

Intel® Quark™ SoC X1000

Software Developer's Manual for Linux*

February 2016

Document Number: 330235-005US



You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein
No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.
The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.
Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting: http://www.intel.com/design/literature.htm

Intel, Quark, and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.
*Other names and brands may be claimed as the property of others.

Copyright © 2016, Intel Corporation. All rights reserved.

Contents

1.0 Introduction 6
1.1 About this Manual 6
1.2 Introduction 6
1.3 Related Documentation 6
1.4 Terminology 7
1.5 Conventions 7

2.0 Platform Overview 8
2.1 Platform Synopsis 8
2.2 SoC Features 9

3.0 Software Overview 10
3.1 High-Level Software Architecture Overview 10
3.2 Linux* Support 11
3.3 User-Space Software Dependencies 11

4.0 Intel® Quark™ SoC X1000 Drivers 12
4.1 Overview 12
4.2 USB OHCI Controller Interface Driver 12
4.3 USB 2.0 EHCI Controller Interface Driver 13
4.4 USB Device Interface Driver 13
4.5 SD/MMC Controller Interface Driver 13
4.6 HSUART Interface Driver 14
4.7 SPI Interface Driver 14
4.8 I^{2}C* Interface Driver 15
4.9 GPIO Interface Driver 16
4.10 Ethernet Interface Driver (STMMAC) 17
4.11 Userspace I/O Subsystem (UIO) enabling for GPIO 18

5.0 Intel® Quark™ SoC X1000 Host Bridge Drivers 19
5.1 eSRAM Configuration Driver 19
5.2 Isolated Memory Region Driver 21
5.3 Thermal Driver 21

6.0 Legacy Block Driver 23
6.1 Legacy GPIO 23

7.0 Expansion Drivers 25
7.1 AD7298 Driver 25
7.2 Bluetooth* Driver 26
7.3 Wi-Fi* Driver 27
7.4 3G Modem Driver 29

8.0 Sample Applications 34
8.1 Generic Buffer 34
8.2 Generic Buffer High Resolution Timer 35

9.0 Secure Boot Implementation 37
9.1 Overview 37
9.2 Isolated Memory Regions 37
9.3 Bootloader Security 37
9.4 OS Security 40

10.0 Yocto Bitbake Distros Built Variations 41
10.1 High level Implementation Variations 41

11.0 Platform Loadable Driver Modules Support Variations.....43

Figures

1 Intel® Quark™ SoC X1000 Block Diagram 8
2 Software Architecture Overview.....10
3 Multiplexing using Intel® Quark™ SoC X1000 SPI Driver.....15
4 ADC Location in Software Stack.....25
5 Grub Secure Boot Flow.....39

Tables

1 Product Documentation 6
2 Terminology 7
3 Intel® Quark™ SoC X1000 Hardware Interfaces and Drivers.....12
4 Intel® Quark™ SoC GPIO Interface Pin Table 17
5 Intel® Quark™ SoC Legacy GPIO Interface Pin Table 24
6 High level Implementation Variations 41
7 Auto-load and Loadable Modules.....43
8 Platform Specific Loadable Driver Modules 43

Revision History

Date	Revision	Description
February 2016	005	Updated for 1.2.1 release: • Section 4.2, "USB OHCI Controller Interface Driver" • Section 4.4, "USB Device Interface Driver" • Section 4.6, "HSUART Interface Driver" • Section 4.10, "Ethernet Interface Driver (STMMAC)" • Section 7.4, "3G Modem Driver" • Table 6, "High level Implementation Variations" on page 43 • Section 11.0, "Platform Loadable Driver Modules Support Variations" • Table 8, "Platform Specific Loadable Driver Modules" on page 45
October 2015	004	Updated for 1.2 release: • Section 1.1, "About this Manual" • Section 1.4, "Terminology" • Section 2.2, "SoC Features" • Section 4.0, "Intel® Quark™ SoC X1000 Drivers" (introduction paragraph) • Section 4.1, "Overview" • Section 4.2, "USB OHCI Controller Interface Driver" • Section 4.3, "USB 2.0 EHCI Controller Interface Driver" • Section 4.4, "USB Device Interface Driver" • Section 4.5, "SD/MMC Controller Interface Driver" • Section 4.6, "HSUART Interface Driver" • Section 4.7, "SPI Interface Driver" • Section 4.8, "I^2C* Interface Driver" • Section 4.9, "GPIO Interface Driver" • Section 4.10, "Ethernet Interface Driver (STMMAC)" • Section 5.1, "eSRAM Configuration Driver" • Section 5.1.1.1, "Example showing eSRAM stat usage" • Section 5.1.1.2, "Example of Mapping a Virtual Address into eSRAM" • Section 5.2.1, "IMR Run-time Kernel Protection" • Section 5.3, "Thermal Driver" • Section 6.1, "Legacy GPIO" • Section 7.1, "AD7298 Driver" • Section 7.2, "Bluetooth* Driver" • Section 7.4, "3G Modem Driver" • Section 7.4.9, "Send an SMS Text Message to 0123456789" • Section 7.4.15, "Enable Data Packet Connection (PPP)" • Section 8.0, "Sample Applications" • Section 9.1, "Overview" • Section 9.2, "Isolated Memory Regions" • Section 9.4.1, "Linux* IMR setup" • Moved "Debug Interface" from Section 9.4.2 to Section 9.4.1.3, "Debug Interface" Added: • Section 4.11, "Userspace I/O Subsystem (UIO) enabling for GPIO" • Section 10.0, "Yocto Bitbake Distros Built Variations " • Section 11.0, "Platform Loadable Driver Modules Support Variations"

Date	Revision	Description
February 2015	003	Updated for 1.1 release: • Section 4.6, "HSUART Interface Driver" • Section 5.1, "eSRAM Configuration Driver" • Section 9.2, "Isolated Memory Regions" • Section 9.3.1, "Asset Verification Flow" • Section 9.3.2, "Isolated Memory Region Flow" • Section 9.4, "OS Security"
May 2014	002	Updates for software release 1.0.1 including: • Modified Section 4.6 to change driver name from "RS232+DMA" to "UART+DMA" to be more clear. See changebars for details. • Updated with trademarked term: Intel® Quark™ SoC.
March 2014	001	First public release of document.

1.0 Introduction

1.1 About this Manual

Intel® Quark™ SoC is a next generation secure, low-power Intel Architecture (IA) System on a Chip (SoC) for deeply embedded applications. The Intel® Quark™ SoC X1000 integrates the Intel® Quark™ Core plus all the required hardware components to run off-the-shelf operating systems and to leverage the vast x86 software ecosystem.

This document describes the architecture and usage of the Intel® Quark™ SoC X1000 Software for Linux* kernel 3.14 with Quark modifications.

1.2 Introduction

The Intel® Quark™ SoC X1000 Software is a set of silicon enabling software that exposes silicon features to a run-time kernel and user-space in a convenient manner. Drivers that have been extended to enable Intel® Quark™ SoC are described in terms of standard driver interfaces. Drivers that have been created to expose a particular silicon feature are detailed in terms of their specific in-kernel and/or user-space API.

Intel® Quark™ SoC has standard x86 environment enumeration with legacy block and PCI enumeration mechanisms that are highly compatible with previous silicon configurations. Where possible, commercial off-the-shelf (COTS) drivers have been used and/or modified to achieve maximum compatibility with minimum software code churn.

1.3 Related Documentation

Table 1 lists the product documentation supporting this release.

Table 1. Product Documentation

Title	Number
Intel® Quark™ SoC X1000 Datasheet [Datasheet]	329676
Intel® Quark™ SoC X1000 Secure Boot Programmer's Reference Manual	330234
Intel® Quark™ SoC X1000 Software Developer's Manual for Linux* (this document)	330235
Intel® Quark™ SoC X1000 Board Support Package (BSP) Build and Software User Guide [Build & SW User Guide]	329687
Intel® Quark™ SoC X1000 Software Release Notes	330232
Intel® Quark™ SoC X1000 UEFI Firmware Writer's Guide	330236

Standard Linux* documentation can be found at: www.kernel.org/doc/

1.4 Terminology

Table 2. Terminology

Term	Description
ADC	Analogue to Digital Converter
BSP	Board Support Package - a set of silicon enabling software which enables and enhances a run-time operating system kernel, such as Linux*.
DMA	Direct Memory Access
EDK	EFI Developer Kit
EFI	Extensible Firmware Interface
EHCI	Enhanced Host Controller Interface
eSRAM	embedded SRAM
GIP	GPIO I2C Peripheral
GPIO	General Purpose Input/Output
I^2C*	I-squared-C - a type of two wire communications bus
IMR	Isolated Memory Region
LAN	Local Area Network
MMC	Multi Media Card
OHCI	Open Host Controller Interface
PCH	Platform Control Hub
SD	Secure Digital Flash
SoC	System on Chip
SPI	Serial Peripheral Interconnect
SRAM	Static Random Access Memory
STMMAC	STMicroelectronics Media Access Controller
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VLAN	Virtual LAN

1.5 Conventions

The following conventions are used in this manual:

- `Courier` font - code examples, command line entries, API names, parameters, filenames, directory paths, and executables.
- **Bold** text - graphical user interface entries and buttons

