU Memory Management Unit (MMU) is a part of the processor that translates memory access addresses to the physical memory addresses. 4, 6

NO_SOURCE license The NO_SOURCE license is the license provided by WindRiver in the context of WindRivers university program. 21, 24–27

OCD On-Chip-Debugging (OCD) is a method to utilize the processors debug capabilities and thus debug the running software directly on the processor. 21, 22

OMAP4430 Texas Instrument OMAP4430 Dual Cortex-A9 processor. 55, 56

OpenOCD Open source On-Chip-Debug (OpenOCD) is a software to talk to the *JTAG* connected *ICs*. 10, 19

OS A Operating System (OS) is a software that manages hardware and its resources to provide the user with system services in order to run applications.. 1, 2, 4–7, 19, 21, 22, 24, 51, 53, 54, 56

PandaBoard The PandaBoard is a printed circuit *board* featuring the *OMAP4430* processor and many peripherals. Its design is open, which means there exists a hardware documentation including circuit diagram of all the connections on the *board* and the manufacturer of the processor provides an extensive documentation [17]. The processor itself is quite powerful and there exists an official Linux port. ix, xi, 1, 3, 9, 10, 13, 14, 17, 19, 22, 23, 25, 27, 56

PHY The physical layer (PHY) is a *IC* that is in charge of coding and decoding digital signals into analog signals used to communicate with external devices like USB, Ethernet, a.s.o. It is the lowest layer in the ISO/OSI network model. 24, 54

QNX QNX is a real time operating system, like VxWorks. 3

RaspberryPI RaspberryPI is a cheap development hardware based on an ARM processor. 27

RTOS A Real Time Operating System (RTOS) is a operating system for time critical applications. 1, 3, 27, 56

SD A Secure Digital (SD) is a non-volatile memory card. 6, 22–24, 27, 55, 56

SDIO The Secure Digital Input Output (SDIO) interface is a general high speed interface based on the *SD* specification. 11, 26

SMP Symmetric Multi Processing (SMP) is a system of identical processors that access the same memory. 23

SMPS A Switching Mode Power Supply (SMPS) is a efficient electronic power supply, *IC* that controls electric power delivered to a device.. 24

SoC A System-on-a-chip (SoC) is an IC that has many computer functions integrated. 19

Standalone-VxWorks image A Standalone-VxWorks image is kernel image with the bootloader integrated. 23

TI Texas Instruments (TI) is the manufacturer of the *OMAP4430* processor used on the *PandaBoard*. 9, 19, 24, 26, 27

U-Boot Boot loader used on many embedded systems. 19, 22–24, 27, 56

UART A Universal Asynchronous Receiver Transmitter (UART) is a serial communication interface. 5, 11, 13, 19, 25

usb2End The VxWorks USB ethernet driver layer (usb2End) is a software library, responsible for the communication to USB ethernet devices.. 24

VIP A VxWorks Image Project (VIP) is a project type based on a *BSP* and builds application code against the provided pre-compiler libraries. 23

VTS The BSP Validation Test Suite (VTS) tests and validates *VxWorks* core functionality [25].. 25

VxBus VxBus is a *OS* local in-memory bus system that handles initialization and routes access to device drivers. 7, 19

vxprj vxprj is a command line tool to manage *VxWorks* project settings and components. 18

VxWorks *VxWorks* is a proprietary *RTOS* from *Wind River* designed to run on embedded hardware. Its applications are mainly automotive, aerospace, defence, medical devices, transportation and robotic. ix, xi, 1–3, 6, 8–13, 18, 19, 21, 23, 25, 27, 56

WDB The Wind River Debug Bridge (WDB) connects the target device to the workstation host running the target server. It can connect through serial or IP. 25

Wind River Wind River Systems (Wind River) is the company which develops embedded system software. 1, 5, 6, 13, 18, 20, 21, 24, 25, 56

Workbench Workbench is the *IDE* from WindRiver for the development of the *BSP* and VxWorks applications. 18, 20, 21, 25

X-Loader X-Loader is the second *boot loader* loaded from *SD* card to the internal Boot ROM, which then starts *U-Boot*. The X-Loader is also called Secondary Program Loader (SPL) and the file on the *SD* card is called MLO. 22, 23

Bibliography

[1] *ti_omap4430_panda BSP Detailed Test Report.* repo/testresults/suiteResults_ti_-omap4430_panda.html, 2018.

[2] ARM. *ARM Generic Interrupt Controller Architecture Specification.* arm_gic_-architecture_specification_v2.pdf, 2013.

