Implementation of Windows Kernel Mode driver for Battery Management and Camera Sensor

Major Project Report

Submitted in partial fulfillment of the requirements
for the degree of

Master of Technology
in
Electronics & Communication Engineering
(Embedded Systems)

By

Hardik Panchal
(13MECE28)

Electronics & Communication Engineering Branch
Electrical Engineering Department
Institute of Technology
Nirma University
Ahmedabad-382 481

May 2015

Implementation of Windows Kernel Mode driver for Battery Management and Camera Sensor

Major Project Report

Submitted in partial fulfillment of the requirements
for the degree of

Master of Technology
in
Electronics & Communication Engineering
(Embedded Systems)

By

Hardik Panchal
(13MECE28)

Under the guidance of

External Project Guide:

Mr. Parag Gulhane
Mr. Pralhad Madhavi
Intel India Technology pvt Ltd.,
Bangalore.

Internal Project Guide:

Prof. Ami Vora
Professor, EC Department,
Institute of Technology,
Nirma University, Ahmedabad.

Electronics & Communication Engineering Branch
Electrical Engineering Department
Institute of Technology
Nirma University
Ahmedabad-382 481

May 2015

Declaration

This is to certify that

a. The thesis comprises my original work towards the degree of Master of Technology in Embedded Systems at Nirma University and has not been submitted elsewhere for a degree.

b. Due acknowledgment has been made in the text to all other material used.

- Hardik Panchal

13MECE28

Disclaimer

"The content of this paper does not represent the technology,opinions,beliefs, or positions of Intel Technology India Pvt. Ltd.,its employees,vendors, customers, or associates."

Certificate

This is to certify that the Major Project entitled **"Implementation of Windows Kernel Mode driver for Battery Management and Camera Sensor"** submitted by **Panchal Hardik S.(13MECE28)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by him under our supervision and guidance. The submitted work has reached a level required for being accepted for examination.The results embodied in this major project, to the best of our knowledge,haven't been submitted to any other university or institution for award of any degree or diploma.

Date: Place: Ahmedabad

Prof. Ami Vora
Internal Guide

Dr. N.P. Gajjar
Program Coordinator

Dr. D.K.Kothari
Section Head, EC

Dr. P.N.Tekwani
Head of EE Dept.

Dr. K. Kotecha
Director, IT

Certificate

This is to certify that the Major Project entitled **"Implementation of Windows Kernel Mode driver for Battery Management and Camera Sensor"** submitted by **Panchal Hardik S.(13MECE28)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by him under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination.

Mr. Parag Gulhane
MCG-CTS Engineering Manager
Intel India Technology pvt Ltd
Bangalore

Mr. Pralhad Madhavi
Software Architect
Intel India Technology pvt Ltd
Bangalore

Acknowledgements

I would like to express my gratitude and sincere thanks to **Dr. P.N.Tekwani**, Head of Electrical Engineering Department,**Dr. N. P. Gajjar**, Coordinator of M.Tech Embedded Systems program and my Manager **Mr. Parag Gulhane**, MCG-CTS Engineering Manager, Intel Corporation for allowing me to undertake this thesis work and for his guidelines during the review process.

I am deeply indebted to my thesis supervisors **Mr. Pralhad Madhavi**, Manager, Intel Corporation and **Prof. Ami Vora**, Professor,EC Department, Nirma University for their constant guidance and motivation. I also wish to thank all other team members at Intel India for their constant help and support. Without their experience and insights, it would have been very difficult to do quality work.

I wish to thank my friends of my class for their delightful company which kept me in good humor throughout the year.

Last, but not the least, no words are enough to acknowledge constant support and sacrifices of my family members because of whom I am able to complete the degree program successfully.

- Hardik Panchal

13MECE28

Abstract

In world of mobility one of the most important component in Tablet or mobile devices is rechargeable battery, which is expected to have lowest possible battery charging time, battery life should be high and provide reliable working time span. In past there are many cases of blowing or burning of battery due to over charged or over heating so it also required that battery charging system decreases those risks.

This thesis focuses on efficient battery charging control system for Li-ion rechargeable battery specifically design for windows based devices. This report introduces software solution of controlling charging over hardware. Which requires minimum hardware like for coulomb counter, ADC, current-voltage regulators etc. This thesis also explains working of fuel gauge, how it used to measure state of charge, remaining capacity. It explains how charging source detection is done on hardware as per Battery specification 1.2 standard.

This thesis also covers functional flow of camera sensor driver and requirement of camera sensor driver.Camera sensor driver implements common control functionality which expected by Image signal Processor (ISP) driver for specific sensor.

Contents

List of Tables

List of Figures

Acronyms

ACA ..... Accessory Charger Adapter
BM ..... Battery Management
CDP ..... Charging Downstream Port
DBP ..... Dead Battery Provision
DCD ..... Data Contact Detect
DCP ..... Dedicated Charging Port
OTG ..... On-The-Go
PC ..... Personal Computer
PD ..... Portable Device
PHY ..... Physical Layer Interface for High-Speed USB
SDP ..... Standard Downstream Port
SRP ..... Session Request Protocol
TPL ..... Targeted Peripheral List
USB ..... Universal Serial Bus
USBCV ..... USB Command Verifier
USB-IF ..... USB Implementers Forum
VBUS ..... Voltage line of the USB interface
WDM ..... Windows Device Model
WDF ..... Windows Driver Foundation
ICDP ..... Charging Downstream Port Rated Current (1.5A-5A)
ICFG_MAX ... Maximum Configured Current when connected to a SDP (500mA)
ISUSP ..... Suspend current (2.5mA)
IUNIT ..... Unit load current (100mA)

Chapter 1

Introduction

In current era of portable device, Tablet and Mobile play an important role in people's life and lot of activities like reading books, making video calls relayed through those devices. The Battery is one of the important part of portable devices which determine working hours. There are different types of batter available in market form which Lithium-Ion (Li-ion) is most commonly used battery. Battery charging and discharging is chemical reaction but for Li-ion battery its energy flowing in and out as part of ion movement between anode and cathode.

1.1 Problem statement

The performance and longevity of rechargeable batteries are to a large extent governed by the quality of the charging system. Incorrect charging system makes battery unusable, lowers battery life line and sometime it leads to battery explosion due over heating.

Battery energy is instantaneously delivered to support each load/ hardware as required.For reliable operation It is require that system should maintain and observe charging level in continuous manner. At the same time, it is expected that charging shell be controlled in order to improve battery performance and operational life of battery.

1.2 Objective

To overcome problems with charging system it is required to have software Driver to control and monitor all hardware related functionality, which includes

- Battery capacity monitoring
- Battery Overcharging protection
- Battery discharge control
- Battery Temperature control while charging as recommended by "JEITA (Japan Electronics and Information Technology Industries Association)" compliance
- Charging source detection as per defined in "Battery Charging specification 1.2"

1.3 Thesis Organization

Many Tablets have on chip hardware for controlling all battery related functionality. Hardware generally include

- Battery charging IC mainly referred as "Charger", which used to control charging current and voltage, to read Battery pack temperature.
- Power Management IC (PMIC) which distribute power to all on-chip hardware and has feature like Voltage scaling, power sequencing, DC-DC conversion.

For controlling all battery related functions using above hardware Battery management driver is used. This driver is developed in Windows Kernel Mode Driver Framework (KMDF).

This thesis broadly divided into seven chapters:

- Chapter 2 describe algorithm for charging source detection done at hardware layer, maximum current allowed from charging source under various condition or device state as per recommended in Battery Specification 1.2 standard.
- Chapter 3 introduces to JEITA compliance, what can be charging current for Li-ion battery at specific temperature.
- Chapter 4 describes how fuel gauging works and its different type.
- Chapter 5 introduces basic of windows driver framework, how kernel mode device driver framework is superior to older windows device model.
- Chapter 6 shows overall working flaw and objects of battery charging driver.
- Chapter 7 shows functional flow of camera sensor Driver.

Chapter 2

Battery charging specification

The standards has been defined by Battery Charging working Group which defines limits, detection control and mechanism wich report maximum allowed current which can be sink from USB 2.0 specification for powering or charging device from charging downstream port, host, deidcated charger or hubs.

2.1 Specification introduction

The Portable Device (PDs) generally use personal computing USB port for their battery charging. which leads manufacture to create charger which having USB standard Type-A receptacle and it allows PDs to use same charging cable for charging from Dedicated wall charger or from personal computer. this standard says that if PD is connected to Hubs or USB host then it must meet current limit criteria listed below:

- 2.5 mA average when bus is in suspended state
- 100 mA when bus is not in suspended state and not configured
- 500 mA when bus is not in suspended state and its configured for 500 mA

The Portable Device is allowed to draw IDEV_CHG if connected to to a Charging Port, (i.e. CDP, DCP, ACA-Dock or ACA) irrespective of its configuration or suspended state. There is need of some mechanism which allows PDs to differentiate between SDP (Standard Downstream port) and DCP (Dedicated Charging Port), this section describes such mechanism.

It accepted that user can attached their PDs to charger from various manufacturers and should get same user experience. This specification describes rules for a compliant USB charger.

PDs having Dead Battery can draw current upto 100mA as per Dead battery provision explain in preceding sub section.

2.2 Dead Battery provision

The Dead Battery Threshold is maximum charge level of a battery such that below that charge level threshold PD will bot be able to power up. Battery is said to be dead when it is below dead battery threshold.

PDs can draw ISUSP from SDP when its in suspended state or connected as USB 2.0 Specifications and up to ICFG_MAX in configured state. The limit of ISUSP is problematic when PDs having Dead Battery because some device requires more than Iunit to charge up to level at which PDs can power up. so with ISUSP PDs will not be able to power up if they attached to SDP.

BC 1.2 defines two clause to overcome Dead battery problem.

2.2.1 DBP Un-configured Clause

The PD having Dead or Weak Battery can draw IUNIT from a Downstream Port using DBP while not configured as longs as It follows following clause:

a. after specific timeout Reduce current to ISUSP

- If PD is not ready to connect and be enumerated within TSVLD_CON_WKB after attach, then it should scale down its current to ISUSP

b. Enable VDP_SRC when attached but not connected

- PD have to enable VDP_SRC within TDBP_ATT_VDPSRC of attach
- PD have to connect within TDBP_VDPSRC_CON of disabling VDP_SRC

c. Power up and enumerate as soon as it reaches weak battery threshold

- PD can not use power to perform unrelated tasks like:
  - Charging beyond the Weak Battery Threshold
  - phone call
  - Playing a song, video or game
  - connecting to wireless connection

d. Passes inrush test

- PD having Dead or Weak Battery has to pass USB-IF compliance inrush test

The PDs is said to be in un-configured state when it is attached and not configured. It has to enter into configured state on receiving of command SET_CONFIGURATION.[1]

2.2.2 DBP – Configured Clause

The PD having Weak Battery or Dead Battery can draw upto ICFG_MAX from SDP using DBP when its in configured state and not requires to pass USBCV test. PDs shell requires to follow:

a. Responding to received tokens

- PD have to respond to any tokens, with either a NAK or any other valid USB response

b. Responding to USB reset

- Upon receiving of USB reset, The PD have to lower its current to IUNIT. PD is allowed to disconnect upon receiving a reset. While disconnected, PD can use DBP Un-configured Clause.

c. Responding to USB suspend

- Upon receiving of USB suspend,The PD have to remain connected, or reduce its current to ISUSP, or it have to disconnect.PD can use DBP Un-configured Clause in disconnected state.

d. PDs has to give full functionality after disconnect or timeout

- After TDBP_FUL_FNCTN period from attach, a PD has to remain connected or either able to pass USBCV, or it has to disconnect.PD can use DBP Un-configured Clause in disconnected state.

e. PD shell informs user thats its charging and not able to perform other task within TDBP_INFORM of attach.

2.3 Charging source or type detection

Tablet or mobile device hardware/software has to capable to detect type of physical connection and charging source with help of algorithm describe in this section, type of physical connection is shown in 2.1.

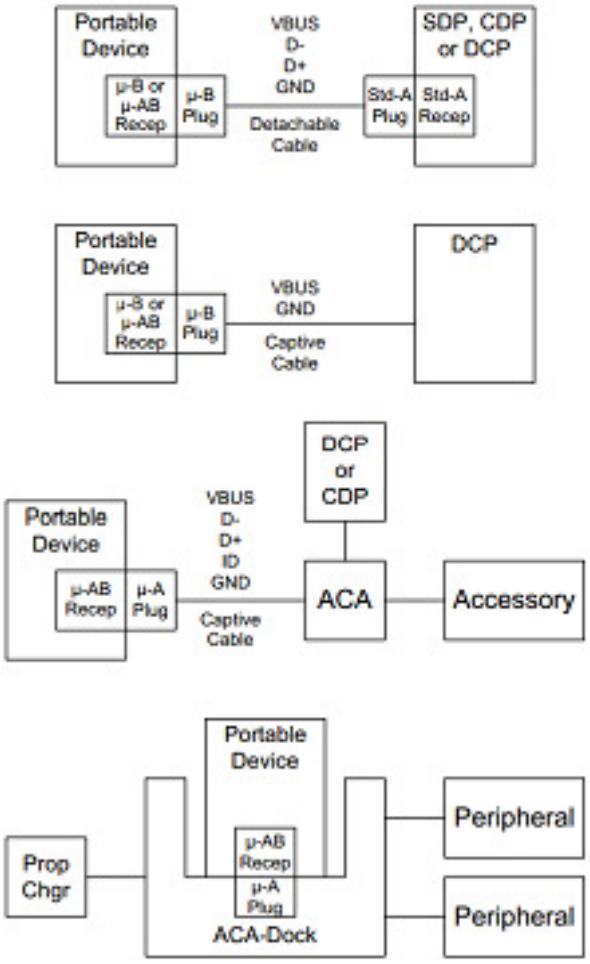

Figure 2.1: Physical Connection overview [1]

2.3.1 Charger detection hardware

This section provides information on the hardware used for charger detection. figure 2.2 shows sequence and type of task required for proper source detection.