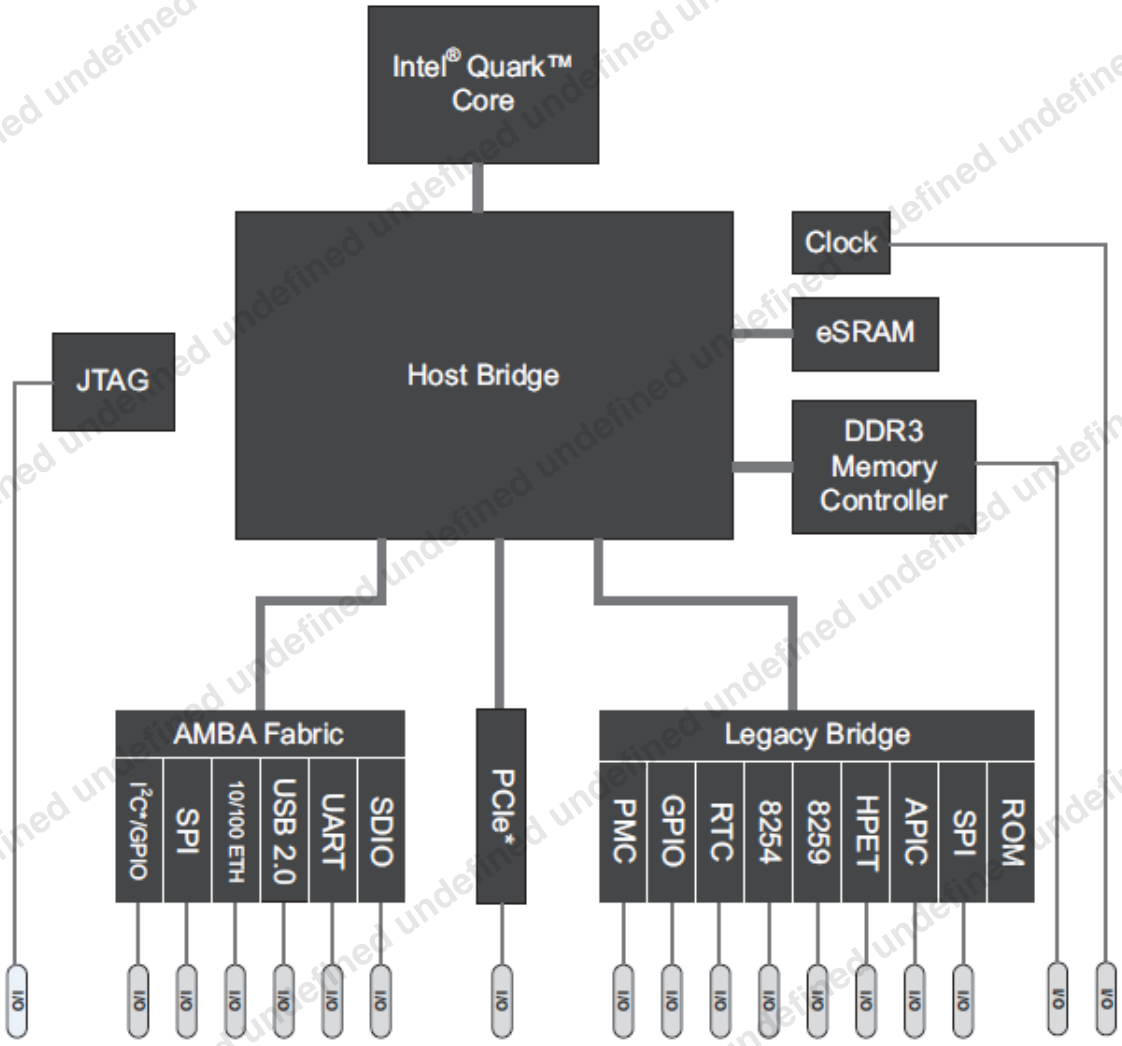
§ §

2.0 Platform Overview

2.1 Platform Synopsis

Intel® Quark™ SoC X1000 is a next generation, secure, low-power Intel Architecture System on Chip (SoC) for deeply embedded applications. As shown in Figure 1, Intel® Quark™ SoC X1000 is comprised of a Intel® Quark™ Core processor with a host bridge, PCIe expansion, a range of I/O interfaces, DDR3 controller, and an eSRAM block.

Figure 1. Intel® Quark™ SoC X1000 Block Diagram



2.2 SoC Features

The main features relevant to the Intel® Quark™ SoC X1000 Software are as follows:

- Intel® Quark™ Core
  - Intel® Pentium® compatible instruction set architecture (ISA)
  - Time stamp counter register (TSC)
  - Local APIC (LAPIC)
  - MSR compatability CPUID family = 0x5 revision = 0x09
- Host Bridge
  - 512k of fast access embedded SRAM (eSRAM)
  - 8 x memory protection regions, called Isolated Memory Regions (IMRs)
  - Thermal Sensor
- Legacy block
  - 8254 Programmable Interval Timer (PIT)
  - 2 cascaded 8259 Programmable Interrupt Controllers (PIC)
  - High Precision Event Timer (HPET)
  - IO-APIC
  - Real Time Clock (RTC)
  - GPIO x 8 - 6 in suspend well - driving NMI, SCI, or SMI
  - Legacy SPI and Boot ROM
- Intel® Quark™ SoC X1000
  - OCHI USB Host controller
  - EHCI USB Host controller
  - USB Device controller
  - 2 x 16550 UART with DMA enhancements
  - 2 x SPI Master interface
  - I2C* Master interface
  - 8 x GPIO interface (non-legacy)
  - 2 x 100 Mbit Ethernet with external PHY
  - eMMC/MMC controller interface

§ §

3.0 Software Overview

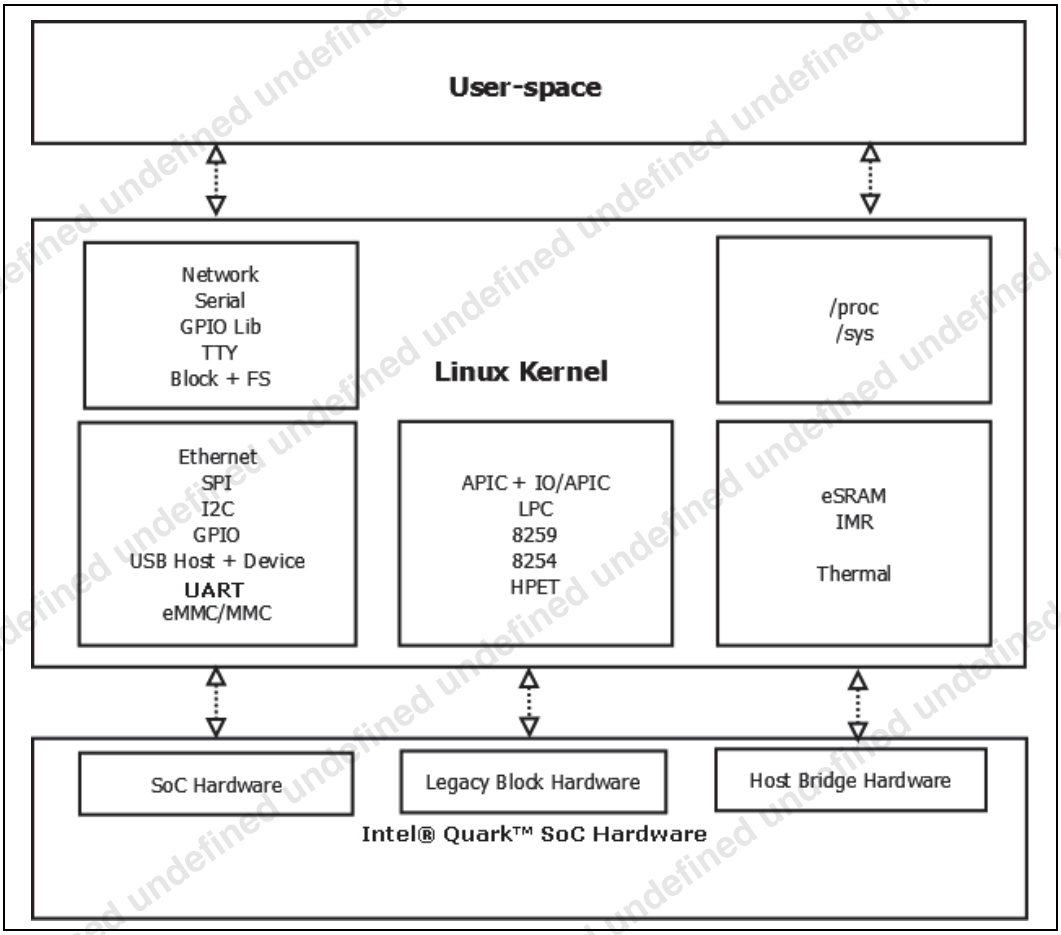
3.1 High-Level Software Architecture Overview

The Intel® Quark™ SoC X1000 uses many off-the-shelf software components to enable product features. This aim is pervasive throughout the system in terms of Intel® Quark™ Core, Host Bridge, and SoC components.

Intel® Quark™ SoC X1000 has two key categories of software deliverables:

- Extensions to existing Linux* device drivers to enable the Intel® Quark™ SoC X1000
- Creation of entirely new drivers for Host Bridge-related functions

Figure 2. Software Architecture Overview



3.2 Linux* Support

3.2.1 Standard OS Drivers

The software delivery supports Linux. Many of the I/O drivers, including USB, Ethernet, UART, I^2C, and SPI, are derived from existing upstream kernel components. (The I^2C/GPIO driver was created for Intel® Quark™ SoC X1000.) Driver modifications maintain compatibility with existing software while enabling Intel® Quark™ SoC X1000 specific features.

See Table 3, "Intel® Quark™ SoC X1000 Hardware Interfaces and Drivers" on page 13 for details.

3.2.2 Host Bridge OS Drivers

Host Bridge silicon enabling software is specific to the Intel® Quark™ SoC X1000 and as such has no formal operating system interface that exactly matches the conceptual paradigms. For this reason, Intel® Quark™ SoC X1000 specific APIs and user-space interfaces via `sysfs` and `proc` have been developed for the IMR and eSRAM interface.

Details on the interfaces for IMR and eSRAM configuration are provided later in this document.