[3] BusBlaster. *Bus Blaster OpenOCD Guide.* http://dangerousprototypes.com/docs/Bus_-Blaster_OpenOCD_guide, 2013.

[4] Florin Hillebrand. *I could not find any documentation about the USB2_END_CONFIG_FLAG structure.* Wind River, https://windriver.force.com/support/CaseReadOnly?id=5001600000vA52gAAC, 2016. Wind River support case number: 00063398.

[5] Florin Hillebrand. *OpenOCD configuration documentation for the PandaBoard.* repo/tools/ocd/README, 2016.

[6] Florin Hillebrand. *Pegasus USB END driver sources.* Wind River, https://windriver.force.com/support/CaseReadOnly?id=5001600000uKQboAAG, 2016. Wind River support case number: 00063020.

[7] Florin Hillebrand. *UART TX interrupt not forwarded to GIC on OMAP4430.* Wind River, https://windriver.force.com/support/CaseReadOnly?id=5001600000uIXUaAAO, 2016. Wind River support case number: 00062543.

[8] Florin Hillebrand. *Update to VxWorks 6.9.4.8 errors.* Wind River, https://windriver.force.com/support/CaseReadOnly?id=5001600000uJ6dIAAS, 2016. Wind River support case number: 00062722.

[9] Florin Hillebrand. *Wireless Driver.* Wind River, https://windriver.force.com/support/CaseReadOnly?id=5001600000vm8RTAAY, 2016. Wind River support case number: 00064015.

[10] Florin Hillebrand. *DHCPC not working.* Wind River, https://windriver.force.com/support/CaseReadOnly?id=5001M000017UpJ1QAK, 2017. Wind River support case number: 00080312.

[11] Florin Hillebrand. *TI OMAP4430 board's target-specific documentation.* ti_-omap4430_panda/target.ref, 2017.

[12] Florin Hillebrand. *WDB target agent with USB GEN2 END driver.* Wind River, https://windriver.force.com/support/CaseReadOnly?id=5001M000017VJPZQA4, 2017. Wind River support case number: 00080453.

[13] Kiyoyuki Manei. *AM3517 EHCI controller ULPI DIR signal remains asserted.* https://e2e.ti.com/support/arm/sitara_arm/f/791/t/233308, 2013.

[14] OpenOCD. *OpenOCD User's Guide.* http://openocd.org/doc/html/, 2016.

[15] pandaboard.org. *OMAP4 PandaBoard System Reference Manual.* http://www.ccs.neu.edu/course/cs4610/docs/Panda_Board_Spec_DOC-21010_-REV0_6.pdf, 2010.

[16] PandaBoard.org. *Troubleshooting.* http://pandaboard.org/content/resources/troubleshooting, 2016.

[17] Texas Instruments. *OMAP4430 Technical Reference Manual*, ap edition, 2010.

[18] Texas Instruments. *OMAP™4 mobile applications platform.* http://www.ti.com/lit/ml/swpt034b/swpt034b.pdf, 2011.

[19] Texas Instruments. *Boot Sequence.* http://processors.wiki.ti.com/index.php/Boot_-Sequence, 2012.

[20] Kyrill Winkler Ulrich Schmid. *Building Reliable Distributed Systems (182.704).* https://ti.tuwien.ac.at/ecs/teaching/courses/brds.

[21] Wind River. *VxWorks Kernel Shell.* vxworks_kernel_shell_users_guide_6.9.pdf, 2010.

[22] Wind River. *VxWorks 6.0 BSP Developer's Guide.* vxworks_bsp_developers_-guide_6.9.pdf, 2011.

[23] Wind River. *VxWorks 6.9 Device Driver Developer's Guide.* vxbus_device_driver_-developers_guide_6.9.pdf, 2011.

[24] Wind River. *VxWorks 6.9 Kernel Programmer's Guide.* vxworks_kernel_programmers_guide_6.9.pdf, 2011.

[25] Wind River. *VxWorks BSP Validation Test Suite.* vxworks_bsp_vts_users_guide_-6.9.pdf, 2011.

[26] Wind River, https://www.windriver.com/announces/curiosity/. *Wind River's VxWorks Powers Mars Science Laboratory Rover, Curiosity*, 2012.

[27] Wind River. *VXW6-16179 : pegasus USB ethernet device might not always be detected correctly in vxWorks 6.9.2.2*. Wind River, https://knowledge.windriver.com/en-us/000_Products/000/020/010/010/040/000_VXW6-16179_2013. Wind River defect ID: VXW6-16179.