VBUS Detect Each PDs needs comparator that detects present VBUS is higher then internal session threshold. Internal session threshold said to be valid within VOTG_SESS_VLD

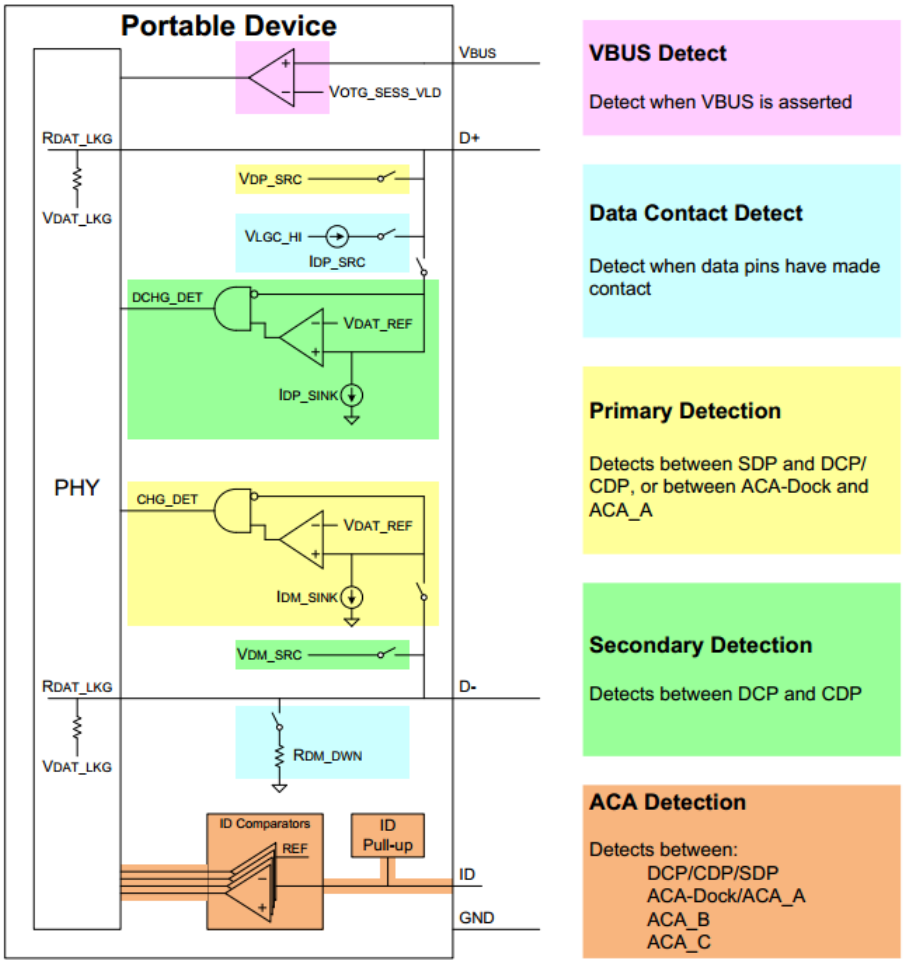

Figure 2.2: Charging Source detection hardware [1]

Data contact detect (DCD) having internal current source (IDP_SRC) used for detecting contact of Data Pin between PD and host (i.e. SDP, CDP). PD shell wait for period TDCD_TIMEOUT before staring primary detection in case it does not have DCD mechanism.

Basic purpose of DCD mechanism is to start primary detection by PD as soon as it has made data pin contact.

DCD may not work for following cases:

- DCP having higher leakage current
- ACA with charger and FS or HS B-device on Accessory Port

- ACA-Dock
- PS2 port which pulls up D+ high
- Some chargers that pull up D+ high

due to above limitation of DCD, PD shell start Primary Detection after attached event within maximum TDCD_TIMEOUT even if pin contact not detected.

USB port and receptacles are structured in way that power pin do contact before data pins as shown in figure **??**. and thus VBUS detected before data pin make contact. In general PDs take 200ms of delay in between data and power pin has made contact.

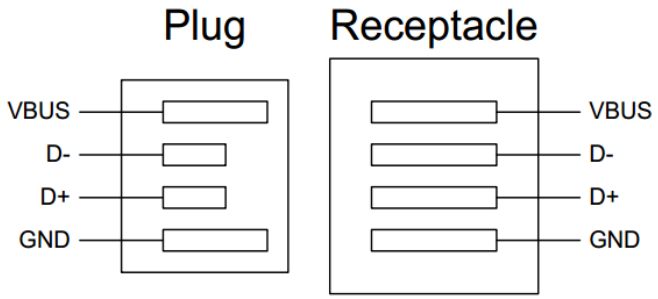

Figure 2.3: Data pin length offset

PD differentiate SDP from CDP based on primary detection, Id Data pin make contact after Primary detection, PD will know it is attached to SDP, but If PD incorrectly determine is connected to SDP than it sinks ISUSP while waiting for enumeration, but since it actually connected to CDP (which does not perform enumeration) it will not get charge.

a. Data contact detect when no device attached

   as per Data contact protocol PD shell follow those steps:

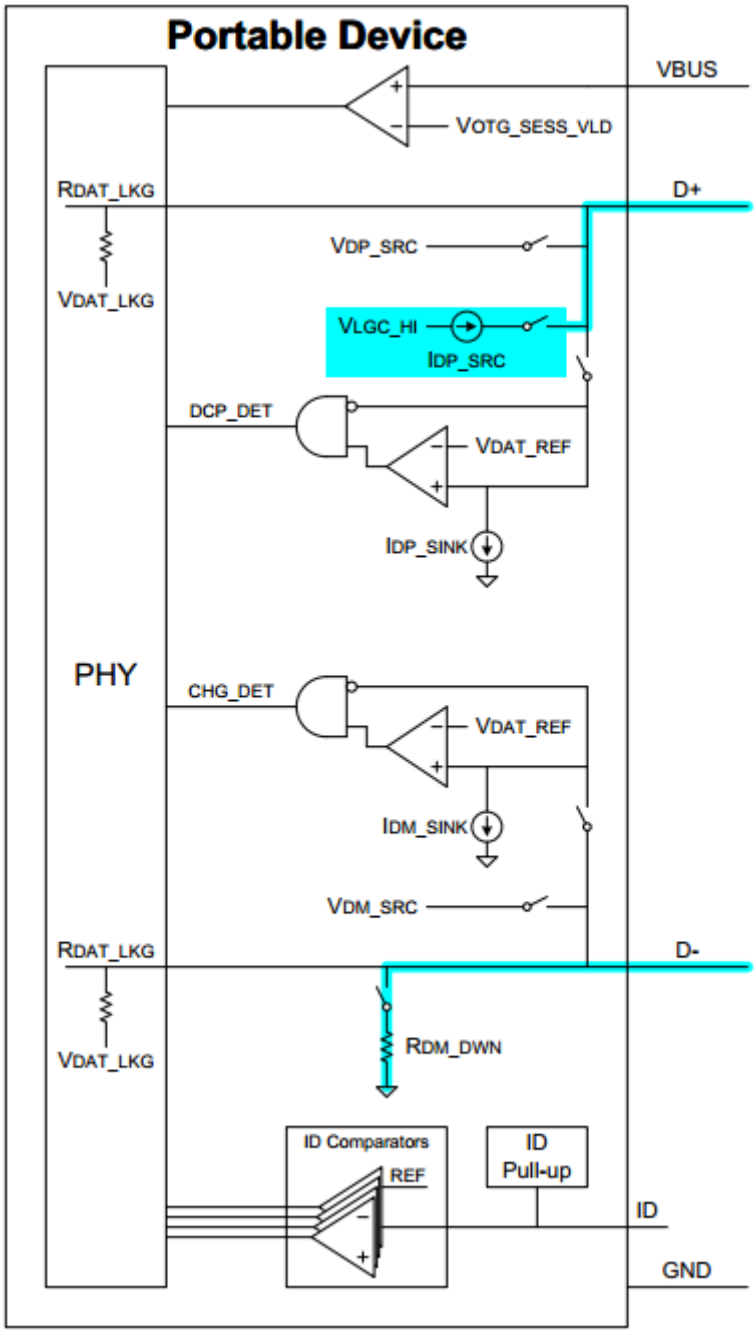

Figure 2.4: Data contact detect, not attached [1]

- PD identifies VBUS asserted
- PD turns on D- pull-down resistor and IDP_SRC

- PD shell wait for D+ line to get low for period of TDCD_DBNC
- PD shell turn off D- pull-down resistor and IDP_SRC

D+ line is on high state when nothing is attached.

b. DCD when PD attached to SDP (Standard Downstream Port)

When the PD is attached to an Standard Downstream Port, the D+ line is pulled low by RDP_DWN in the SDP.

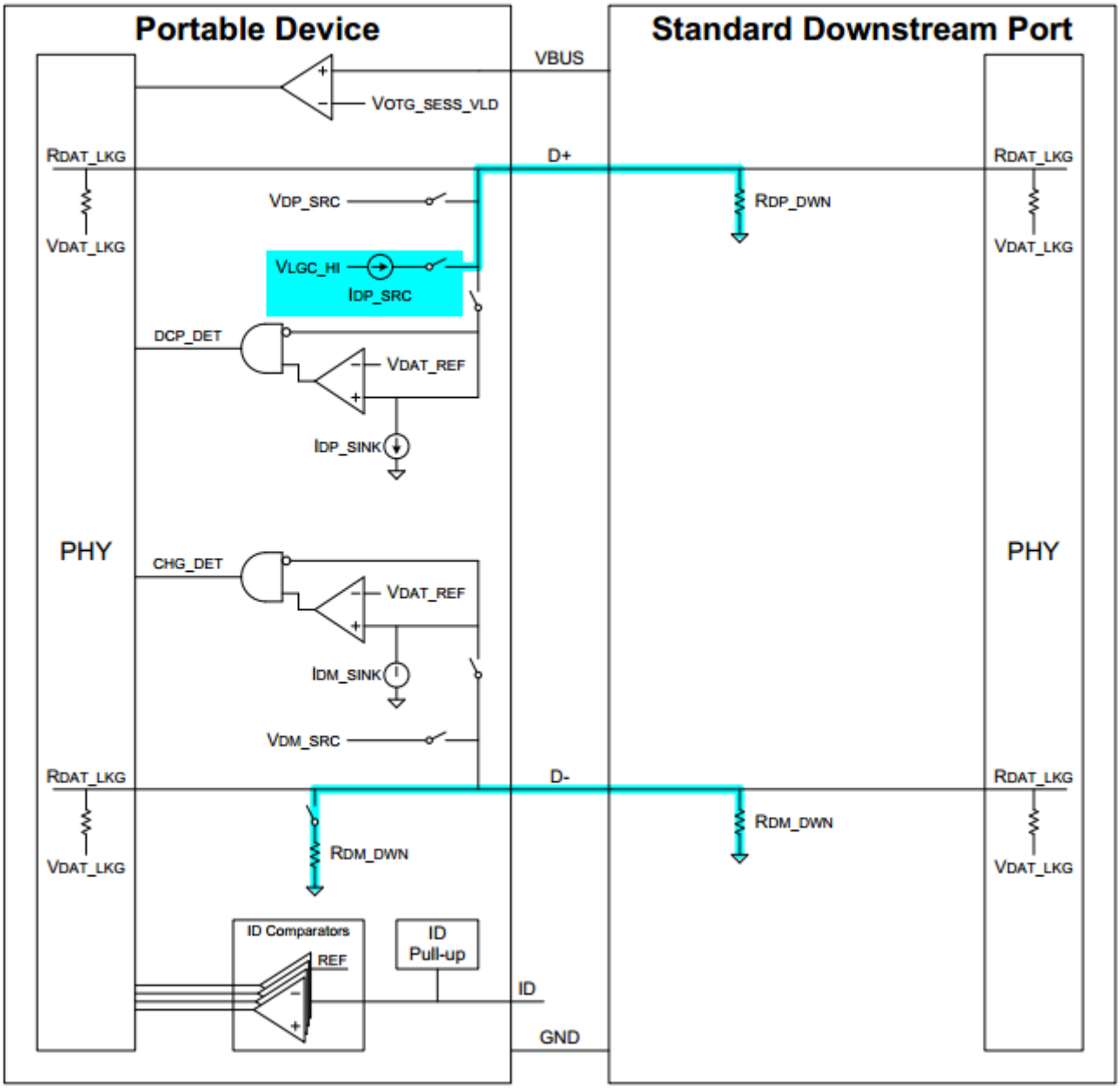

Figure 2.5: Data contact Detect SDP [1]

Primary Detection It is used to differentiate SDP and other type of charging ports. PD shell implement this algorithm.

a. Primary detection in case of DCP

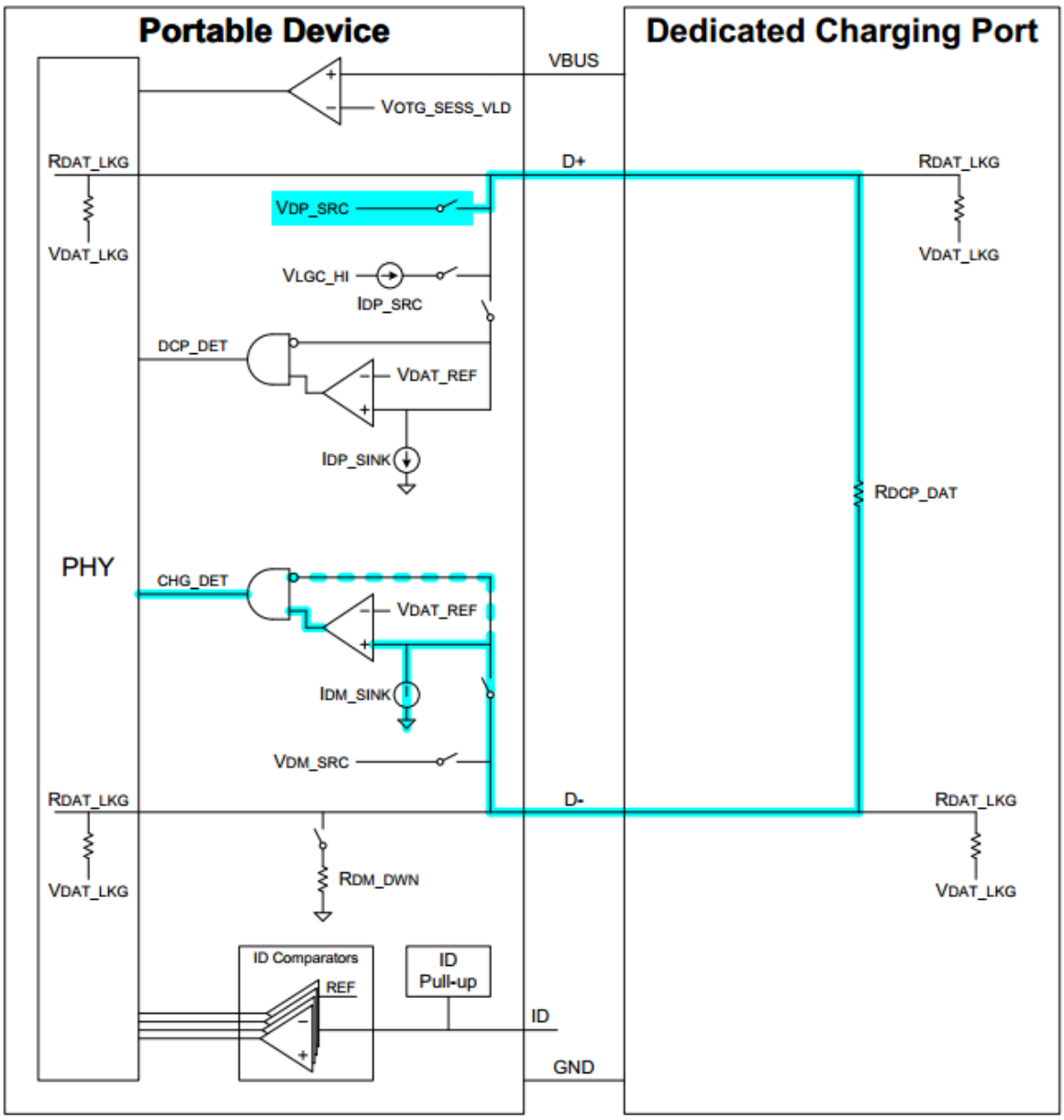

Figure 2.6: Primary detection for DCP [1]