3.2.3 Bootloader Host Bridge Drivers

In order to facilitate secure boot, the reference bootloader `grub` v 0.97 with EFI extensions has been modified to support setup and teardown of IMRs as appropriate to transition from UEFI to run-time OS. Section 9.0, "Secure Boot Implementation" on page 38 describes this flow.

3.3 User-Space Software Dependencies

To facilitate exposure of silicon features, the user-space component of the runtime reference OS requires the following utilities:

- `ethtool` - customized version of ethtool updated to include registers exported by the Intel® Quark™ SoC X1000
- `ptpd` - Precision Time Protocol Daemon

These utilities are included with the Intel® Quark™ SoC X1000 yocto layer.

§ §

4.0 Intel® Quark™ SoC X1000 Drivers

System on a Chip in the context of Intel® Quark™ SoC X1000 refers to peripheral hardware south of the host bridge interface. SoC software drivers bind the hardware interfaces into standard Linux* sub-systems. Linux* kernel baseline of 3.14 (or higher) is required to ensure proper integration and compatibility of upstream reused kernel drivers.

4.1 Overview

Table 3 lists the hardware interfaces implemented on Intel® Quark™ SoC X1000 and identifies whether the associated driver is one of the following:

- standard (unmodified) off-the-shelf driver
- modified version of off-the-shelf driver, enhanced to enable Intel® Quark™ SoC X1000 specific features

  Note: Refer to the software sources to determine the complete list of modified or added files as compared to the Linux* kernel baseline 3.14.
- created to be Intel® Quark™ SoC X1000 specific

Table 3. Intel® Quark™ SoC X1000 Hardware Interfaces and Drivers

Hardware Interface	Standard Linux* Driver	Modified Linux* Driver	Intel® Quark™ SoC X1000 Specific Driver
USB OHCI Controller Interface	X		
USB 2.0 EHCI Controller Interface	X		
USB Device Interface		X	
SD/MMC Controller Interface		X	
UART + DMA Interface		X	
SPI Master Interface		X	
I^2C/GPIO Interface			X
Intel Legacy Block GPIO		X	
IMRs		X	
Ethernet Interface		X	

4.2 USB OHCI Controller Interface Driver

The standard Linux* OHCI driver is compatible with Intel® Quark™ SoC X1000. This driver provides full USB host control and arbitration of the USB in OHCI mode.

To load this driver in Linux* as root, type:

```
modprobe ohci_pci
```

Once loaded, the OHCI driver provides access to USB 1.1 devices through either of the USB host ports, thus enabling host controller interface with full speed and low speed USB devices.

4.3 USB 2.0 EHCI Controller Interface Driver

The standard Linux* EHCI driver is compatible with Intel® Quark™ SoC X1000. This driver has a prerequisite for the OHCI to be loaded before the EHCI driver is loaded. Once loaded, the EHCI driver provides full host control and arbitration of the USB in EHCI mode.

To load this driver in Linux* as root, type:

```
modprobe ehci_pci
modprobe ohci_hcd
```

Once loaded, the EHCI driver provides access to High speed USB devices through either of the Intel® Quark™ SoC X1000 host controller ports.

4.4 USB Device Interface Driver

The standard PCH UDC driver (with the addition of Intel® Quark™ SoC X1000 PCI vendor/device identifiers and quirks) is compatible with Intel® Quark™ SoC X1000.

Using the reference driver released in the software package, type:

```
modprobe pch_udc
```

This loads the hardware driver.

To have the Intel® Quark™ SoC X1000 appear as a USB mass storage device to the USB host machine, and assuming a suitable file exists at `/media/mmc1/floppy.img`, type:

```
modprobe g_mass_storage file=/media/mmc1/floppy.img
```

To have the Intel® Quark™ SoC X1000 appear as a serial device to the USB host machine, type:

```
modprobe g_serial
```

The USB device hardware does not support alternative settings. Therefore the system will not support gadget drivers that use this functionality. For example g-ether.

The X1000 contains only 3 endpoints. It will therefore can not support gadget drivers that require more endpoints than this. For example g-multi.

4.5 SD/MMC Controller Interface Driver

The standard Linux* MMC/SD driver (with the addition of Intel® Quark™ SoC X1000 quirks) is compatible with Intel® Quark™ SoC X1000. Once loaded, an MMC or SD storage device appears as a standard Linux* block interface, upon which a file system can be formatted and mounted.

This example loads the SDHCI PCI driver and MMC block device driver:

```
modprobe sdhci-pci
modprobe mmc_block
```

Once loaded, assuming the MMC card is partitioned and formatted, device entries appear in `/dev` representing the partitions found on the MMC device.



4.6 HSUART Interface Driver

Note: In the Intel® Quark™ SoC X1000 Datasheet, this is referred to as the high speed UART.

The standard upstream 16550 PCI UART will work with Intel® Quark™ SoC X1000, with the addition of the relevant PCI vendor/device strings. The Intel® Quark™ SoC X1000 UART interface is 100% compatible with the standard 16550 register interface, however, the standard driver does not support DMA.

The FIFO depth is 16 bytes and hardware flow control is included. The Intel® Quark™ SoC X1000 has two UARTs.

Note: There is no support supplied for legacy I/O port access at addresses 0x3F8, 0x2F8, 0x3E8 or 0x2E8.

Inside the PCI configuration space of each UART, a second PCI BAR exists pointing to the DMA resource range.

A SoC-specific driver called `intel_quark_hsuart_dma` is provided to enable DMA operation. This driver is a thin glue layer that binds the upstream 16550 driver with the upstream driver for the UART's DMA controller.

This driver registers:

```
/dev/ttyS0
/dev/ttyS1
```

Note:

DMA operation is enabled by default. To disable DMA on a UART instance, add the following kernel parameter:

- ttyS0 ==> intel_quark_hsuart_dma.uart0_dma = 0
- ttyS1 ==> intel_quark_hsuart_dma.uart1_dma = 0

Note: When DMA mode is enabled, the UART does not support software flow control.

4.7 SPI Interface Driver

The Intel® Quark™ SoC X1000 SPI interface exports a standard SPI interface from kernel-space to user-space. Two SPI master interfaces are available on Intel® Quark™ SoC X1000. To increase the number of devices that Intel® Quark™ SoC X1000 can communicate with simultaneously, GPIOs are used to achieve *multiplexing* (also called *muxing*) of the SPI master interface.

This muxing approach allows Intel® Quark™ SoC X1000 to communicate with up to four SPI slave interfaces, with a maximum of two slave devices at any one time as shown in Figure 3.

To load Intel® Quark™ SoC X1000 SPI driver, type:

```
modprobe spi-pxa2xx-platform
modprobe spi-pxa2xx-pci
modprobe spidev
```

GPIO pin selection is achieved by providing board-specific data in the file:

```
drivers/platform/x86/intel-quark/<boardname>.c
```

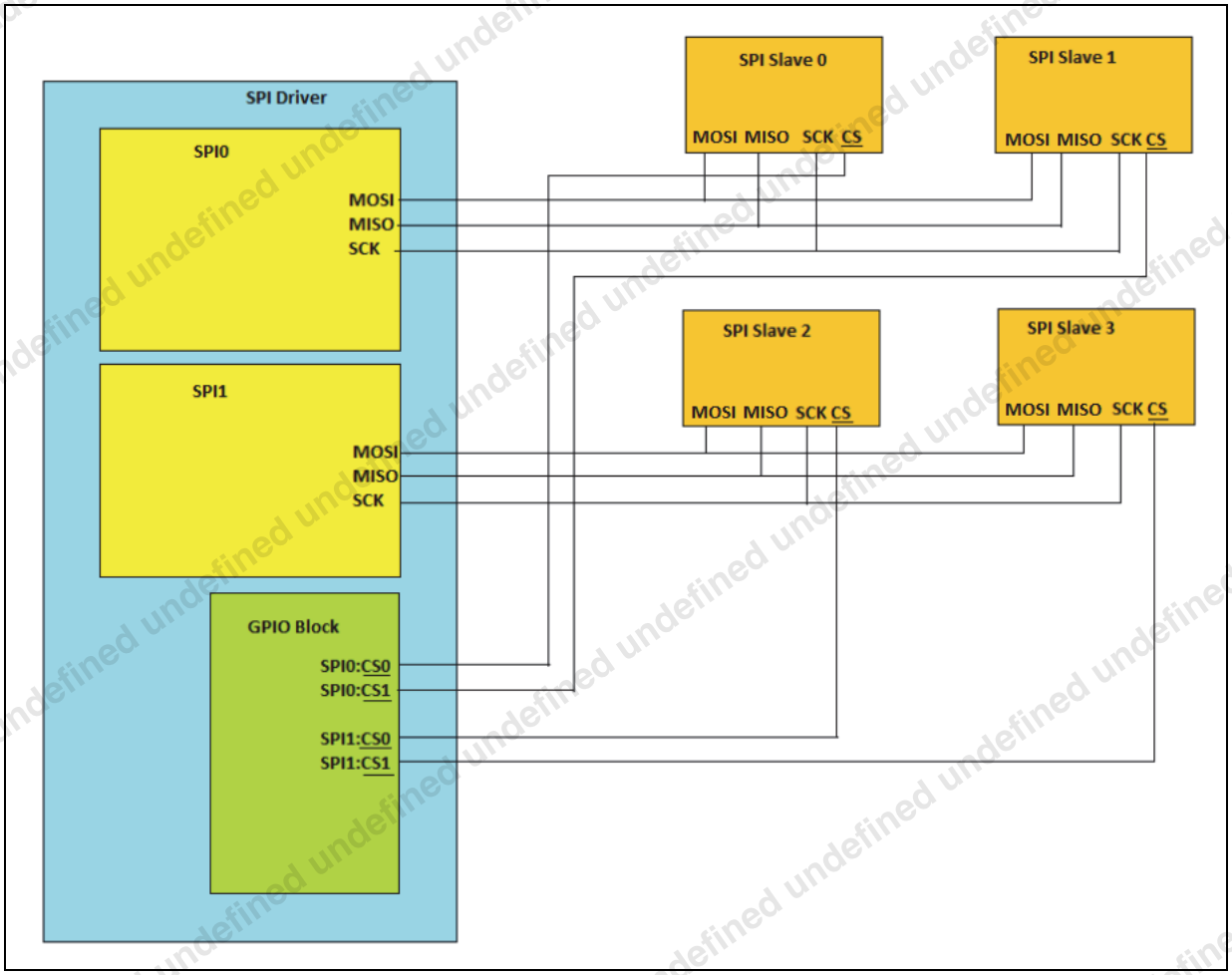
Once loaded, the master SPI driver populates entries in `/dev` as follows:

```
/dev/spidev0.0
/dev/spidev0.1
/dev/spidev1.0
/dev/spidev1.1
```

The format is `/dev/spidevX.Y` where:

- `X` indicates the master interface
- `Y` indicates the slave interface

Figure 3. Multiplexing using Intel® Quark™ SoC X1000 SPI Driver



4.8 I²C* Interface Driver

The I²C and GPIO components are contained within the same PCI function and share resources as a consequence. The I²C register interface is 100% compatible with the upstream `i2c-designware-core` driver.



This register interface is incorporated in the `intel_qrk_gip` driver, which provides a standard I^2C interface when loaded. The GIP interface can be loaded in either MSI or non-MSI mode using the commands:

```
modprobe intel_qrk_gip
```

or

```
modprobe intel_qrk_gip enable_msi=0
```

In either case, loading this driver and using the command `modprobe i2c-dev` populates:

```
/dev/i2c-0
```

Once populated, it is possible to communicate with downstream I^2C devices using the standard Linux* API to interact with the I^2C bus.

To load the I^2C driver in isolation (that is, without the GPIO enabling logic contained in the GIP block), type:

```
modprobe intel_qrk_gip gpio=0
```

or

```
modprobe intel_qrk_gip gpio=0 enable_msi=0
```

The GIP block defaults the I^2C devices to fast mode operation (400 kHz). To set to the standard mode operation (100 kHz), type:

```
modprobe intel_qrk_gip i2c_std_mode=1
```

Alternatively, you can set the i2c_std_mode parameter to "1" in the platform data (intel_qrk_gip_get_pdata) section of the Intel® Quark™ SoC X1000 Software.

4.9 GPIO Interface Driver

Note: This driver is different from the one described in Section 6.1, "Legacy GPIO" on page 24.

The GPIO and I^2C components are contained within the same PCI function and share resources as a consequence. This GPIO interface is a new register interface and is enabled by the GPIO section of the `intel_qrk_gip` device driver module.

In the Intel® Quark™ SoC X1000 Datasheet, these pins are referred to as GPIO[7:0]. These GPIO pins are interrupt-capable. They support rising/falling edge-triggered interrupts (but not both edged interrupts) and high/low level-triggered interrupts.

To load the GPIO driver in isolation (that is, without the I^2C enabling logic contained in the GIP block) type:

```
modprobe intel_qrk_gip i2c=0
```

or

```
modprobe intel_qrk_gip i2c=0 enable_msi=0
```

Note: MSIs are enabled by default. Disabling MSIs is not recommended for performance reasons.

Table 4. Intel® Quark™ SoC GPIO Interface Pin Table

Linux GPIO	Well	Pin Name
gpio8	Core	GPIO[0]
gpio9	Core	GPIO[1]
gpio10	Core	GPIO[2]
gpio11	Core	GPIO[3]
gpio12	Core	GPIO[4]
gpio13	Core	GPIO[5]
gpio14	Core	GPIO[6]
gpio15	Core	GPIO[7]

4.10 Ethernet Interface Driver (STMMAC)

The STMMAC driver upstream in the Linux* kernel is nearly entirely compatible with Intel® Quark™ SoC X1000, with some minor updates to the DMA component of the STMMAC driver. This update to STMMAC is based on modification of the upstream driver.

In addition to the necessary DMA enumerating descriptors in STMMAC, additional Intel® Quark™ SoC X1000 specific silicon-enabling enhancements have been made to the standard STMMAC. The enhancements include:

- VLAN
  - — Hardware filtering has been added
  - — Maximum number of hardware filtered VLAN tags is 16
  - — Tag ID range 0 - 15

The following commands demonstrate how to load the STMMAC in either MSI or non-MSI mode.

```
modprobe stmmac_pci
```

or

```
modprobe stmmac_pci enable_msi = 0
```

Note: MSI mode is enabled by default.

4.10.1 VLAN

The standard Linux* commands `ip` or `vconfig` can be used to add or remove hardware accelerated VLAN tag filtering entries in STMMAC.

The following commands demonstrate how to add VLAN # 5:

```
vconfig add enp0s20f6 5
ifconfig enp0s20f6.5 xxx.yyy.zzz.qqq
```

Once setup is complete, VLAN frames with tag ID 5 are processed by Intel® Quark™ SoC X1000 while other Ethernet frames with different tags are not processed by hardware and do not raise interrupts to the core.

To remove a hardware filtered VLAN interface, enter the command:

```
vconfig rem enp0s20f6.5
```

4.11 Userspace I/O Subsystem (UIO) enabling for GPIO

A UIO interface is added to the Intel Quark SoC GPIO Interface Driver and Intel Quark SoC Legacy GPIO Driver. This allows the user to bypass the default `gpiolib sysfs` methods.

To enable this interface, make the following kernel configuration changes:

```
CONFIG_INTEL_QRK_GIP = m

CONFIG_GPIO_SCH = m

CONFIG_INTEL_QRK_GPIO_UIO = y
```

Note: The UIO interface is non-coherent with the default `gpiolib sysfs` methods. The UIO interface is not able to make atomic changes to the GPIO registers but the default `gpiolib sysfs` methods are able to. Changes made by the default `gpiolib sysfs` methods would be overwritten by the UIO interface in a race condition.

Note: The current driver architecture requires the Intel Quark SoC GPIO Interface Driver and Intel Quark SoC Legacy GPIO Driver to be built as modules to enable the UIO interface.

4.12 Enable TPM in Yocto OS

The following changes are required to enable TPM module for CrossHill secure platform.

Enable the following options in kernel configuration file:

```
CONFIG_HW_RANDOM_TPM=m
CONFIG_TCG_TPM=m
CONFIG_TCG_TIS_I2C_INFINEON=m
CONFIG_TCG_INFINEON=m
```

Enable TPM support in Yocto recipe to load the TPM modules during boot.

In recipe: meta-intel-quark/recipes-kernel/linux/linux-yocto-quark_3.14.bb

Add the following lines at the end of the file:

```
# TPM
KERNEL_MODULE_AUTOLOAD_append_quark = " tpm"
KERNEL_MODULE_AUTOLOAD_append_quark = " tpm_i2c_infineon"

run bitbake image-full to generate the images.
```

§ §

5.0 Intel® Quark™ SoC X1000 Host Bridge Drivers

Host Bridge Drivers in the context of Intel® Quark™ SoC X1000 refer to drivers for silicon functionality that are part of the Host Bridge interface on Intel® Quark™ SoC X1000. This functionality is exposed via a *side-band* driver that arbitrates access to the various components using the Host Bridge interface.

The side-band driver provides access to the following blocks of functionality:

- eSRAM
- Isolated Memory Regions
- Thermal

5.1 eSRAM Configuration Driver

Intel® Quark™ SoC X1000 contains a set of embedded SRAM (eSRAM). There is 512 kilobytes of eSRAM sub-divided into 128 pages of four kilobytes each. eSRAM can be configured in a per-page manner, and eSRAM can exist in an overlay of memory in the address space.

eSRAM is a fast access low-latency memory that has been measured on Intel® Quark™ SoC X1000 to be approximately 3x faster than DDR, in terms of CPU wait-states and access times.

For Linux* enabling purposes, eSRAM has been configured in a per-page overlay mode. This approach allows overlay of specific regions of memory. For example, the interrupt descriptor table or arbitrary interrupt service routines (ISRs) can be locked into eSRAM.

Kernel virtual addresses can be can be mapped into eSRAM. The minimum granularity for any map operation is 4 kilobytes, hence any other data within the same 4 kilobyte address range is also mapped.

Note: Unmapping is neither supported nor advised due to potential coherency issues when flushing eSRAM back to DRAM.

Warning: Due to the eSRAM hardware architecture, there is a time window during the page overlaying process whereby the DRAM page itself is not accessible. As a consequence, users should avoid overlaying pages containing symbols used by the driver's page-populate routine. A fully comprehensive list may be obtained by analyzing the subroutine call graph of `intel_qrk_esram_page_populate_atomic()`.

5.1.1 Userspace API Reference

A `sysfs` interface has been provided to configure eSRAM mappings.

- `/sys/devices/platform/intel-qrk-esram.0/map_range`
  - — Allows overlaying of a kernel page given its virtual address
  - — Allows viewing of all current overlaid pages
- `/sys/devices/platform/intel-qrk-esram.0/stats`



— Gives a status overview of current eSRAM state
— Number of free pages
— Other miscellaneous data

Note: At the time of writing, the `map_range` interface only allows overlaying one page at a time.