During Primary Detection PD has to turn on VDP_SRC and IDM_SINK. As DCP short D+ to D- through a resistance of RDCP_DAT, PD will determine a voltage on D- that is near to VDP_SRC. A PD has to compare voltage on D- with VDAT_REF. If D- is higher than VDAT_REF, then the PD determines it is attached to either a DCP or CDP.

b. Primary detection in case of CDP

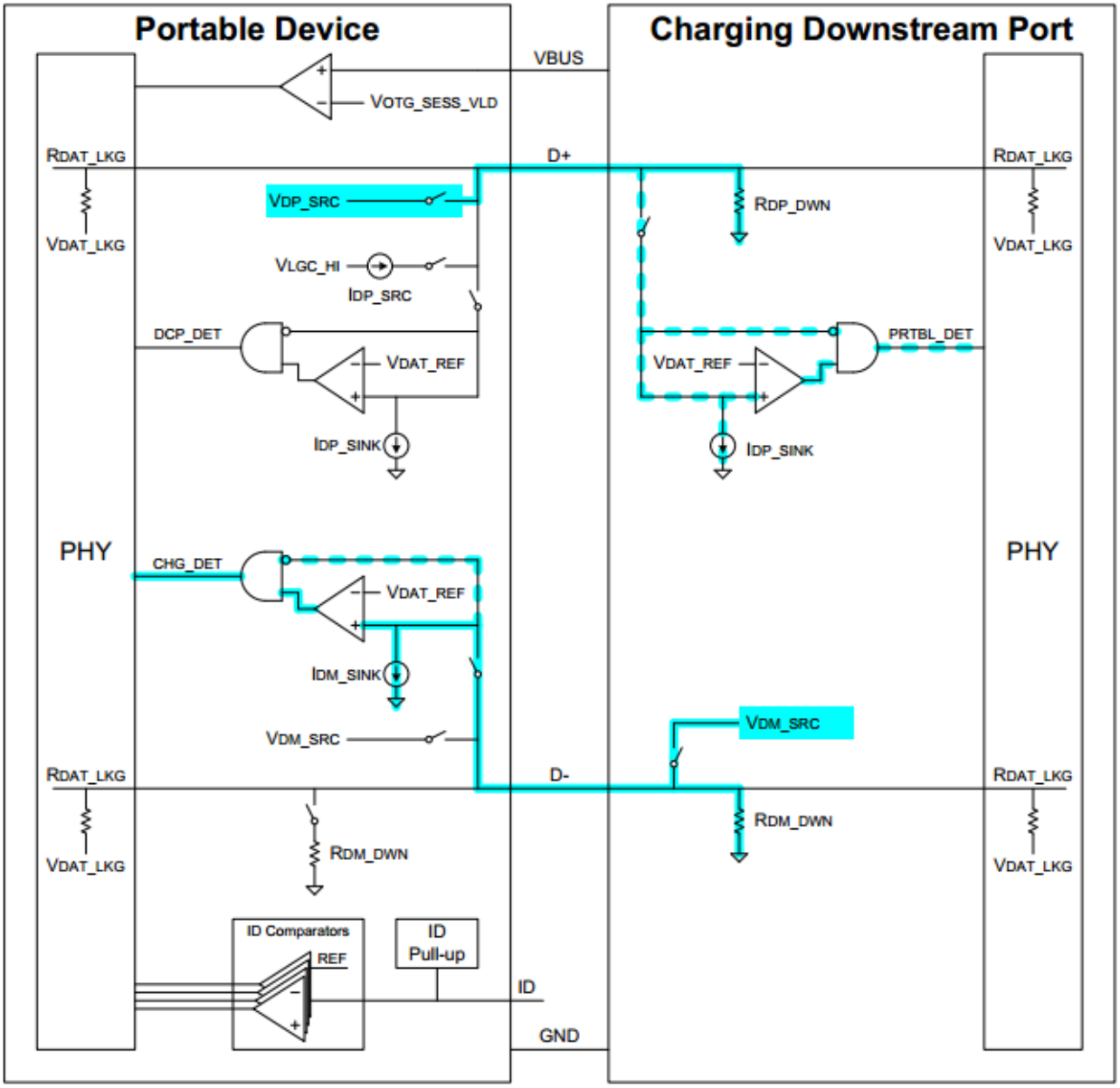

Figure 2.7: Primary detection CDP [1]

c. Primary detection when SDP is connected

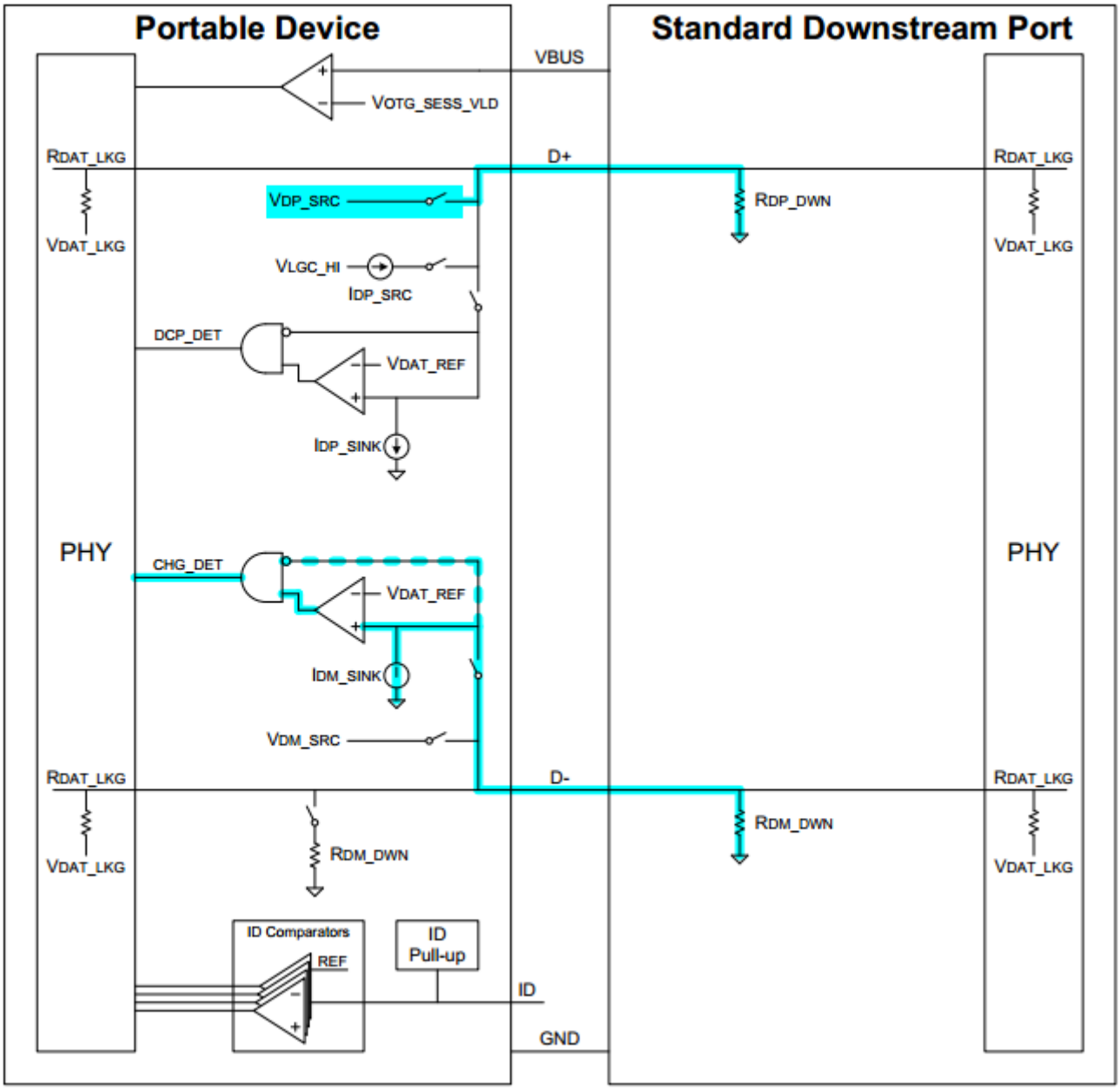

Figure 2.8: Primary detection SDP [1]

During Primary Detection PD need to turn on IDM_SINK and VDP_SRC. When a voltage of VDP_SRC is applied to D+, an SDP will continue pulling D- low through RDM_DWN. A PD will have to compare the voltage on D- with VDAT_REF. If D- is lower than VDAT_REF, then the PD determine that it is attached to an SDP.

Secondary Detection Secondary Detection used to differentiate between CDP and DCP. PDs that are not ready to be enumerated within TSVLD_CON_PWD after detecting VBUS are required to implement Secondary Detection. PDs that are ready to be enumerated can bypass Secondary Detection.

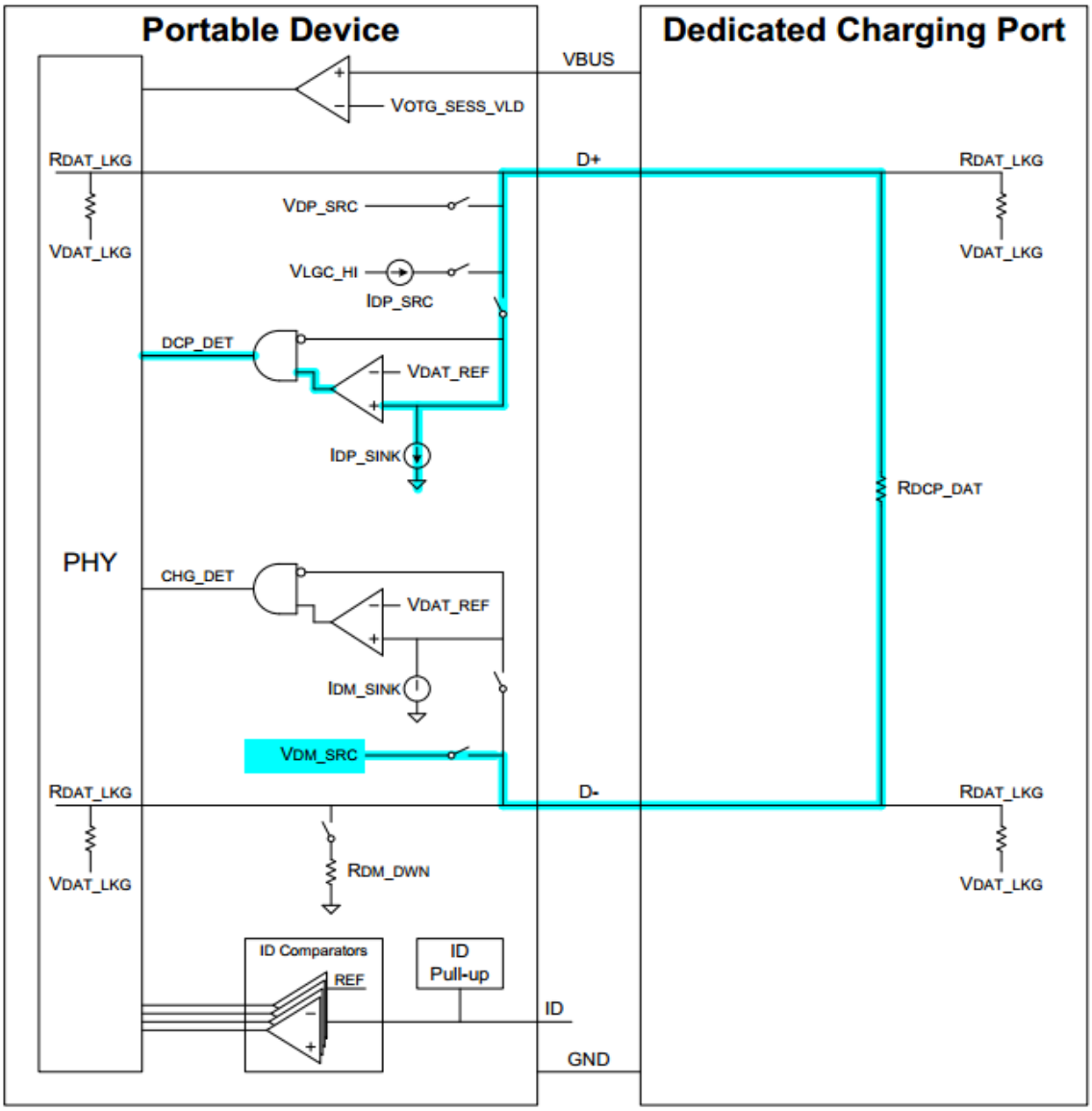

Figure 2.9: secondory detection DCP [1]

During Secondary Detection, a PD will have to output VDM_SRC on D-, turn on IDP_SINK, and compare voltage on D+ to VDAT_REF. as DCP short D+ to

D- through a resistance of RDCP_DAT, voltage on D+ will be near to VDM_SRC, which is above VDAT_REF. If a PD detects that D+ is higher than VDAT_REF, it identifies that it's attached to a DCP. It's then suppose to enable VDP_SRC or pull D+ to VDP_UP through RDP_UP, as per the Good Battery Algorithm

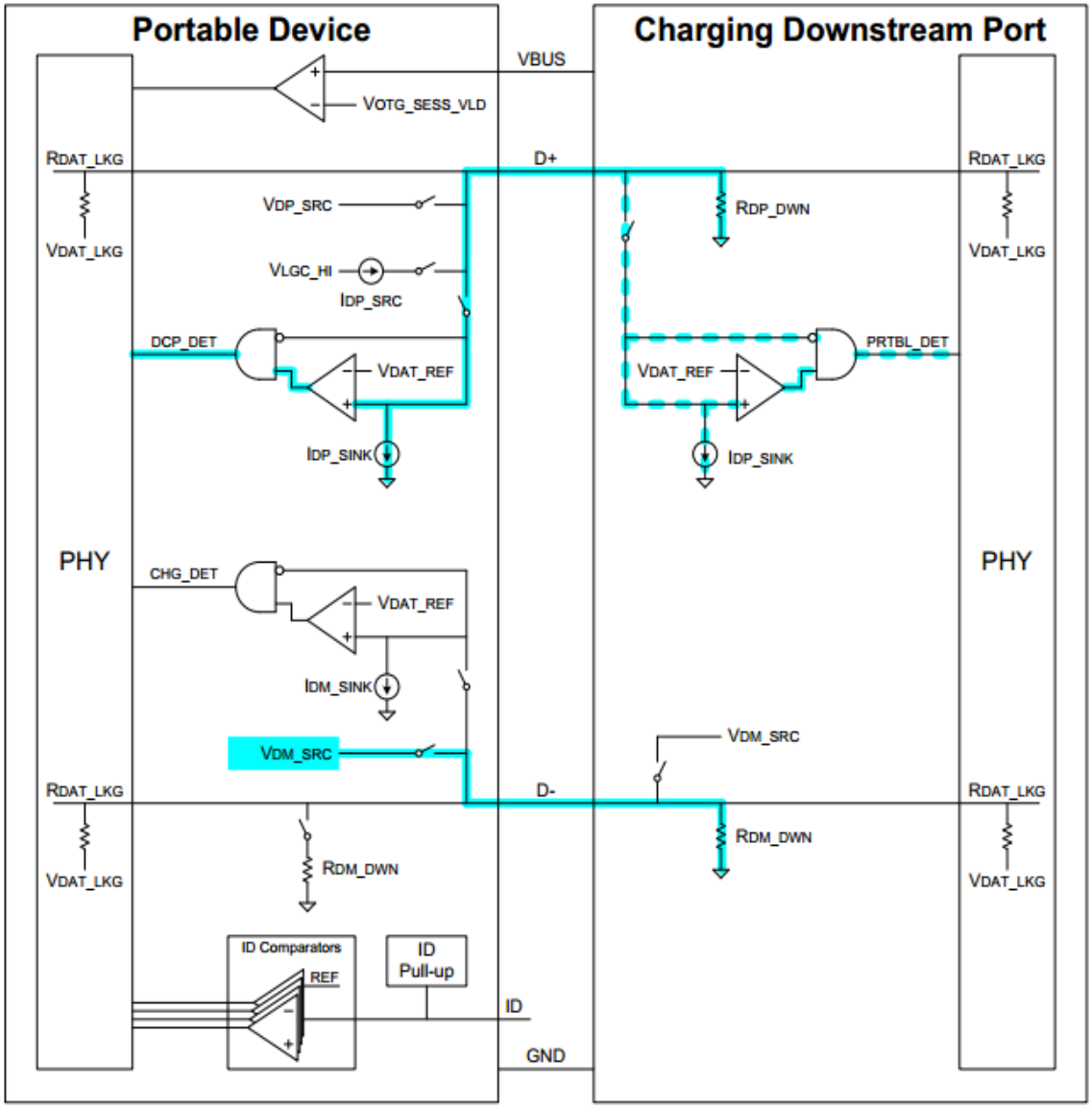

Figure 2.10: Secondary detection CDP [1]

During Secondary Detection, PD will have to output VDM_SRC on D-, turn on IDP_SINK, and compare the voltage on D+ to VDAT_REF. as CDP do not short

D+ to D-, voltage on D+ will be near to ground, which will be below VDAT_REF. If a PD detects that D+ is lower than VDAT_REF, it identifies that it is attached to a CDP.

2.3.2 Charger Detection algorithm

This section explains all charger detection steps by help of flowchart. PD vendor can choose different algorithm based on requirement. [1]

The **Weak Battery Threshold** is defined as the minimum threshold of a battery charge level such that above this threshold, a device is guaranteed to power up successfully. A Weak Battery having charge level higher than Dead Battery Threshold and lower then the Weak Battery threshold. A device with a Weak battery may or may not be able to power up a device successfully. A **Good Battery** is defined as one that is above the Weak Battery Threshold.

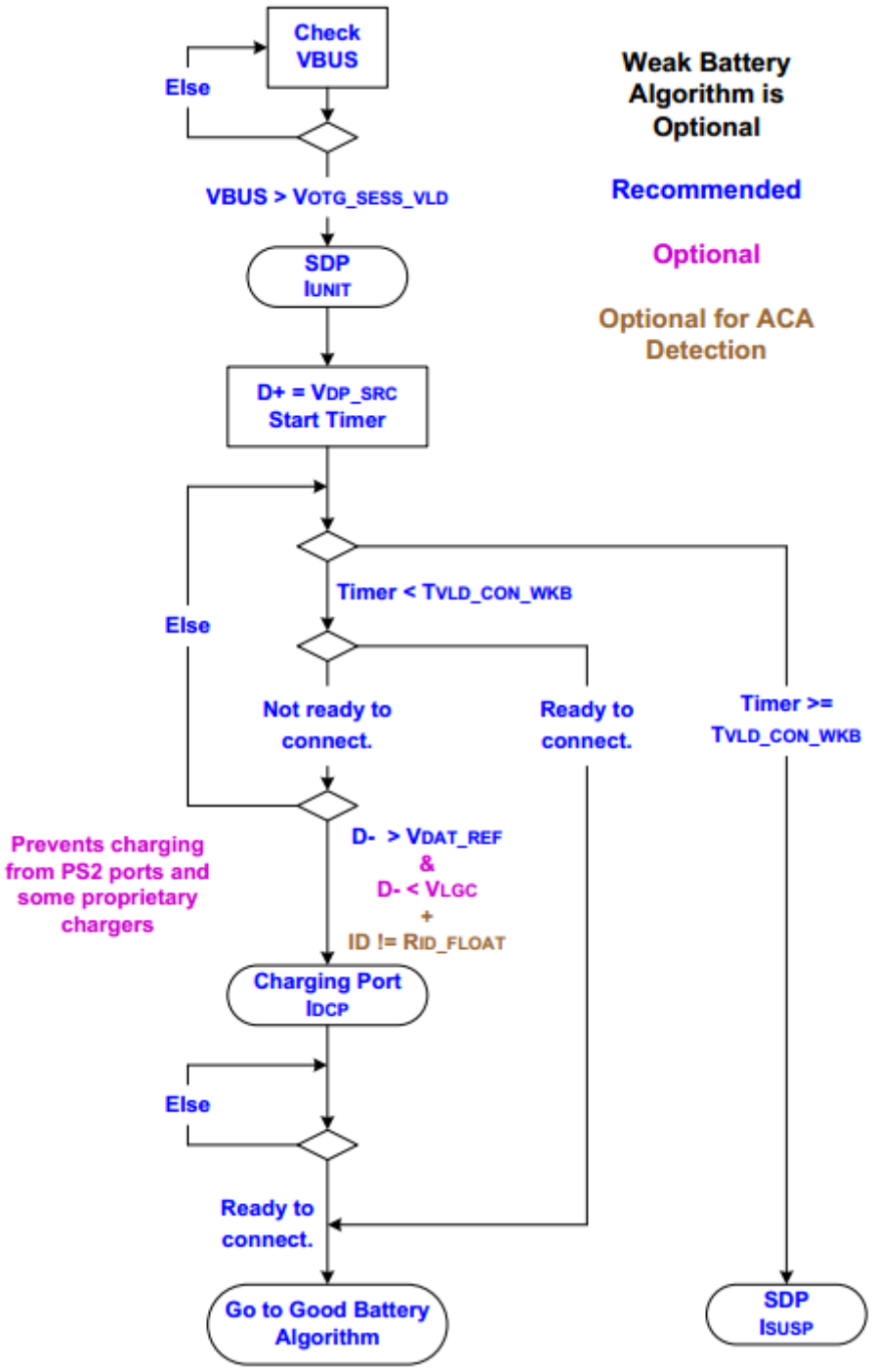

Figure 2.11: Weak battery algorithm [1]

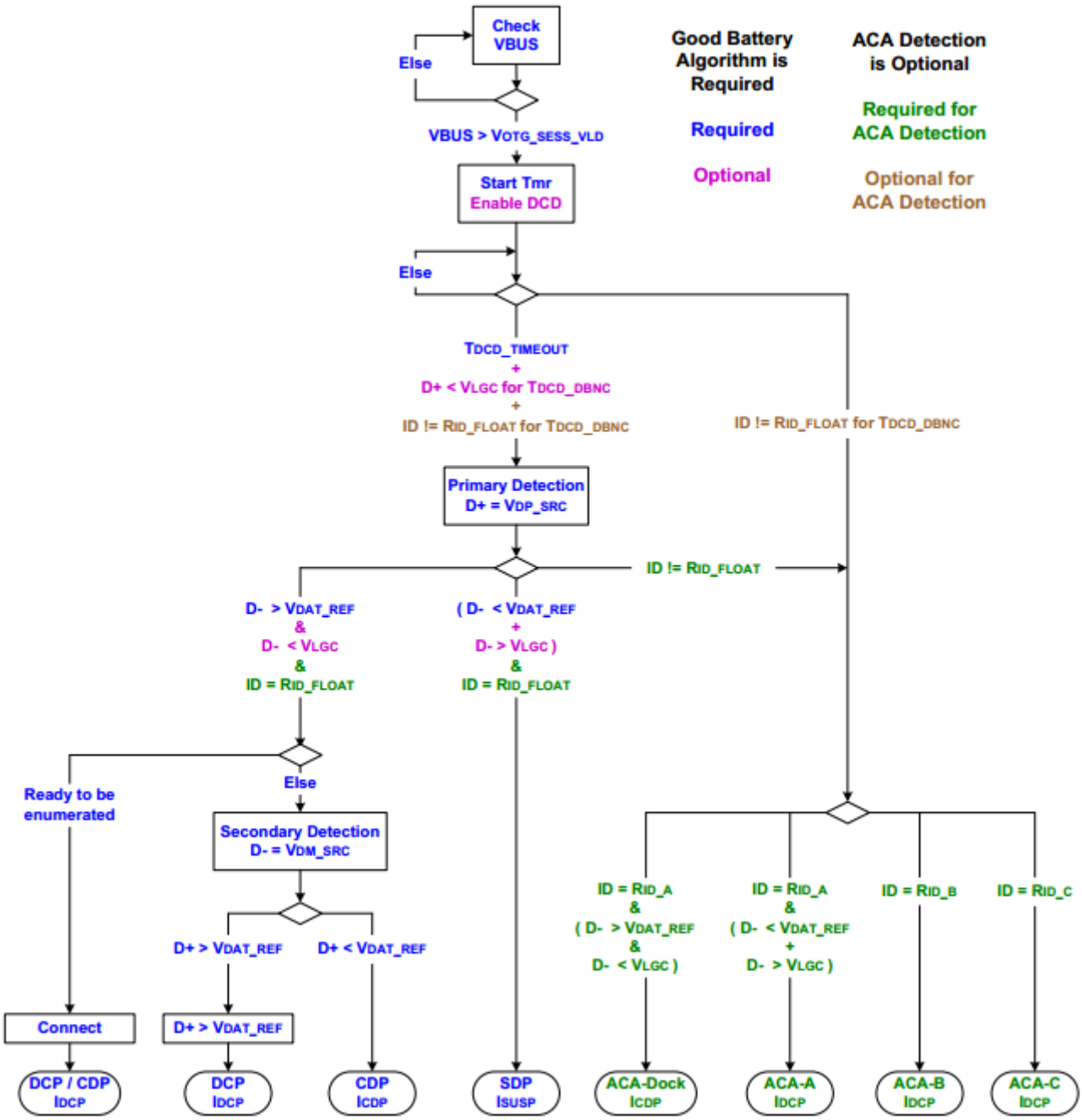

Figure 2.12: Good Battery algorithm [1]

Chapter 3

JEITA Compliance

Lithium-ion (Li-ion) batteries tend to become dangerous when they are overcharged at high temperatures. Safely charging these batteries has become one of the most important design specifications in battery-powered portable equipment. Progress has been made in establishing industry standards such as the Japan Electronics and Information Technology Industries Association (JEITA) guidelines for improving battery-charging safety.

3.1 Li-ion Battery safety

Widely used in consumer electronics from cell phones to laptops, Li-ion batteries have the highest volumetric and gravimetric energy densities among the rechargeable batteries.

Everyone in the industry has seen pictures of exploding laptops and heard about the massive and unprecedented recalls of Li-ion batteries due to cell safety concerns. Such battery explosions or fires originated within the manufacturing process. Batteries contain several metal parts that can sometimes result in undesirable metal impurities within the cell. These impurities are typically sharp metal shards from the battery casing or from electrode materials. If these shards get between the battery's electrode and separator, battery cycling in the negative electrode can eventually

cause the shards to puncture the separator. This results in a micro-short between the positive and negative electrodes, producing high heat that may ultimately result in fire and/or an explosion[6].

High temperatures, fire, and explosions are all results of thermal runaway—a condition whereby a battery enters into an uncontrollable reaction. Thermal runaway is a process in which the internal temperature of a battery with LiCoO2 as the cathode material and graphite as the anode material reaches approximately 175°C. This is an irreversible and highly exothermic reaction that can cause a fire, usually when the battery is charging[6].

3.2 JEITA guidance

To improve the safety of charging Li-ion batteries, JEITA and the Battery Association of Japan released new safety guidelines on April 20, 2007. Their guidelines emphasized the importance of avoiding a high charge current and high charge voltage at certain low and high temperature ranges. According to JEITA, problems in the Li-ion batteries occur at high charge voltages and high cell temperatures.

Figure 2 shows the JEITA guidelines for the charge current and charge voltage over cell temperature for batteries used in notebook applications. These batteries have LiCoO2 as the cathode active material and graphite as the anode active material. In the standard charging temperature range from T2 to T3, a Li-ion cell can be charged in the optimal conditions of the upper-limited charge voltage and the upper-limited charge current recommended by the cell's manufacturer for safety.