5.1.1.1 Example showing eSRAM stat usage

```
root@quark:~# cat /sys/devices/platform/intel-qrk-esram.0/stats
esram-pgpool                    : 0x19fcfe00
esram-pgpool.free               : 128
esram-pgpool.flushing           : 128
esram-ctrl                      : 0x047f3f91
esram-ctrl.ecc                  : enabled
esram-ctrl.ecc-theshold         : 63
esram-ctrl.pages                : 128
esram-ctrl.dram-flush-priorityi : 2
esram-block                     : 0x00000000
free page                       : 128
used page                       : 0
refresh                         : 0ms
page enable retries             : 0
page disable retries    : 0
ecc next page                   : 127
```

5.1.1.2 Example of Mapping a Virtual Address into eSRAM

The following examples shows how to overlay the page where printk is defined into eSRAM.

```
root@quark:~# cat /proc/kallsyms |grep " printk$"
c1341299 T printk
[ Inspect page @ 0xc1341000 to ensure it is safe to overlay ]
root@quark:~# echo 0xc1341000 > /sys/devices/platform/intel-qrk-esram.0/
map_range
root@quark:~# cat /sys/devices/platform/intel-qrk-esram.0/map_range
sysfs
        Page virt 0xc1341000 phys 0x01341000
        Refcount 1
```

5.1.2 Kernel API Reference

An API to map kernel address ranges is available.

5.1.2.1 intel_qrk_esram_map_range

Map 4kB increments at given address to eSRAM. Maps any arbitrary kernel virtual address from vaddr to vaddr + size bytes. This mapping is then named `mapname`.

```
int intel_qrk_esram_map_range(void * vaddr, u32 size, char * mapname);
```

- vaddr: Virtual address to start mapping (must be 4k aligned)
- size: Size to map from - aligned to a 4 kilobyte boundary

- mapname: Mapping symbolic name shown in sysfs
- return 0 success < 0 failure

5.2 Isolated Memory Region Driver

Isolated Memory Region (IMR) allocation and assignments are detailed in the Intel® Quark™ SoC X1000 Secure Boot Programmer's Reference Manual. In Linux* a run-time interface provides an convenient method to view IMR allocations.

This interface shows the IMR allocations provided as part of the secure boot reference code on the Intel® Quark™ SoC X1000.

5.2.1 IMR Run-time Kernel Protection

```
root@quark:~# cat /sys/kernel/debug/imr_state

imr00: base=0x0000000000000000, end=0x0000000000000000, size=0x00000000
rmask=0xbfffffff, wmask=0xffffffff, disabled, locked

imr01: base=0x0000000000000000, end=0x0000000000000000, size=0x00000000
rmask=0xbfffffff, wmask=0xffffffff, disabled, locked

imr02: base=0x0000000000000000, end=0x0000000000000000, size=0x00000000
rmask=0xbfffffff, wmask=0xffffffff, disabled, locked

imr03: base=0x0000000000000000, end=0x0000000000000000, size=0x00000000
rmask=0xbfffffff, wmask=0xffffffff, disabled, locked

imr04: base=0x0000000000000000, end=0x0000000000000000, size=0x00000000
rmask=0xbfffffff, wmask=0xffffffff, disabled, locked

imr05: base=0x0000000000000000, end=0x0000000000000000, size=0x00000000
rmask=0xbfffffff, wmask=0xffffffff, disabled, locked

imr06: base=0x0000000000000000, end=0x0000000000000000, size=0x00000000
rmask=0xbfffffff, wmask=0xffffffff, disabled, locked

imr07: base=0x0000000001000000, end=0x00000000014e1fff, size=0x004e1fff
rmask=0x80000001, wmask=0xc0000001, enabled, locked
```

5.3 Thermal Driver

Linux* provides a standard thermal driver interface. Intel® Quark™ SoC X1000 hooks its particular thermal silicon into this Linux* sub-system. The thermal driver is minimalistic in design with no associated thermal cooling device attached to the one and only thermal zone, thus the Intel® Quark™ SoC X1000 may require external cooling.

Intel® Quark™ SoC X1000 hardware is set up to automatically shutdown on critical temperature detection. The trip points described below are set in the driver and cannot be changed.

Linux* provides an entire sub-system dedicated to triggering events based on hot and critical events. The task of the thermal driver is to provide the minimum level of silicon support to drive these events.

- Hot trip point: 95 degrees Celsius



The thermal driver incrementally polls the thermal sensor and when this threshold is exceeded, a hot trip event is propagated into the thermal sub-system.

- Critical trip point: 104 degrees Celsius
  The Linux* thermal sub-system triggers a graceful system shutdown if the critical trip threshold is reached.
- Hardware failover critical temperature: 105 degrees Celsius
  As a precautionary measure, Intel® Quark™ SoC X1000 silicon is configured to drive a shutdown signal at 105 degrees Celsius. Assumption is that software polling should catch an over-temperature situation when temperature meets or exceeds the critical trip point (104 degrees Celsius). A one degree over-limit from the maximum specified critical temperature forces embedded hardware to take preventative action and drive a shutdown signal directly.

§ §

6.0 Legacy Block Driver

The LPC address space contained within Intel® Quark™ SoC X1000 legacy block has the following component that has been enabled in the Linux* run-time:

- Legacy GPIO

In order to enable this silicon functionality, a small modification is necessary to LPC enabling software in Linux, adding appropriate PCI vendor/device.

6.1 Legacy GPIO

Note: This driver is different than the one described in Section 4.9, "GPIO Interface Driver" on page 17.

Intel® Quark™ SoC X1000 contains eight GPIOs within the legacy bridge. These GPIO pins are interrupt-capable. They support rising/falling/both edge-triggered interrupts.

These legacy GPIOs provide the ability to drive GPE events and hence to resume an Intel® Quark™ SoC X1000 device in a low-power state.

There are:

- 6 GPIO pins in the resume power well
  In the Intel® Quark™ X1000 Datasheet, these pins are referred to as GPIO_SUS[5:0].
  The GPIOs in the resume well can be used to drive a General Purpose Event (GPE) through the ACPI sub-system that subsequently takes the Intel® Quark™ SoC X1000 out of a low-power state.
- 2 GPIO pins in the core well
  In the Intel® Quark™ X1000 Datasheet, these pins are referred to as GPIO[9:8].

The eight legacy GPIO are indexed in the range [0,7] and can be accessed from user-space through `sysfs` interface.

To load the legacy GPIO driver, type:

```
modprobe gpio-sch
```

The commands below demonstrate how to drive a signal to the first legacy GPIO:

```
root@quark# echo 0 > /sys/class/gpio/export # Reserve first legacy GPIO
root@quark# echo "out" > /sys/class/gpio/gpio0/direction # Set as output
root@quark# echo "1" > /sys/class/gpio/gpio0/value # Drive logical one
```



Table 5. Intel® Quark™ SoC Legacy GPIO Interface Pin Table

Linux GPIO	Well	Pin Name
gpio0	Core	GPIO[8]
gpio1	Core	GPIO[9]
gpio2	Suspend	GPIO_SUS[0]
gpio3	Suspend	GPIO_SUS[1]
gpio4	Suspend	GPIO_SUS[2]
gpio5	Suspend	GPIO_SUS[3]
gpio6	Suspend	GPIO_SUS[4]
gpio7	Suspend	GPIO_SUS[5]

§ §

7.0 Expansion Drivers

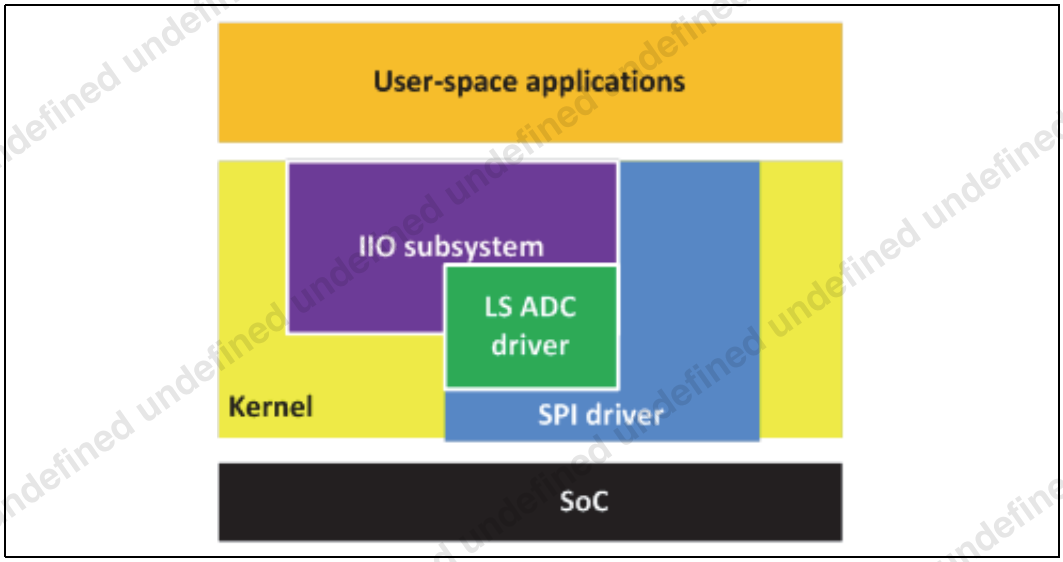
This section describes drivers that are included with the Intel® Quark™ SoC X1000 Software package to enable board-specific functionality.

- AD7298 Driver
- Bluetooth* Driver (requires mini-PCIe card)
- Wi-Fi* Driver (requires mini-PCIe card)
- 3G Modem Driver (requires mini-PCIe card)

7.1 AD7298 Driver

The Analog Devices* AD7298 is a 12-bit, low power, 8-channel, successive approximation ADC with an internal temperature sensor. The LS-ADC does not provide a user-space interface directly, it is provided by the IIO subsystem in the Linux* kernel. The ADC registers with the IIO subsystem as an IIO ADC device driver. As such, it makes calls to functions on the IIO kernel API and provides callbacks which can be used by the IIO subsystem to invoke driver operations.