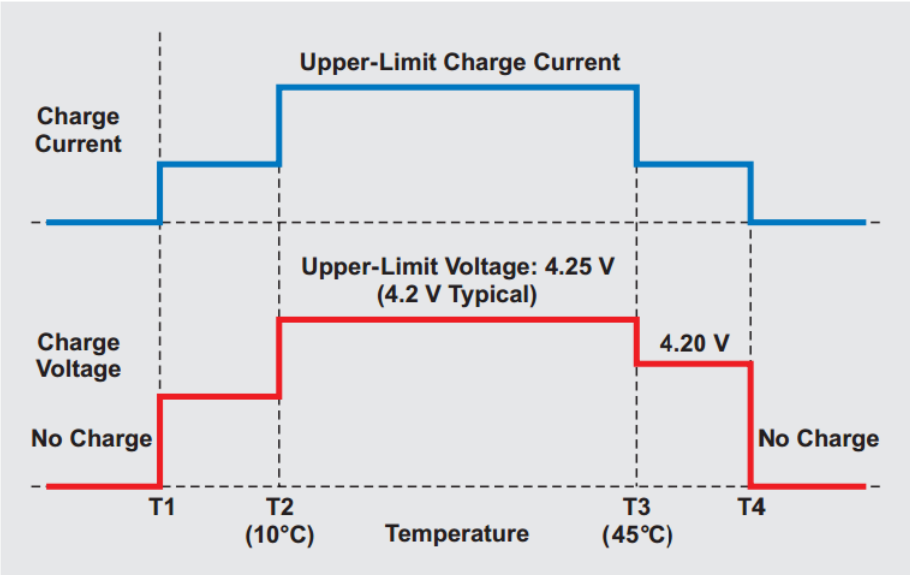

Figure 3.1: JEITA guidelines for charging Li-ion batteries [6]

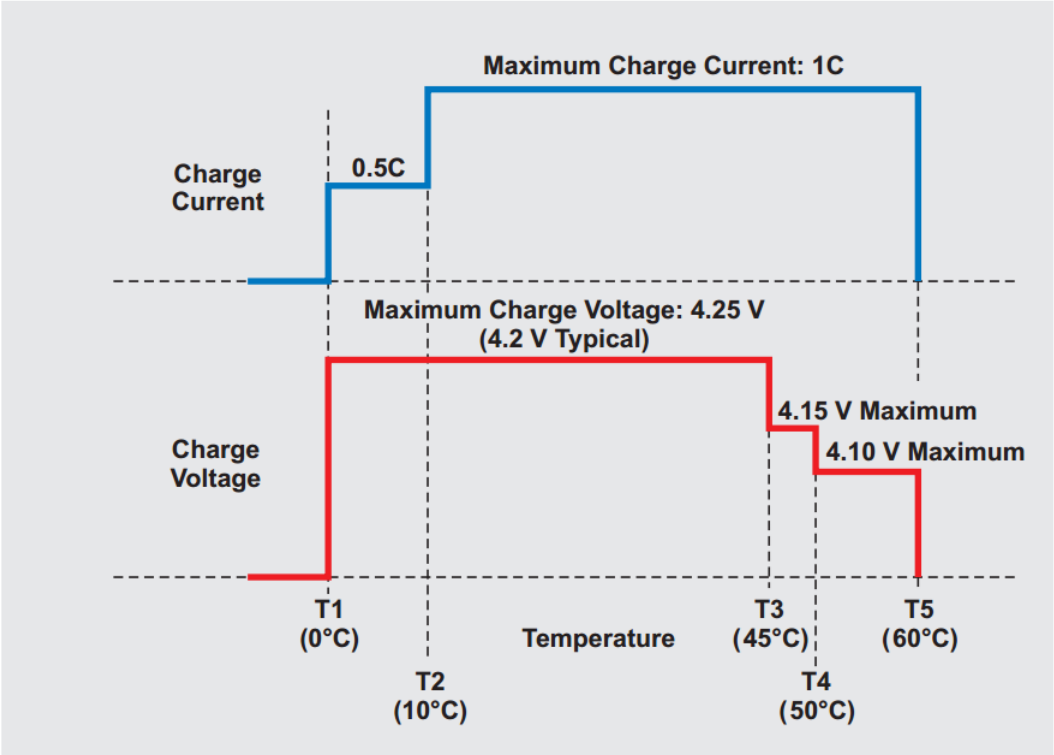

Figure 3.2: JEITA guidelines for charging Li-ion batteries in single-cell handheld applications [6]

Chapter 4

Battery Fuel gauging

Battery fuel gauges determine the amount of charge remaining in a battery and how much longer, under specific operating conditions, the battery can continue to provide power. As handheld devices become thinner and less expensive, batteries are becoming smaller yet have increased capacity. Highly accurate battery fuel gauges are needed to make efficient use of all available cell energy in today's portable devices.

Battery SOC (State of Charge) is percentage of charge left in a battery ranges from 0% to 100%. Since SOC measurement having same purpose as a gas gauge in an automobile, ICs that measure and provide SOC are typically called "gas gauge" or "fuel gauge" ICs.

Battery management driver uses this SOC to determine remaining battery life and it also ensure safety and maximizing life of battery. A non reliable Battery SOC management hardware/software results in lowering battery performance as well as life time due to over-charging or over discharging of Battery.

Li-ion cells have nearly constant voltage across its discharging (Figure 4.1). and so comparing current cell voltage is not enough for measuring remaining charge.

As a result Li-ion battery require more sophisticated fuel gauge having more than one SOC measurement techniques.

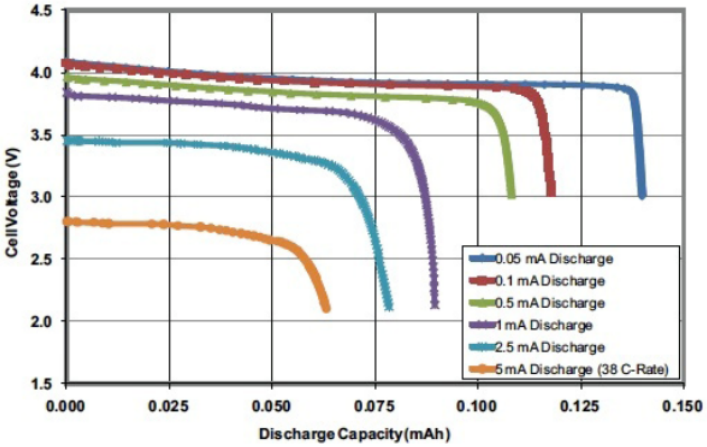

Figure 4.1: typical lithium-ion battery voltage during discharge [6]

4.1 General SOC measurement methods

a. Current based method

b. Voltage based method

c. Model based method

d. Internal impedance measurement

4.1.1 Current based SOC estimation

This method usually has counter which incremented and decremented during charging and discharging of battery respectively. Driver running on host controller calculate SOC by use of ACR (Accumulated charge Register).

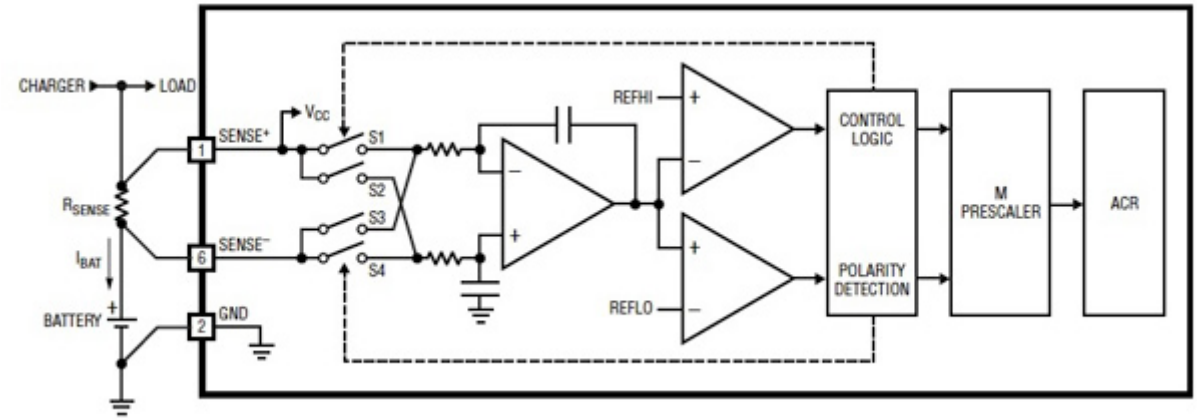

Figure 4.2: Current based (Coulmb counting) SOC functional diagram (Courtesy of Texas Instrument.)

4.1.2 Voltage based SOC estimation

Generally voltage based algorithm include one look up table known as OCV profile for specific battery, at no load or minimum load battery voltage measurement done by fuel gauge which is then compared to profile data by means of software to decide current battery capacity.

Each method has its own advantage and disadvantage, BM Driver uses both of above explained method for SOC estimation. It uses coulomb counting method while system is running and voltage based method while system is on sleep state or having minimum load. also Driver measure Voltage based method when system boot for first time since battery chemical is on stable state.

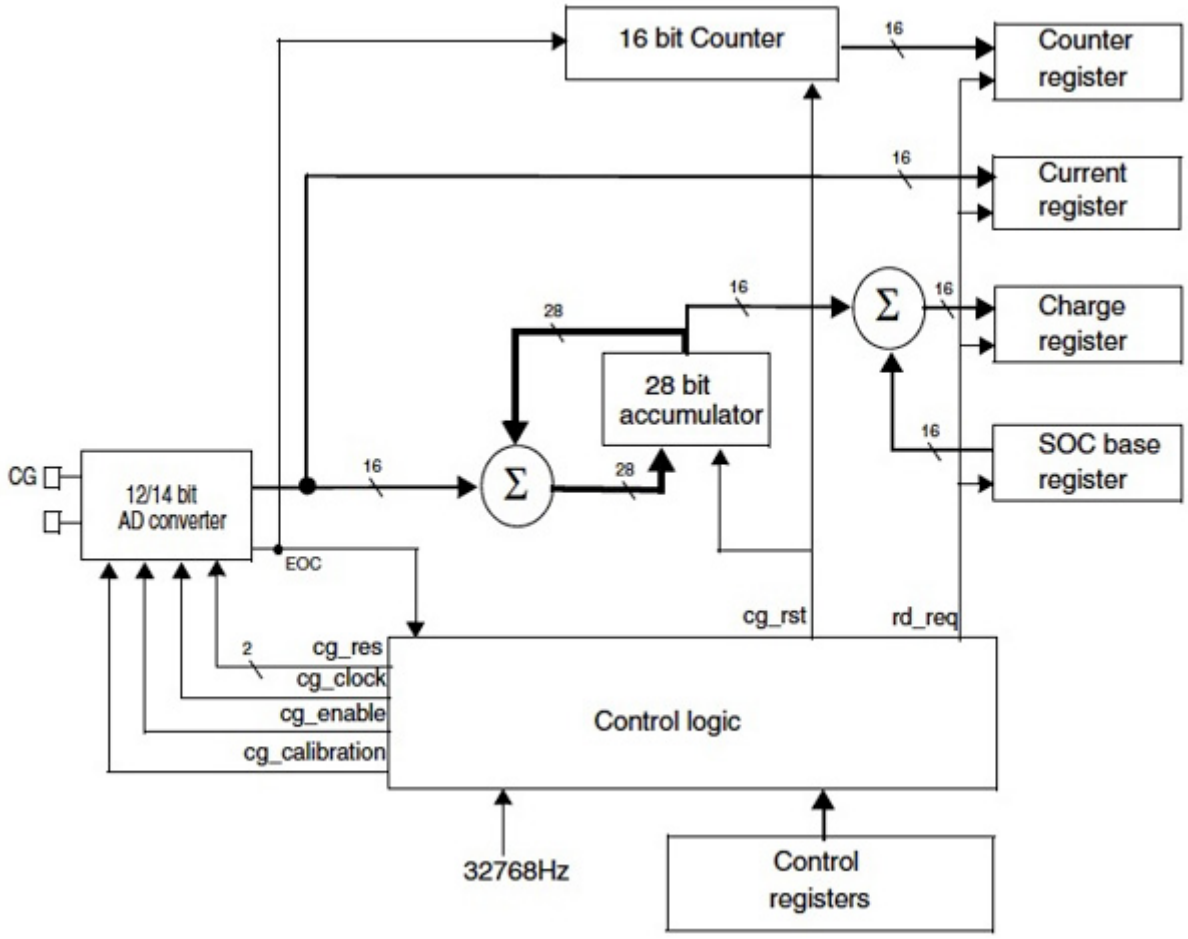

Figure 4.3: ST3105 dedicated digital coulomb counter. (Courtesy of STMicroelectronics.)

	Capacity (%)	OCV (V)
Breakpoint 8	100	
Breakpoint 7	85	
Breakpoint 6	55	
Breakpoint 5	45	
Breakpoint 4	25	
Breakpoint 3	15	
Breakpoint 2	10	
Breakpoint 1	5	
Breakpoint 0	0	

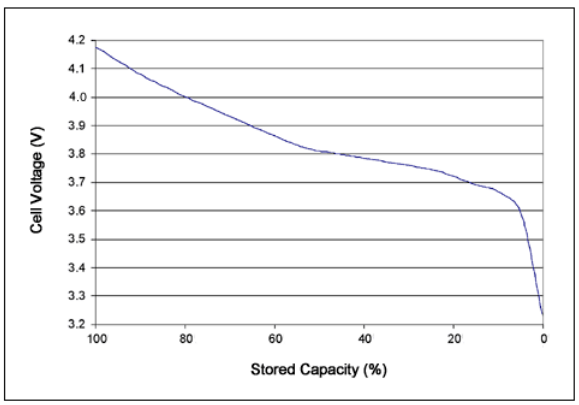

Figure 4.4: Figure illustrate chart for sample OCV profile

Chapter 5

Windows Driver Frameworks (WDF)

Windows Driver Frameworks (WDF) is a set of libraries that you can use to develop device drivers that are interoperable with Windows. WDF is comprised of Kernel-Mode Driver Framework (KMDF) and User-Mode Driver Framework (UMDF). This section mainly concentrate on explaining KMDF on which Battery management driver is developed [2].

5.1 Windows§Driver Model

The Windows Driver Model (WDM) was defined to provide a common driver development model and paradigm for Windows 98, Windows 2000 and later Windows XP [3]. It aimed to reduce the complexity of implementing a Plug 'n Play compatible device driver, and provide standard and secure way of interacting with OS Kernel. since any Issue or crash in driver cause whole system instability even fail of system.

WDM features implemented in way allowing standard abstraction and layering of Driver. for example battery charger driver uses Battery Class driver which is

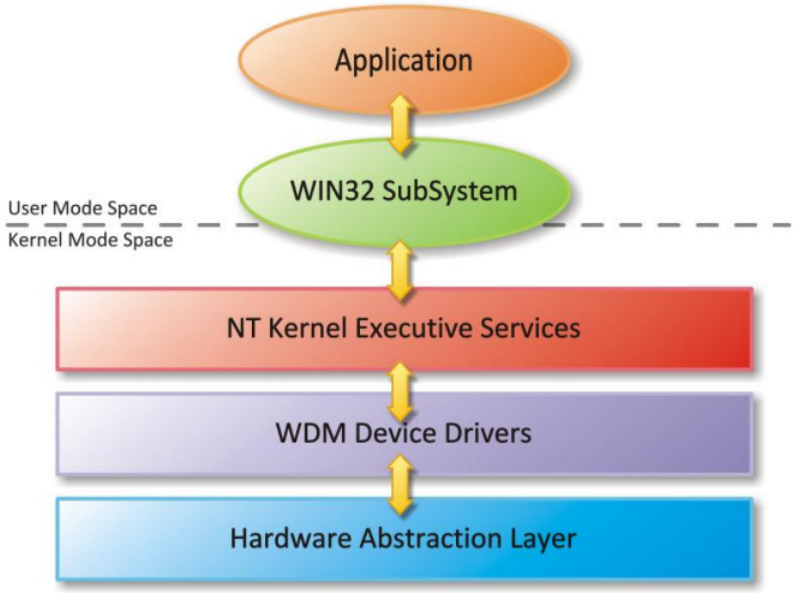

Figure 5.1: Microsoft windows Driver system [3]

implemented by Microsoft as standard layering, manufacture provide filter driver for battery charging for proprietary feature which work on HAL layer[3].

For most devices that Microsoft doesn't directly support, you need to write a WDM driver. You will decide first whether to write a monolithic function driver, a filter driver, or just a mini driver. You'll probably never need to write a class driver because Microsoft would like to reserve that specialty to itself in order to serve the broadest range of hardware makers.

5.1.1 WDM Mini-drivers

The basic rule of thumb is that if Microsoft has written a class driver for the type of device you're trying to support, you should write a mini-driver to work with that class driver. Your mini-driver is nominally in charge of the device, but you'll call subroutines in the class driver that basically take over the management of the hardware and call back to you to do various device-dependent things. The amount of work you need to do in a mini-driver varies tremendously from one class of device

to another.

Here are some examples of device classes for which you should plan to write a mini-driver:

- Non-USB human input devices (HID), including mice, keyboards, joysticks, steering wheels, and so on. If you have a USB device for which the generic behavior of HIDUSB.SYS (the Microsoft driver for USB HID devices) is insufficient, you would write a HIDCLASS mini-driver too. The main characteristic of these devices is that they report user input by means of reports that can be described by a descriptor data structure. For such devices, HIDCLASS.SYS serves as the class driver and performs many functions that Direct-Input and other higher layers of software depend on, so you're pretty much stuck with using HIDCLASS.SYS. This is hard enough that I've devoted considerable space to it later in this as an aside, HIDUSB.SYS is itself a HIDCLASS mini-driver.

- Streaming devices, such as audio, DVD, and video devices, and software-only filters for multimedia data streams. You will write a stream mini-driver.

- Batteries, for which Microsoft supplies a generic class driver. You would write a mini-driver (which the DDK calls a mini-class driver, but it's the same thing) to work with BATTC.SYS.

5.1.2 WDM Filter Drivers

You may have a device that operates so closely to a recognized standard that a generic Microsoft driver is almost adequate. In some situations, you may be able to write a filter driver that modifies the behavior of the generic driver just enough to make your hardware work. This doesn't happen very frequently, by the way, because it's often not easy to change the way a generic driver accesses the hardware.

5.1.3 Monolithic WDM Function Drivers

WDM function driver. Such a driver essentially stands alone and handles all the details of controlling your hardware. When this style of driver is appropriate, I recommend the following approach so that you can end up with a single binary that will work on Intel x86 platforms in all operating systems. First, build with the most recent DDK.

5.2 Windows Driver Foundation

WDM is still complex model while implementing PnP and Power callback for driver and create boiler-plate coding while developing driver. [4]

The Microsoft Windows Driver Foundation (WDF) is Microsoft's next-generation driver-development model. WDF includes a suite of components that support the development, deployment, and maintenance of both kernel-mode and user-mode drivers. As Figure 5.2 shows, WDF components work with existing driver development tools to address the entire driver life cycle.[4]

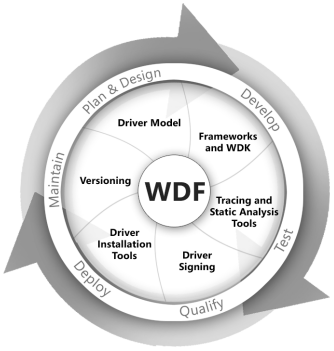

Figure 5.2: Microsoft Windows Driver Foundation Development Model [3]

- **Driver model**: The WDF driver model supports the creation of object-oriented, event-driven drivers. By using WDF, driver writers can focus on

their device hardware, rather than on the operating system. WDF drivers can be written for either kernel mode or user mode.

- **Frameworks and the Windows Driver Kit (WDK**): WDF defines a single driver model and includes frameworks for both kernel-mode and user-mode driver development. The frameworks provide the basic infrastructure to support the WDF model. They implement common features, provide intelligent defaults, and manage most interactions with the operating system.

The kernel-mode driver framework (KMDF) implements basic kernel-mode driver support features that are required by Windows and are common to all kernel-mode drivers.

The user-mode driver framework (UMDF) provides functional support similar to that in the KMDF, but enables drivers for some types of devices to run in user mode instead of in kernel mode.

All WDF drivers are built by using the WDK build environment.

- **Tracing and static analysis tools**: Both the KMDF and the UMDF have built-in verification code and support integrated tracing through Event Tracing for Windows (ETW). The generated traces can help in debugging drivers during development and in diagnosing problems in released drivers. WDF drivers also work with the existing driver verifier. In addition, compile-time driver verification tools, such as PREfast and Static Driver Verifier (SDV), are also part of the WDF effort.

- **Driver signing:** WDF drivers are signed in the same way as Windows Driver Model (WDM) drivers.

- **Driver installation tools:** WDF drivers are installed by using INF files and work with existing driver installation tools, including the Driver Install Frameworks (DIFx) tools.

- **Versioning:** WDF supports versioning so that a single driver binary can run on any version of the operating system and use the same version of the framework with which it was built and tested.

5.2.1 Design Goals for WDF

The following are the primary design principles underlying the WDF model:

- Separate the driver model from the core operating system components.
- Provide a user-mode option for some device types.
- Implement common and default driver features so that driver developers can focus on their hardware.
- Make drivers event driven and define the events at a detailed level so that driver tasks are straightforward.
- Simplify Plug and Play and power management implementation for all drivers.
- Support a consistent installation process for both user-mode and kernel-mode drivers.
- Provide integrated tools, including built-in tracing and verification support, to help find and diagnose problems both during debugging and after release.
- Enable a single driver binary to work with several versions of the framework and the operating system.

5.2.2 User mode Driver Framework (UMDF)

The UMDF implements a subset of the KMDF functionality, including support for Plug and Play, power management, and asynchronous I/O. Drivers that run in user mode have access only to the user address space and therefore pose low risk to

system stability. User-mode drivers cannot handle interrupts, perform DMA, or use kernel-mode resources such as non-paged pool.

Using the UMDF, developers can create drivers for any protocol- or serial-bus–based device. Although these drivers run in user mode, they use the standard Plug and Play installation mechanism and the same I/O model as kernel-mode WDF drivers. To determine whether a user-mode driver is suitable for your device, see "Introduction to the WDF User-Mode Driver Framework," which is listed in the Resources section.

Figure 5.3 shows the components involved in transmitting an I/O request from an application to a user-mode WDF driver. [4]

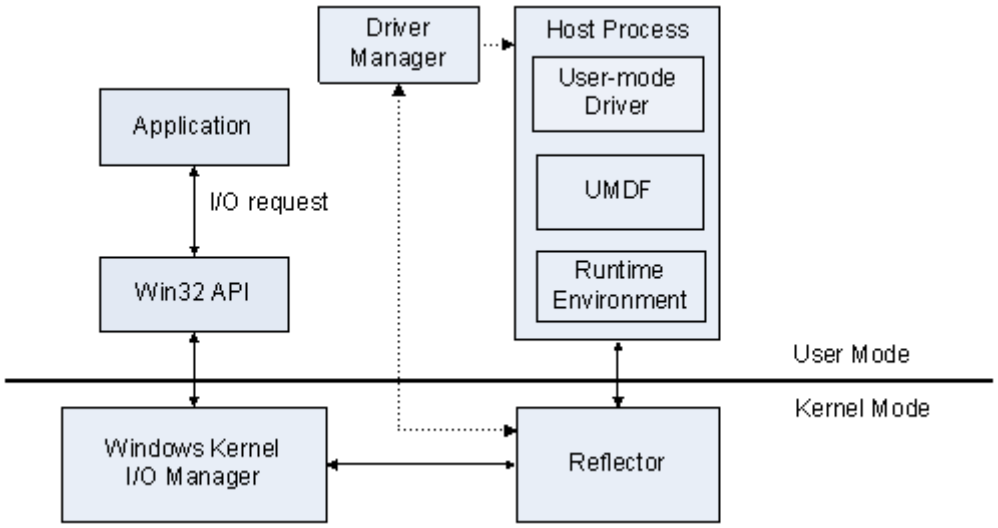

Figure 5.3: I/O Flow to User-Mode WDF Driver [4]

- **Application:** The application is a user-mode process that issues I/O requests through the Win32 API.
- **Win32 API:** In response to the application's I/O request, the Win32 API calls I/O routines in the Windows kernel.

- **Windows Kernel:** The I/O manager in the Windows kernel creates IRPs to represent the requests and presents them to the target driver by calling the driver at a designated entry point. If the target of the request is a user-mode WDF driver, however, the I/O manager cannot call the driver or the UMDF directly because these components run in a user mode process and kernel-mode components cannot call back to user mode. Therefore, the I/O manager does not present the request directly to the user-mode driver. Instead, the I/O manager presents the request to a kernel-mode component called the reflector.

- **Reflector:** The reflector is a kernel-mode WDM filter driver that represents the user-mode driver in the kernel-mode driver stack. The reflector passes the I/O request to the user-mode driver host process.

  The reflector manages communication between the kernel-mode components and the user-mode driver host process. It monitors the driver host process to ensure that it responds properly to messages and completes critical operations in a timely manner, thus helping to prevent driver and application hangs. The reflector also sends messages to the driver manager as required.

- **Driver Host Process.** The driver host process is the user-mode process in which the user-mode driver runs.

5.2.3 Kernel mode Driver Framework (KMDF)

The kernel-mode driver framework (KMDF) is an infrastructure for developing kernel-mode drivers. It provides a C-language device driver interface (DDI) and can be used to create drivers for Microsoft§Windows§2000 and later releases. In essence, the framework is a skeletal device driver that can be customized for specific devices. KMDF implements code to handle common driver requirements. Drivers customize the framework by setting object properties, registering callbacks to be notified of important events, and including code only for features that are unique to

their device.

KMDF provides a well-defined object model and controls the lifetime of objects and memory allocations. Objects are organized hierarchically in a parent/child model, and important driver data structures are maintained by KMDF instead of by the driver. [5]

The Windows Driver Foundation (WDF) also includes a user-mode driver framework (UMDF). If your device does not handle interrupts, perform direct memory access (DMA), or require other kernel-mode resources such as non-paged pool memory, then one should consider writing a user-mode driver instead

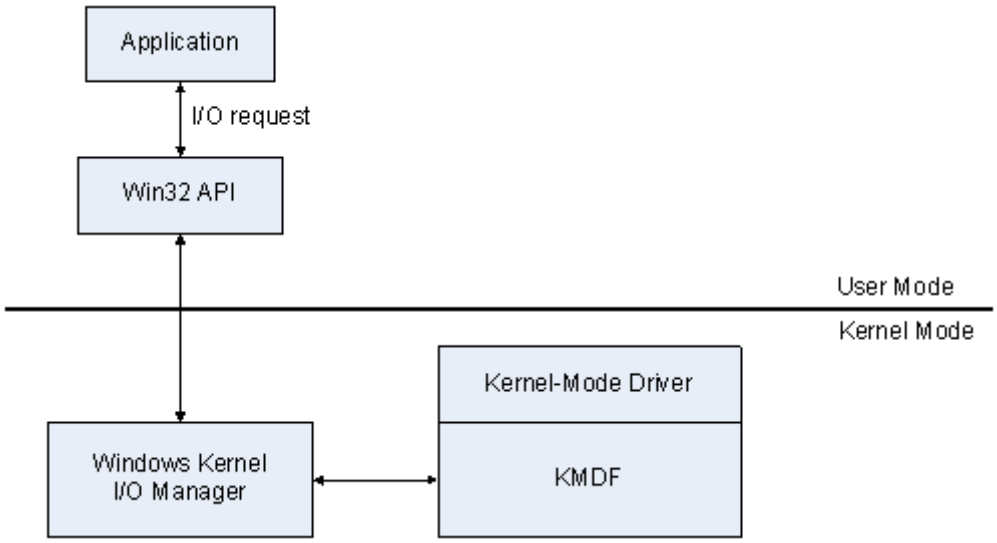

Figure 5.4: I/O Flow to Kernel-Mode WDF Driver [5]

KMDF Components KMDF is distributed as part of the Windows Driver Kit (WDK) and consists of header files, libraries, sample drivers, development tools, public debugging symbols, and tracing format files. By default, KMDF is installed in the WDF subdirectory of the WDK root installation directory. KMDF-based drivers are built in the WDK build environment. Table lists the KMDF components that are installed as part of WDF. [5]

Table 5.1: KMDF Component

Component	**Location**	**Description**
Header files	wdf/inc	Header files required to build KMDF drivers
Libraries	wdf/lib	Libraries for x86, x64, and Intel Itanium architectures
Sample drivers	wdf/src	Sample drivers for numerous device types; most are ported from Windows Driver Development Kit (DDK) WDM samples
Tools	wdf/bin	Tools for testing, debugging, and installing drivers; includes the redistributable KMDF co-installer, WdfCoinstaller*nn*.dll
Debugging symbols	wdf/symbols	Public symbol database (.pdb) files for KMDF libraries and co-installer for checked and free builds
Tracing format files	wdf/tracing	Trace format files for the trace messages generated by KMDF libraries and co-installer

To aid in debugging, KMDF is distributed with free and checked builds of the run-time libraries and loader, along with corresponding symbols.

Structure of a KMDF Driver A KMDF driver consists of a "DriverEntry" function that identifies the driver as based on KMDF, a set of callback functions that KMDF calls so that the driver can respond to events that affect its device, and other driver-specific utility functions. Nearly every KMDF driver must have the following:

- A **DriverEntry** function, which represents the driver's primary entry point.
- An **EvtDriverDeviceAdd** callback, which is called when the Plug and Play manager enumerates one of the driver's devices (not required for drivers that support non-Plug and Play devices).

- One or more **EvtIo*** callbacks, which handle specific types of I/O requests from a particular queue. Drivers typically create one or more queues into which KMDF places I/O requests for the driver's device. A driver can configure its queues by type of request and type of dispatching.

A minimal kernel-mode driver for a simple device might have these functions and nothing more. KMDF includes code to support default power management and Plug and Play operations, so drivers that do not manipulate physical hardware can omit most Plug and Play and power management code. If a driver can use the defaults, it does not require code for many common tasks, such as passing a power IRP down the device stack. The more device-specific features a device supports and the more functionality the driver provides, the more code the driver requires.

Device Objects Every driver creates one or more device objects, which represent the driver's roles in handling I/O requests and managing its device. KMDF supports the development of the following types of device objects.

- **Filter device objects** (filter DOs) represent the role of a filter driver. Filter DOs "filter," or modify, one or more types of I/O requests that are targeted at the device. Filter DOs are attached to the Plug and Play device stack.

- **Functional device objects** (FDOs) represent the role of a function driver, which is the primary driver for a device. FDOs are attached to the Plug and Play device stack.

- **Physical device objects** (PDOs) represent the role of the bus driver, which enumerates child devices. PDOs are attached to the Plug and Play device stack.

- **Control device objects** represent a legacy non–Plug and Play device or a control interface. They are not part of the Plug and Play device stack.

Chapter 6

Battery Charging Driver Flow

Battery management Driver is mini-class KMDF driver and it performs following functions:

a. Communicate with windows battery class driver Battc.sys.

b. Initialize all battery related hardware such as Battery charger, PMIC, Fuel Gauge IC.

c. Implement call back function for response of interrupt generated like charger insertion or removal, battery insertion and fault interrupts.

d. Battery capacity measurement at periodic interval.

e. Battery charging enabling and disabling functions.

f. Read Battery ID.