Figure 4. ADC Location in Software Stack



To load the drivers for the AD7298, perform the following sequence:

- Enable GPIO driver:
  ```
  modprobe intel_qrk_gip
  ```
- Enable IIO support:
  ```
  modprobe industrialio
  ```



- Enable SPI driver:
  `modprobe spi-pxa2xx-pci`
- Enable AD7298 driver:
  `modprobe ad7298`

After the driver loading sequence is complete, the AD7298 driver enables the following data points via the Industrial I/O (IIO) kernel API directly read from the ADC chip.

- Provide the RAW voltage at the input in the range 0 - 4095 representing the voltage range 0 to +5 Volts
  `/sys/bus/iio/devices/iio:device0/in_voltage[0-7]_raw`
  `/sys/bus/iio/devices/iio:device0/in_voltage0_raw`
  `/sys/bus/iio/devices/iio:device0/in_voltage1_raw`
  `etc`
- Scaling value to apply to the raw voltage input
  `/sys/bus/iio/devices/iio:device0/in_voltage_scale`
- Temperature offset
  `/sys/bus/iio/devices/iio:device0/in_temp0_offset`
- Raw instantaneous temperature of the ADC die
  `/sys/bus/iio/devices/iio:device0/in_temp0_raw`
- Temperature scaling factor
  `/sys/bus/iio/devices/iio:device0/in_temp0_scale`

Other data points are provided by the Linux* IIO API but are out of scope for this document.

Using the above values, it is possible to calculate the real instantaneous voltage in milli-Volts at a given voltage input using the following formula:

(Raw value * scale value) / 1000 = V_{in0} actual input voltage in mV

Using the above values, it is possible to calculate the internal die temperature on the AD7298, in milli-degrees Celsius using the following formula:

((in_temp0_offset + in_temp0_raw) * in_temp0_scale) = T_{die}

7.2 Bluetooth* Driver

Bluetooth functionality is provided by a mini-PCIe card connected to the mini-PCIe slot on the platform. The following cards have been validated with the Intel® Quark™ SoC X1000 Software:

- Intel® Centrino® Wireless-N 135 card
- Intel® Centrino® Dual Band Wireless-N 7260 (Dual Band Wi-Fi, 2.4 and 5 GHz)

A requirement exists to include the firmware for the card in the root filesystem at the following path:

`/lib/firmware/iwlwifi-135-6.ucode` (Intel® Centrino® Wireless-N 135)

or

`/lib/firmware/iwlwifi-7260-9.ucode` (Intel® Centrino® Dual Band Wireless-N 7260)

The following drivers must be loaded to enable USB-bluetooth components:

```
modprobe ehci-pci
modprobe ohci-hcd
modprobe btusb
```

Once loaded, the sysfs entry below should appear:

```
/sys/module/bluetooth
```

The following user-space components are required:

```
bluetoothd
hciconfig
hcitool
```

7.2.1 Device Discovery

```
hciconfig <BT DEVICE NAME> noscan
hciconfig <BT DEVICE NAME>
    Expected UP_RUNNING
hcitool scan --flush
hciconfig <BT DEVICE NAME> piscan
```

7.2.2 Service Discovery

```
sdptool browse <BT_2_BD_ADDR>
```

7.2.3 Establish Connection

```
hcitool dc <BT_ADDR>
hcitool cc <BT_ADDR>
hcitool con
hcitool dc <BT_ADDR>
```

7.2.4 Ping

```
l2ping -c 5 <BT_ADDR>
```

7.3 Wi-Fi* Driver

Wi-Fi functionality is provided by a mini-PCIe card connected to the mini-PCIe slot. The Intel® Centrino® Advanced-N 6205 Wi-Fi Radio Module (Dual Band Wi-Fi, 2.4 and 5 GHz) has been validated with the Intel® Quark™ SoC X1000 Software.

A requirement exists to include the firmware for the Intel® Centrino® Advanced-N 6205 Wi-Fi Radio Module in the root filesystem at the following path:

```
/lib/firmware/iwlwifi-6000g2a-6.ucode
```

Latest firmware for this card can be downloaded from:

http://wireless.kernel.org/en/users/Drivers/iwlwifi/?n=downloads#Firmware

To load a driver for the Intel® Centrino® Advanced-N 6205 Wi-Fi Radio Module, type the following command:

```
modprobe iwlwifi
```

After a successful load of this driver, the following sysfs path is available:

```
/sys/class/net/wlan0
```

7.3.1 Enable/Disable WLAN Radio

- Get the index of the device
  `rfkill list`
- Disable radio
  `rfkill block 0`
- Enable radio
  `rfkill unblock 0`

7.3.2 Scan for Wi-Fi Networks

```
wlist wlan0 scan
```

7.3.3 Configure a Wi-Fi Device

Enter the command:

```
edit  /etc/network/interfaces
```

Add the following:

```
auto wlan0
iface wlan0 inet static
    address <IP ADDRESS>
    netmask <NETMASK>
    wireless_mode managed

    wireless_essid <SSID_NAME>
    wpa-driver wext
    wpa-conf /etc/wpa_supplicant.conf
```

7.3.4 Generate wpa_supplicant File

This file is used to configure a protected Wi-Fi network.

Generate the WPA Passphrase:

```
wpa_passphrase essid <PassPhrase>
```

Generate the `wpa_supplicant.conf` file:

```
network={
    ssid="essid"
    #psk=<PassPhrase>
    psk=<Result from last command>
}
```

7.3.5 Connect to a Wi-Fi Network

```
ifup wlan0
```

7.3.6 Disconnect from a Wi-Fi Network

```
ifdown wlan0
```

7.4 3G Modem Driver

GSM/3G communications functionality can be provided by a mini-PCIe card connected to the mini-PCIe slot. The Telit* HE910 mini-PCIe module (specifically, the functionality for GSM Voice and SMS communications, and HSPA+ data communications) has been validated with the Intel® Quark™ SoC X1000 Software.

Driver Requirements:

- Telit* HE910 requires USB2.0 support in kernel
- Telit* HE910 requires PPP (point-to-point protocol) support in kernel
- Use of active GPS antenna needs external circuit for powering antenna's amplifier

Software tool requirements:

- `minicom` - for running scripts
  Can be compiled as ipk package
- `microcom` - handy for executing simple AT commands
  Microcom is a part of busybox package.
  If it is not installed, it can be enabled in yocto using the command:
  `bitbake busybox -c menuconfig`
  then re-installed as ipk package.
- pppd - Point-to-point protocol
  ppp is used for data packet connection. It can be enabled in yocto as an image feature "ppp"

To load the drivers, perform the following sequence:

- Enable USB controllers:
  ```
  modprobe ehci-pci
  modprobe ohci-hcd
  ```
- Enable Communication Device Class Abstract Control Model interface:
  ```
  modprobe cdc-acm
  ```

References

For information on the following documents, refer to the *SUPPORT* tab at this link: www.telit.com

- HE910/UE910 AT Commands Reference Guide
- GPS Application Note
- DVI Application Note - I2S communication with Maxim 9867 codec
- Hardware guide
- Minicom manual

7.4.1 Verify System Installation and Configuration

```
dmesg | grep ttyACM
/dev/ttyACM<X>
- list of port devices created by cdc-acm driver
```

The serial port used for communicating with the 3G modem is `/dev/ttyACM0`

7.4.2 Send an AT Command to HE910 with Microcom

```
echo -ne "ATE1\r" | microcom -X -t 500 /dev/ttyACM0
```

7.4.3 Use Minicom

Starting minicom:

```
minicom -D /dev/ttyACM0
```

AT commands can be sent to the modem from minicom's console by typing.

For HE910 AT commands reference guide, see: References [•]

For detailed minicom guide, see: References [•]

7.4.4 Request Model Identification

```
AT+GMM
```

Expected:

```
HE910
OK
```

7.4.5 Request Modem Capabilities

```
AT+GCAP
```

Expected:

```
+GCAP: +CGSM,+DS,+FCLASS,+MS,+ES
OK
```

7.4.6 Check Radio Access Network Registration

```
AT+CREG?
```

Expected sample:

```
0,1
- registered to home network
```

Note: Result may vary, depending on condition. For details / see: references [•]

7.4.7 Check Signal Strength

```
AT+CSQ
```

Expected sample:

```
+CSQ: 11,2
OK
```

7.4.8 List all Available Networks

```
AT+COPS=?
```

Expected sample:

```
+COPS: (2,"Vodafone IRL",,"27201",2),(2,"Vodafone IRL",,"27201",0),
(3,"O2 - IRL",,"27202",2),(3,"IRL 05",,"27205",2),
(3,"IRL-METEOR",,"27203",2),(3,"O2 - IRL",,"27202",0),
(3,"IRL-METEOR",,"27203",0),,(0-4),(0,2)
```

7.4.9 Send an SMS Text Message to 0123456789

Set the message format 1=Text

```
AT+CMGF=1
```

Expected:

```
OK
```

Start sending the text message, specifying the number to send to.

```
AT+CMGS="0871234567"
```

The modem returns a > prompt. Type the message and press **Ctrl-z**.

```
> Hello World
```

After the **Ctrl-z**, the modem pauses for a few seconds and the following response is returned:

```
+CMGS: <n>
OK
```

7.4.10 Receive an SMS Text Message

Set the message format 1=Text

```
AT+CMGF=1
```

Expected:

```
OK
```

Select SIM card memory as SMS storage

```
AT+CPMS="SM"
```

Expected:

```
OK
```

After entering the following command, all messages are printed:

```
AT+CMGL="ALL"
```



7.4.11 Place a Call to 0871234567

Switch to voice mode

```
AT+FCLASS=8
```

Expected:

```
OK
```

Dial the number

```
ATD0871234567
```

Expected:

```
OK
```

7.4.12 Receive a Call

Switch to voice mode

```
AT+FCLASS=8
```

Once modem is called

```
RING
```

Message is printed on console.

Call can be answered with following command

```
ATS0=1
```

7.4.13 Hang Up

```
AT+CHUP
```

Expected:

```
OK
```

7.4.14 Configure Data Packet Connection (PPP)

There are many PPP configuration guides available in the internet.