6.1 Interaction flow between windows battery class driver and BM mini-class driver

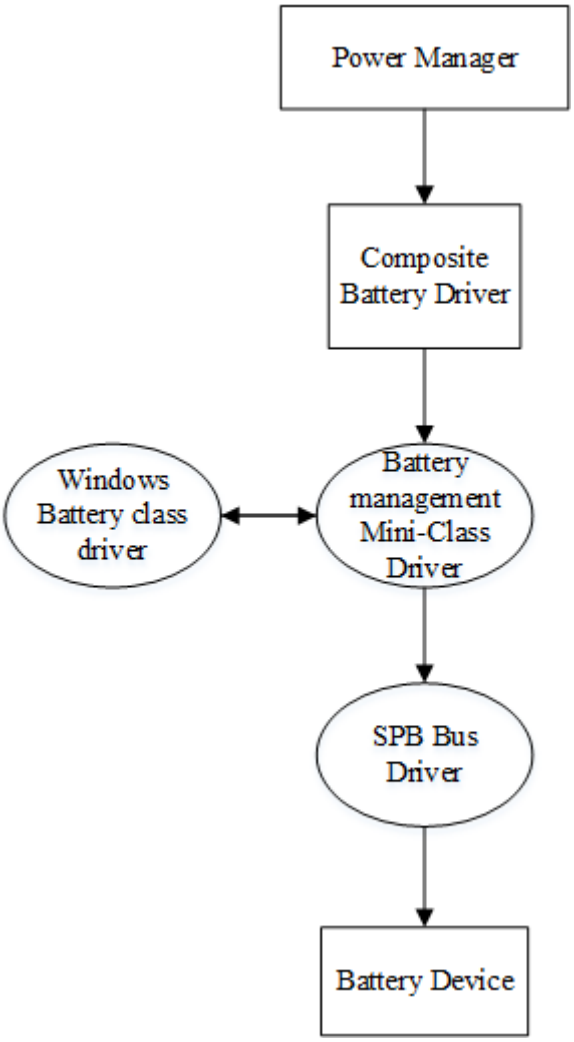

Figure 6.1: Basic Driver interection diagram

As shown in the preceding figure, the role of each component in battery operations is as follows:

a. Depending on system there can be Bus driver between Battery driver and mini class driver i.e. ACPI filter driver.

b. A **battery miniclass** (battery management) driver is the functional driver having manufacturer proprietary feature for Specific battery type. Which responsible for all battery control functions.

c. The **composite battery driver** supplied by Microsoft. It communicate to

power manager in response of IRPs sent by Power manager, It stores all battery related information such as SOC. It also notifies system in case of any change in battery charge.

d. The **battery class** supplied by Microsoft, and it support multiple battery mini-class driver, It allocates resource for miniclass driver, handles device IRPs and It calls BatMiniXX function for querying Battery related information.\

e. The **power manager** generates IRPs for PnP and Power to Driver stack, It does not communicate to mini-class or Device driver. It sends IRPs to composite driver which queries to mini-class.

f. The **battery GUI** Its interface where application layer software use it for retrieving any battery related information such as remaining time, current SOC with help of power management. It also can query Device driver directly for any specific battery information such as current battery voltage (uses Battery GUI for referring specific battery)

6.2 Battery management Driver Entry function

The miniclass driver's DriverEntry routine sets up the following driver-specific entry points:

a. The Unload routine in DriverObject->DriverUnload

b. The driver's AddDevice routine in DriverObject->DriverExtension->AddDevice

c. The DispatchPower routine in DriverObject->MajorFunction[IRP_MJ_POWER]

d. The DispatchPnP routine in DriverObject->MajorFunction[IRP_MJ_PNP]

e. The DispatchCreate routine in DriverObject->MajorFunction[IRP_MJ_CREATE]

f. The DispatchClose routine in DriverObject->MajorFunction[IRP_MJ_CLOSE]

g. The DispatchDeviceControl routine in DriverObject->MajorFunction[IRP_MJ_DEVICE_CON

h. The DispatchSystemControl routine in DriverObject->MajorFunction [IRP_MJ_SYSTEM_CC

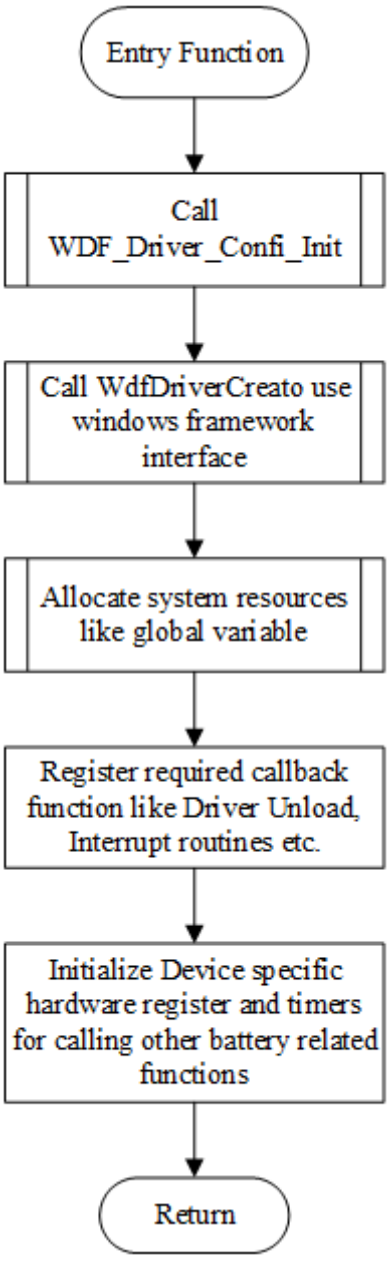

Figure 6.2: Entry function task

All of above is predefined standard windows structure for managing KMDF based driver. Figure 6.2 shows some BM driver entry function tasks

6.3 Windows battery management Driver Task

As listed in starting of this section battery management diver responsible to battery charging and reading battery parameters like SOC, Battery ID. figure 6.3 shows high layer task flow diagram of BM driver for which corresponding functions is implemented in C language. Some of the function called regularly by use of time created at time of driver entry and many of function implemented as callback function.

Battery management not responsible for controlling power drawn by hardware of handheld device, it only take part in switching system power source between battery and charger power.

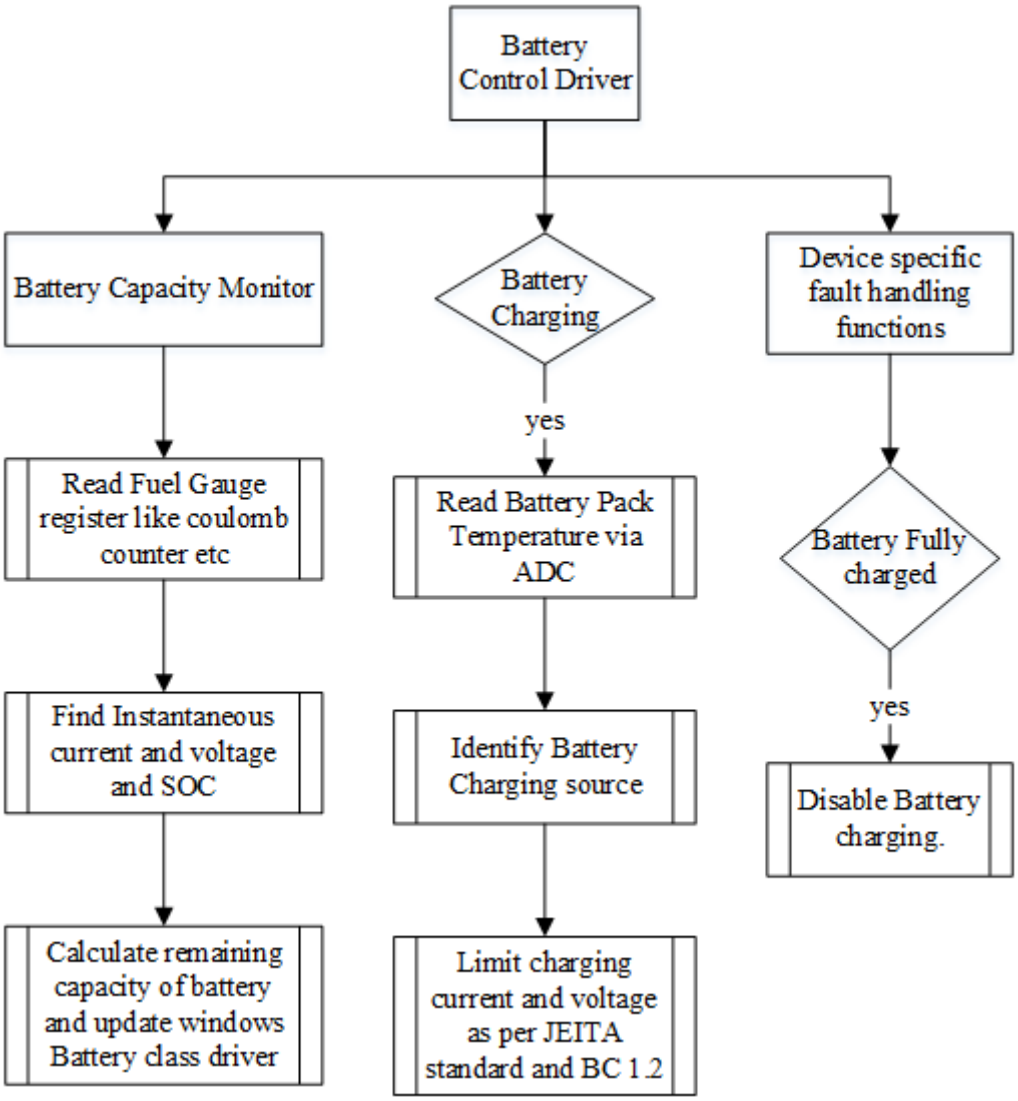

Figure 6.3: Battery management Driver functional flowchart

Chapter 7

Camera Sensor Driver Flow

Camera Driver development process include developing display and camera driver, trough DirectShow and camera application. Camera application use middleware layer of DirectShow video capture for control and interacting with Camera Hardware, camera application is not intended to communicate directly to Camera hardware.

Camera could be used for multiple Application like

a. Still Image Capture

b. Video Capture

c. Processing of frames and identifying something

d. Compressing and transmitting over the network

Various steps of Image Processing done on frames captured by camera sensor before passing those frames to application running on user level.
Controlling the flow of data from a camera device requires working with three kinds of objects: camera adapters, pins, and streams.

7.1 Elements of camera Systems

a. Device driver

The layer of software that controls the physical camera hardware. A single camera driver can control multiple cameras.

b. Device instance

A single complete physical set of camera hardware. A camera device driver can support multiple device instances.

c. Adapter

The logical representation of a device instance in software.

d. Pin

The camera driver architecture uses the concept of a pin as defined by the DirectShow middleware. For more information, see Pins. Pins are used to transmit data in and out of a device and are always in one of three states: stopped, paused, and playing. A camera driver can support up to three types of pins: preview, capture, still. Each pin can support a number of media formats. The DirectShow middleware manages the process matching media formats from camera output pins with pins on downstream filters. For more information, see Video Format Negotiation.

e. Pin driver

Software used to control a pin. A pin driver can support multiple pins.

f. Pin instance

A single pin on an adapter.

g. Pin handle

A unique identifier that provides a means of locating a single specific pin.

h. Stream

Data that flows out of an adapter's pin and to the application.

7.2 Driver Software Architecture

The camera sensor driver is responsible for controlling the specific camera sensor on Tablet platform. It implements a set of common control functionalities expected by the ISP driver for the specific camera sensor like Set/Get Resolution, Set/Get Focus Position, Set/Get Exposure etc, which are fundamental functions for camera preview, camera take-photo, camera video record and 3A control. The sensor driver is a KMDF driver that exports interfaces only to the ISP driver and access from non-privileged user applications are not allowed.

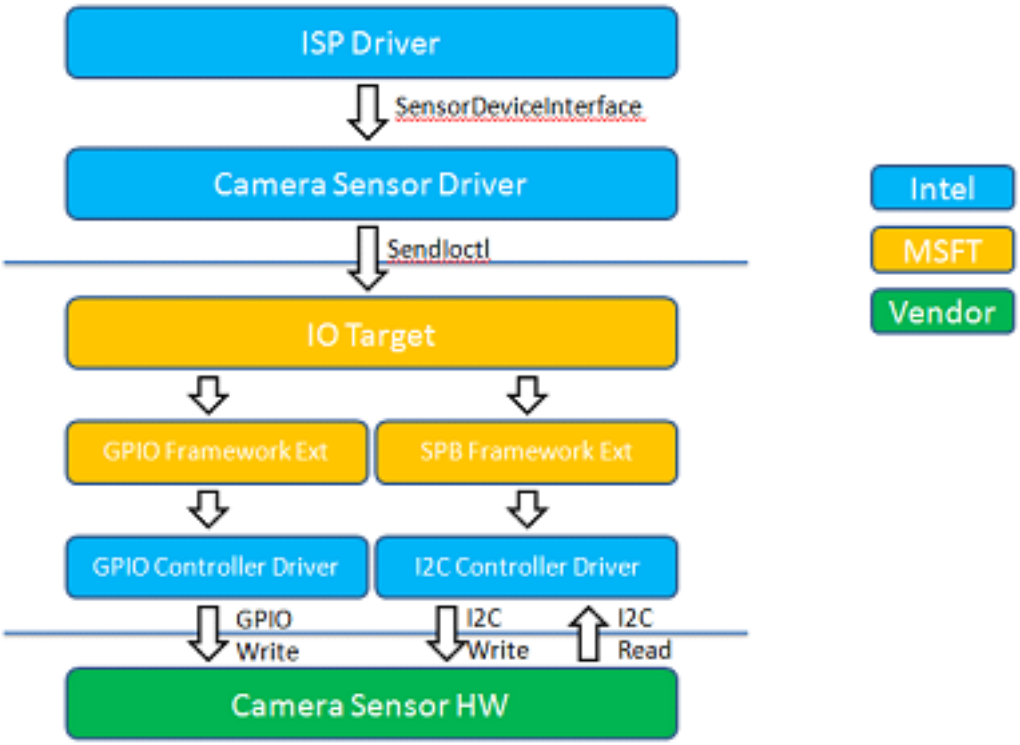

Figure 7.1: Camera Sensor Driver

Driver MFT let device manufacture to apply custom effects on Camera Source stream.

The Windows Store device app of camera runs as separate process than the Windows Store app that invokes it from the ***CameraCaptureUI*** API. Specific predefined sequence of events must occur for the Windows Store device app to control a driver MFT.