Configuration may vary depending on service provider.

Example guide:

https://wiki.archlinux.org/index.php/3G_and_GPRS_modems_with_pppd

7.4.15 Enable Data Packet Connection (PPP)

Once ppp is configured, ppp connection can be established with the command:

```
pon
```

Connection can be tested with:

```
ping www.intel.com
```

Release the connection with:

```
poff
```

7.4.16 Obtain GPS Location

Make sure that GPS antenna is connected to the Telit* HE910 mini-PCIe module.

Initialize GPS module:

```
AT$GPSNVRAM=15,0
```

Expected:

```
OK
```

Enable GPS:

```
AT$GPSP=1
```

Expected:

```
OK
```

The GPS location is updated after a certain amount of time (a few seconds up to a few minutes), depending on GPS signal strength and previously stored GPS data.

GPS location can be obtained with:

```
AT$GPSACP
```

Expected sample:

```
$GPSACP:
152324.000,5267.1849N,00854.8107W,3.00,310.0,3,000.00,0.00,0.00,200412,05
OK
```

§ §

8.0 Sample Applications

This section describes sample applications that can be used with the Intel® Quark™ SoC drivers. The generic_buffer application is available as a source file under the kernel source tree at <kernel>/drivers/staging/iio/Documentation.

8.1 Generic Buffer

generic_buffer is a sample application that demonstrates how to retrieve buffered samples from an ADC driver via the Industrial I/O (IIO) sysfs interface.

This particular example uses the AD7298 ADC driver (see Section 7.1), however, other IIO ADC drivers may also be used.

This example uses the IIO sysfs trigger option, which allows an application or script to explicitly trigger each sampling event, by writing a dedicated file under sysfs. This gives the application control over the timing and quantity of samples collected from the ADC. However, as each trigger incurs the overhead of a system call, this method is not recommended where maximum sampling rates are needed.

Perform the steps below to use generic_buffer for gathering buffered samples from the desired ADC driver:

1. Load the necessary kernel modules:

```
# modprobe ad7298

# modprobe iio-trig-sysfs
```

2. Enable a sysfs trigger that allows us to trigger the driver from user-space to collect a new set of samples from the selected ADC channels:

```
# echo 0 > /sys/bus/iio/devices/iio_sysfs_trigger/add_trigger
```

3. Select the ADC channels that you want to sample. Here's a suggested list:

```
# echo 1 > /sys/bus/iio/devices/iio\:device1/scan_elements/in_timestamp_en

# echo 1 > /sys/bus/iio/devices/iio\:device1/scan_elements/in_current0_rms_en

# echo 1 > /sys/bus/iio/devices/iio\:device1/scan_elements/in_current1_rms_en

# echo 1 > /sys/bus/iio/devices/iio\:device1/scan_elements/

in_power0_apparent_en

# echo 1 > /sys/bus/iio/devices/iio\:device1/scan_elements/

in_power0_avg_act_en

# echo 1 > /sys/bus/iio/devices/iio\:device1/scan_elements/

in_power0_avg_react_en
```

```
# echo 1 > /sys/bus/iio/devices/iio\:device1/scan_elements/in_power0_factor_en

# echo 1 > /sys/bus/iio/devices/iio\:device1/scan_elements/

in_power1_apparent_en

# echo 1 > /sys/bus/iio/devices/iio\:device1/scan_elements/

in_power1_avg_act_en

# echo 1 > /sys/bus/iio/devices/iio\:device1/scan_elements/

in_power1_avg_react_en

# echo 1 > /sys/bus/iio/devices/iio\:device1/scan_elements/in_power1_factor_en

# echo 1 > /sys/bus/iio/devices/iio\:device1/scan_elements/in_voltage0_rms_en

# echo 1 > /sys/bus/iio/devices/iio\:device1/scan_elements/in_voltage1_rms_en
```

4. Run the data collection sample application with the following parameters:

```
# ./generic_buffer -s -w 2000 -c 1 -n ad7298 -t sysfstrig0 -l 2000 -o

output.csv
```

where:

`-s`	Use a sysfs-type trigger.
`-w 2000`	Delay for 2000 microseconds between each invocation of the trigger.
`-c 1`	Collect 1 set of samples. Buffered samples are output after each set.
`-n ad7298`	Name of the IIO device to use.
`-t sysfstrig0`	Name of the IIO trigger to use.
`-l 2000`	Number of samples to collect in each set.
`-o output.csv`	Name of output file to save buffered samples to in CSV format.

The expected result is an output file with header line and 2000 lines of samples. One column contains a timestamp value, expressed in nanoseconds, which should show that the samples are approximately 3300 microseconds apart on average (which translates into a sample rate of approximately 300 Hz). This 3300 microsecond interval is comprised of the 2000 microsecond delay specified, as well as the overhead incurred in the execution of the trigger via sysfs.

8.2 Generic Buffer High Resolution Timer

This application is similar to the generic_buffer application described in Section 8.1, however, it uses a different IIO trigger option, called the High-Resolution Timer trigger. When configured and enabled, this trigger operates at kernel level, using a high-resolution timer interrupt source (if available) to trigger IIO sampling at a desired frequency.

The trigger frequency is set via sysfs. The trigger is associated with the IIO ADC driver and, when buffered sampling is enabled for that driver, the trigger automatically starts firing at the desired frequency and runs until the buffered sampling is later disabled.

1. Load the necessary kernel module:

```
# modprobe iio-trig-hrtimer
```



2. Instantiate the hrtimer trigger:

```
# echo 0 > /sys/bus/iio/devices/iio_hrtimer_trigger/add_trigger
```

3. Enable the set of ADC channels to be sampled as described in Section 8.1, step 3.

4. Run the data collection sample application with the following parameters:

```
# ./generic_buffer_hrtimer -f 100 -p 10 -c 1 -n ad7298 -t hrtimer_trig0 -o
output.csv
```

where:

`-f 100`	Sampling frequency - number of samples to collect per second
`-p 10`	Sampling duration in seconds
`-c 1`	Collect 1 set of samples. Buffered samples are output after each set.
`-n ad7298`	Name of the IIO device to use.
`-t hrtimer_trig0`	Name of the IIO trigger to use.
`-o output.csv`	Name of output file to save buffered samples to in CSV format.

The expected result is an output file with header line and approximately 1000 lines of samples. One column contains a timestamp value, expressed in nanoseconds, which should show that the samples are approximately 10000 microseconds apart on average.

§ §

9.0 Secure Boot Implementation

9.1 Overview

A key feature of the Intel® Quark™ SoC X1000 is the concept of secure boot. Secure boot means that only authenticated software that has been cryptographically verified can be run on a secure SKU Intel® Quark™ SoC X1000 system.

The concept is predicated on a root-of-trust (RoT) from the reset vector, through to the run-time kernel. Each phase of the boot verifies the next phase of the boot, before handing off to that phase.

In this way, Intel® Quark™ SoC X1000 reference software stack provides a mechanism to ensure only authenticated software can be booted on a Intel® Quark™ SoC X1000 system.

There are two variants of Intel® Quark™ SoC X1000:

- Secure boot enabled (called secure SKU)
- Non-secure boot enabled (called base SKU or non-secure SKU)

Both variants enable Isolated Memory Regions (IMRs) during boot, through bootloader and kernel. However, only the secure SKU of Intel® Quark™ SoC X1000 requires cryptographic authentication of images in order to boot.

9.2 Isolated Memory Regions

IMRs are used extensively by grub and Linux* to provide extra security during boot. IMRs can be used to define fine-grained access masks to defined memory regions. These access masks prevent bus masters, from accessing particular memory regions based on the definitions of access rights for a given memory region associated with an IMR.

There is a total of 8 IMR regions.

Refer to Intel® Quark™ SoC X1000 Secure Boot Programmer's Reference Manual for a detailed description of IMR settings throughout the boot process.

9.3 Bootloader Security

The reference second stage bootloader solution carries out two important functions in terms of secure boot:

- Asset verification
  - Kernel
  - Bootloader config file - grub.conf
  - InitRD
- IMR setup/teardown



  - — IMR setup for kernel boot params
  - — IMR setup for compressed kernel image

This reference solution maintains a chain of trust through bootloader into kernel by ensuring that all assets executed have been validated and encapsulated within an IMR.

9.3.1 Asset Verification Flow

Grub verifies any kernel, init-ramdisk or grub configuration file, it relies upon in secure boot mode.

Grub executes the boot logic given to it in `grub.conf`. The `grub.conf` file specifies the boot configuration. The `grub.conf` file also specifies where to find boot assets. Supported locations are:

- SPI Flash
- SD/USB mass storage device

In secure boot mode, grub will:

- Parse the master flash header to identify the location of `grub.conf`
- Read in the contents of `grub.conf`
- Verify `grub.conf` against a cryptographic signature
- For the selected menu entry in the `grub.conf` file
  - — Search for the asset and its signature
  - — Verify the asset against the asset signature

For any of the previous steps, a failure to find an asset or an asset signature, or a failure to verify an asset against an asset signature, will result in grub falling back to a restricted shell exporting a minimal amount of available commands.

9.3.2 Isolated Memory Region Flow

Grub is booted by EDK with IMRs already configured around a number of assets.

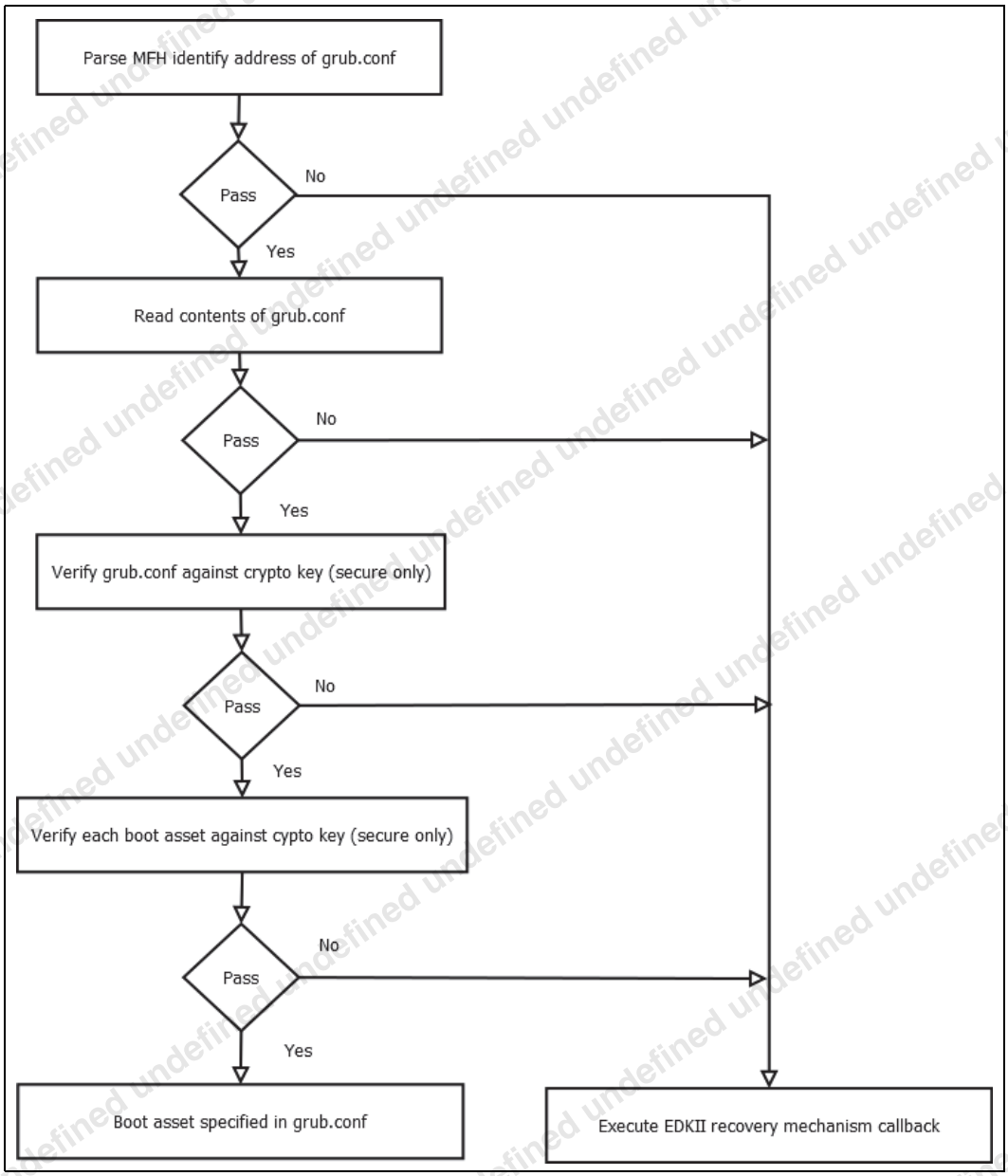
As part of the reference secure boot solution, grub will read a Linux* kernel image from SPI flash or from USB/MMC mass storage.

Refer to Intel® Quark™ SoC X1000 Secure Boot Programmer's Reference Manual for the IMR setup flow in grub.

Grub subsequently verifies bzImage against the cryptographic key for bzImage once the compressed image is placed within the IMR protection region.

Finally, assuming verification succeeds, control is handed from grub to the compressed kernel image with which is wrapped by an IMR, restricting access to CPU read/write only.

Figure 5. Grub Secure Boot Flow



Note: On secure SKUs, grub requires an accompanying signature file in order to successfully boot. For details, see the [Build & SW User Guide].



9.4 OS Security

The reference OS solution for Intel® Quark™ SoC X1000 adds IMR protection to the uncompressed kernel as well as bringing the system to a final state in terms of IMR protection.

Specifically, the reference OS solution:

- Places an IMR around executable sections of the kernel image.
- Tears down any IMRs that are not required for the run-time system.
- Locks any unlocked IMRs.
- Provides a convenient debug interface to view the size, extent, and state of each IMR.

Refer to Intel® Quark™ SoC X1000 Secure Boot Programmer's Reference Manual for a detailed breakdown of IMR state at the OS boot/runtime stage.

9.4.1 Linux* IMR setup

9.4.1.1 Default behaviour

The reference IMR run-time solution on Intel® Quark™ SoC X1000 has the following default behavior:

- allocates an IMR region for the kernel read-only, initialized data section.
- tears down IMRs that are not needed at run-time
- locks all IMRs.

9.4.1.2 IMR setup options

The following options modify the default behaviour of the IMR setup flow. These options can only be passed to the kernel command line at boot time through grub, and are typically only used for debugging or development purposes.

An option is provided by the IMR driver not to lock all IMRs by default.imr.imr_lock=0

A second parameter disables IMR allocation altogether.

```
imr_enable=0
```

When `imr_enable=0` the driver does the following:

- does not allocate any IMR region for the kernel read-only, initialized data section

9.4.1.3 Debug Interface

For the purposes of system debug, an interface is provided in `/sys` to view the setup of the IMRs on a booted reference Intel® Quark™ SoC X1000 system.

Read data from `/sys/kernel/debug/imr_state` to view the address range of each IMR[0-7] and its state, in the run-time system.

§ §

10.0 Yocto Bitbake Distros Built Variations

In the Intel® Quark™ SoC X1000 BSP, the following build distros are supported by the Yocto bitbake system:

- image-spi - A small image capable of fitting into the on-board SPI flash
- image-full - A fully featured image to be placed on SD Card/USB Stick/eMMC
- image-fast - An image capable of fast boot on Intel® Quark™ SoC X1000 targets from SD Card/eMMC

For the best boot up speed, the r1.2.1 BSP is recommended, however the following optimization can be configured to achieve sub-10 seconds boot time:

- Disable boot menu delay in EDKII (default 1s configured in r1.2.1)
- Disable all the auto-load kernel modules in Yocto meta quark fast layer

10.1 High level Implementation Variations

Table 6 presents the high level differences between image distros. For full design details of image configurations, refer to the meta-intel-iot-devkit recipes.

Table 6. High level Implementation Variations (Sheet 1 of 2)

Name	image-full	image-spi	image-fast
Libc version	eglibc	uclibc	eglibc
UEFI caching	Enabled	Enabled	Enabled
UEFI boot menu delay	1 second	1 second	1 second
Secure Boot	Support	Support	Not support
Bootloader	Grub	Grub	Grub
Bootloader menu delay	10 second	10 second	0 second
Init Manager	systemD	systemV	systemV
Initramfs	Minimal initramfs + mountable ext3 file system image	initramfs	No initramfs Direct bootable partition

Table 6. High level Implementation Variations (Sheet 2 of 2)

Name	image-full	image-spi	image-fast
Lib/Utils packages	ethtool strace ppp linuxptp libstdc++ sysstat dmidecode python lib alsa-lib wireless-tools openssh iwlwifi libs Bluetooth libs connman libs (*for full details of the image-full package list, refer to the yocto meta-intel-iot-detkit image-full bitbake recipes)	ethtool strace ppp linuxptp libstdc++ dmidecode	Same as image-spi configuration
Build-in Kernel modules	Default[1]	Default	Default + mmc + mmc block + mmc SDHCI + mmc SDHCI PCI
Loadable Device driver module support	Default	• Subset of default Bluetooth device drivers support • Subset of default USB serial device drivers support • Subset of WLAN device drivers support	Same as image-spi configuration

1. Refer to the kernel configuration files in Yocto meta-intel-quark layer for the full details of the default kernel modules.

§ §

11.0 Platform Loadable Driver Modules Support Variations

The auto-load and loadable modules listed in Table 7 are based on image-spi configuration built.

Note: The kernel features/framework driver module dependencies (for example ALSA; CAN) are not in the scope of this chapter.

The Intel® Centrino® Advanced-N 6205 card, Intel® Centrino® Wireless-N 135 card and the Intel® Dual Band Wireless-N 7260 cards are the three options supported by the Intel® Quark™ SoC X1000 platforms via mini PCI-e bus. The iwlwifi is a single driver that supports all Wifi modules in the Intel® Quark™ BSP.

Table 7. Auto-load and Loadable Modules

Modules	Common SoC driver modules
auto-load modules	usbcore, usb_common, usbhid, stmmac_pci, stmmac, spidev spi_pxa2xx_platform, spi_pxa2xx_pci,sdhci_pci, sdhci led_class, pppoe, pppox, ppp_generic,slhc, usb_f_am u_serial, g_serial, libcomposite, configfs, pch_udc, udc_core ohci_hcd, mmc_block, mmc_core, lpc_sch, gpio_sch, evdev ehci_hcd, ehci_pci,efivars, 6lowpan_iphc, rfkill, bluetooth btusb, at24
User loadable modules	intel_qrk_gip, industrialio, kfifo_buf, industrialio_triggered_buffer, i2c_dev lwlwifi(*), cfg80211

Table 8. Platform Specific Loadable Driver Modules

Product	Platform Specific Loadable Driver Modules
Galileo Gen 2	adc1x8s102, regmap_i2c, pca9685, gpio_pca953x
Kipsbay Fab D	ad7298, cy8c9540a
Cross Hill	ad7298, max78m6610_lmu
Clanton Hill	ad7298, lis331dlh_intel_qrk, intel_qrk_audio_ctrl, st_sensors_i2c, st_sensors intel_qrk_j1708
Reliance Creek	hwmon, lm75, gpio-pcf857x, regmap_spi, sc16is7xx_spi, option

§ §