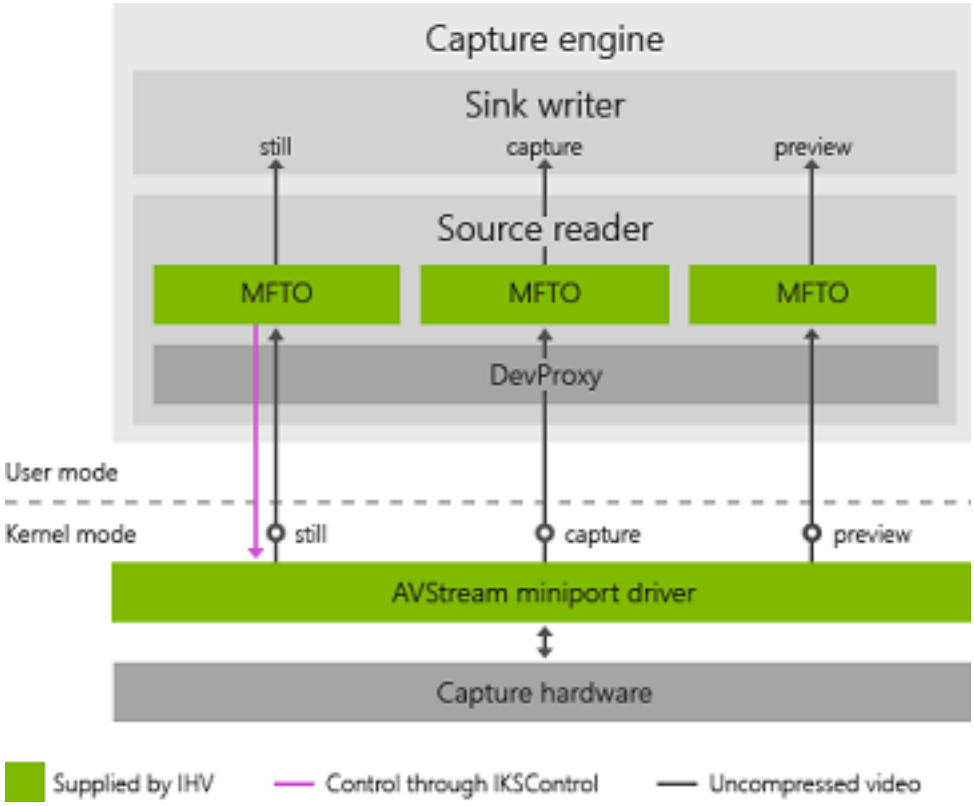

Figure 7.2: Three Pin Camera Capture Engine

a. A Windows Store app wants to capture a photo, so it calls the ***CaptureFileAsync*** method

b. Windows requests the driver MFT pointer and the camera's device ID

c. The driver MFT pointer is passed to a settings host

d. The host queries device properties for the app ID of the Windows Store device app associated with the camera (per device metadata)

e. If no Windows Store device app is found, the default flyout interacts with the capture engine

f. If a Windows Store device app is found, it is activated and the settings host passes the driver MFT pointer to it

g. The Windows Store device app controls the driver MFT using the interface exposed through the pointer

7.3 Camera driver functional requirement

7.3.1 Live viewfinder

The Camera driver should generate continuous video stream with same orientation and aspect ratio as selected mode of capture, which can be used as a Preview Pin (Live viewfinder)

Latency requirements Camera driver should produce preview stream within 500ms once camera application sent command to launch camera. After launching camera, the driver should default in still capture mode with rear camera and can be changed to other mode or Front camera as selection by user with in 500ms or lesser.

Resolution and aspect ratio requirements The preview stream has same orientation as selected camera capture mode. The driver should have preview resolution same as aspect ratio and matches the device display resolution on at least one dimension. Capture resolution having smaller resolution than display in both dimension, preview resolution shell be same as the resolution of capture mode. Preview aspect ratio allowed to vary within 1% of capture aspect ratio.

Examples of various resolution are as follows:

For an 800x480 (WVGA) device, the driver must provide the following preview resolutions:

a. 640 x 480 for 4:3 capture resolutions

b. 800 x 450 for 16:9 capture resolutions

c. 720 x 480 for 3:2 capture resolutions

d. 800 x 480 for 5:3 capture resolutions

For a 1280 x 720 (HD/ WXGA) device, the driver must provide the following preview resolutions:

a. 960 x 720 for 4:3 capture resolutions

b. 1280 x 720 for 16:9 capture resolutions

c. 1080 x 720 for 3:2 capture resolutions

d. 1200 x 720 for 5:3 capture resolutions

Data output format Driver shell produce preview stream in NV12 format and associated with metadata with preview stream if it has enabled in driver.

7.3.2 Still image capture

Driver shell enable still capture mode in following scenario:

Single shot On issue of capture command driver shell capture image and it shall be produced in a standard NV12 format along with metadata associated with the capture.

Captured image The driver should be capable of capturing still images having pre-specified resolutions. The list of supported resolutions should be provided by the driver and which can be used by capture applications to select an appropriate value.

The driver have to support at least following aspect ratios for still capture for rear camera:

a. 4:3

b. 16:9

The 4:3 aspect ratio is optional for the front camera.

Confirmation image It can be possible for an applications to select property for the driver to return a confirmation image after still capture. The confirmation image may be of a different resolution compared to capture resolution, and can be have lower quality. The confirmation image will have same aspect ratio and orientation compared to the captured image.

When this property is enabled, the driver have to provide confirmation image within 500ms and can have low resolution of the actual capture. There may have associated metadata with the confirmation image.

Still capture metadata The driver have to provide metadata associated with still capture as per EXIF 2.2 specification, along with the maker notes which used for image quality tuning. This has to be provided with the captured image stream.

Along with addition to capture metadata, driver can provide the current focus state, ROI (Region of Interest) information, and other custom metadata with the confirmation image. This metadata will not available if confirmation Image is turned off by application

The driver has to provide the focus state metadata with preview stream whenever preview stream is available.

Photo sequence capture The driver captures images in continuous sequence at specific resolution and stores them in buffers and discarding images when memory constraints requires which allows the applications to retrieve images captured

before and after the time of capture triggered by the user. The driver shall support an ability for the application to retrieve frame captured closest to the trigger, and specific number of frames before the trigger or past frames and another specific number of frames after the trigger or future frames. The driver has to allow application to query for the maximum number of past frames that can be retrieved as per hardware constraints.

Warm start In warm start on receiving capture command driver has to reconfigure all camera sensor parameter, which causes additional latency. And it is requires that driver shell image frames with in two frames latency.

Infinite burst capture The driver continuously produce images with specified resolution.

Frame rate requirements The driver has to maintain frame rate of 15 fps or higher during single shot still capture, at the camera sensor. Microsoft recommend a 30 fps frame rate, and the camera driver can maintain this frame rate as long as exposure algorithm will meet this requirement without degradation of the camera image quality.

The driver shall allow the application to query and retrieve the current sensor frame rate. The driver shall also support a query for the maximum frame rate possible for the preview and capture streams at any supported capture resolution.

Latency requirements The driver shall support the following latency requirements:

a. Shutter lag for single shot capture

b. Shot to shot latency

c. Minimum burst capture rate

Time stamp The time stamp corresponding to the start of the frame scan shall be the frame time stamp (not the time stamp corresponding to the instant the last row of pixels is read out). This shall be represented in units of 100 nanoseconds.

Variable photo sequence The driver shall be able to capture consecutive images with varying capture parameters as instructed by the application. The driver shall be able to preprogram the number of frames needed and set independent capture parameters for each frame before capture is initiated. Once the capture command is issued, the frames are captured in succession with as little latency as possible.

7.3.3 Video recording mode

Resolution for capture The camera driver shall support the following resolutions for video recording:

a. QCIF

b. CIF

c. VGA

d. WVGA

e. 720x1280

f. 1080x1920

It shall be possible to record video at 30 fps at all specified resolutions. Microsoft recommends that the driver support the maximum frame rate supported by a given hardware configuration for video recording so as to enable features such as slow motion capture.

Switching between modes It shall be possible to switch from still capture mode to video recording mode with a latency of under two frames.

7.3.4 Focus requirements

The camera driver shall provide a means to focus the scene correctly for all the driver modes specified in the previous sections. The driver shall support the following focus modes.

Focus modes

a. **Single auto focus** - In this mode, the camera is launched with the lens set to a default focus position such as infinity or hyperfocal. When focus is initiated, the driver selects the most optimal lens position. This lens position is maintained till another command to initiate focus is received.

   Typically, the driver shall span the entire focus range to determine the best focus position. If a faster focus algorithm is supported by the driver, there shall be a custom property to enable a faster focus.

b. **Continuous auto focus** - In this mode, the camera is launched and the focus initiated. After the focus converges, the driver monitors the focus statistics, and reinitiates focus if the statistics change beyond a preset threshold. This process keeps repeating and the camera is maintained in a state of optimal focus continuously.

c. **Manual focus** - In this mode, the camera is launched with the lens set to a default focus position such as infinity or hyperfocal. An application can instruct the driver to set the lens position to the desired focus using the ISP controls interface.

d. **Fixed focus** - If the underlying camera hardware does not support a focus mechanism, the driver shall advertise this, and ignore any auto focus related

commands.

e. **Default focus mode** - If continuous auto focus is supported by the hardware, the driver shall set that as the default mode. If continuous auto focus is not supported, the default shall be set to single auto focus. In the case where auto focus is not supported by the hardware, the focus shall default to the fixed focus mode.

Focus weighting

a. **Center weighted** - The driver shall weight the focus statistics from the center of the frame higher than the rest of the frame so as to attain best focus in the center of the frame.

b. **Region of interest** - The driver shall attempt to attain the best focus in a region of interest (ROI) inside the frame as specified by the application. In the AF-C mode, the driver shall ignore any specified ROI and continue to focus on the center of the frame.

c. **Object tracking** – This is optional. In this mode, the driver shall focus on a specified object in the frame and keep the object in focus, as the object or camera moves.

Focus range To improve the efficiency of auto focus searching, the driver shall support the setting the focus search modes. The focus range is defined as the number of steps defined by integers.

a. **Macro** - In the macro mode, the driver shall start the focus sweep from the first step in the focus range and end at a pre-defined focus step, preferably close to the middle of the range.

b. **Normal** - In the normal mode, the driver shall start the focus sweep from a predefined position between the first and the last steps to the last step in the

focus range in search for the best focus position. Microsoft recommends using a step close to the midpoint.

c. **Full range** - In the full range mode, the driver shall step through the entire focus range from minimum to maximum to search for best focus position. The algorithm should be optimized to converge with the fewest number of steps possible.

d. **Default focus range** – The driver shall set the default focus range to normal to optimize capture latency.

Pre-defined focus positions

a. **Nearest focus distance** – The nearest distance that the camera can successfully focus upon.

b. **Hyperfocal** – The focus position where the best focus is at the hyperfocal position.

c. **Infinity** – The focus position where infinity is in the best focus.

Focus lock and default positions The driver shall support the ability to lock focus at any given focus step. The focus shall remain locked in this position until a command to unlock focus is received. The focus lock mechanism shall be supported for both AF-C and AF-S.

If autofocus fails, the driver shall provide a default focus position. Microsoft recommends that this be set to hyperfocal.

Prioritizing focus versus capture

a. **Capture priority** - In this mode, when a capture command is issued by the application while the driver is seeking focus, the focus sweep is canceled. The driver can then do the following to capture the image as fast as possible.

(1) If in the AF-C mode, capture the image immediately.

(2) If in the AF-S mode, move to the hyperfocal position and capture the image as quickly as possible.

b. **Focus priority** - In the focus priority mode, when a capture command is issued by the application while the driver is seeking focus, the driver shall wait for focus to complete before the image is captured. In the event that the focus fails, the driver shall move to the hyperfocal position and capture the image.

c. **Default priority** – The driver shall default to the capture priority mode to optimize capture latency.

Focus tuning

a. **Still capture mode** – During still capture, the continuous auto focus algorithm shall converge as quickly as possible to the new focus position. The focus range shall be set to full range unless specifically instructed by the application. The driver shall attempt to move to the best focus position from the current lens position in as few steps as possible.

b. **Video capture mode** – In the video capture mode, constant fluctuations in focus shall be avoided. The driver shall provide a tuning parameter to limit rapid changes in focus. Microsoft recommends that the nearest focus distance for video be set greater than the nearest focus distance for still capture to prevent abrupt focus changes. However, this shall be a tunable parameter.

c. **Switching between modes** – Whenever the camera switches from still capture mode to the video recording mode or vice versa, the focus shall be reset, and restart in the default mode specified below:

(1) Single still capture: AF-S

(2) Photo sequence still capture: AF-C

(3) Video recording mode: AF-C

If AF-C is not supported by the underlying hardware, the focus shall default to manual for photo sequence and video recording with the lens position set to hyperfocal.

Focus bracketing For variable photo sequence, the camera driver shall support focus bracketing, with a minimum requirement of macro, best focus and infinity.

Focus states The driver shall support the following focus states, and notify the application when the focus state changes. The current focus state shall be always made available during continuous auto focus as metadata. This may be optionally made available during AF-S.

a. Uninitialized

b. Lost

c. Searching

d. Focused

e. Failed

7.3.5 Exposure requirements

The driver shall support the following exposure modes.

Auto exposure

a. **Center weighted** – The exposure statistics shall be weighted to expose the objects in the center of the frame correctly.

b. **Matrix** – The exposure statistics shall be weighted uniformly across the frame.

c. **Spot** – This is the same as center ROI. The exposure shall be computed based on the center of the frame only, using the smallest element of the statistics grid at the center.

Manual exposure All the exposure parameters, including integration time, frame rate, ISO and aperture are specified by the application. If any parameter is not specified, the last computed value for that setting shall be used.

ROI-based exposure In this mode, the driver shall optimize the exposure for a specific region of interest (ROI). This shall be:

a. **Face detection** – If the driver supports face detection, exposure can be optimized for a region containing the face. The ROI shall indicate that a face has been detected with a computed confidence level. The auto exposure algorithm shall use this information as input to compute the exposure for the face correctly.

b. **User-specified ROI** – The driver optimizes the exposure for an ROI specified by the application. The ROI shall vary in size from a single pixel to the entire frame as described in the section on ROI specifications.

Auto exposure for still capture

a. **Maximum exposure time** – The maximum exposure time set by the auto exposure algorithm for capture in low light shall not exceed 66 ms. Microsoft recommends that the ISO value be increased prior to increasing the exposure time beyond 33 ms.

b. **Frame rate** – During still capture, the viewfinder shall support a frame rate of 30 fps as long as the exposure conditions allow. In low light situations, the

driver shall reduce the frame rate to increase brightness. The frame rate shall not fall below 15 fps.

c. **AE convergence latency** – During still capture, the auto exposure algorithm shall converge as quickly as possible to the correct setting. Microsoft recommends using a single frame for AE convergence latency.

Auto exposure for video recording

a. **AE convergence** – The driver shall avoid sharp variations in exposure. It shall be possible to specify the rate of convergence for AE as a tuning parameter.

b. **Frame rate** – During video capture, the driver shall maintain the constant frame rate as specified by the application.

Exposure API

a. **Integration time** - The driver shall allow the application to specify the pixel integration time in milliseconds in the manual mode. In the auto exposure mode, if the application specifies this parameter, the driver shall compute the other exposure parameters such as ISO and frame rate to keep the integration time as close to the specified value as possible.

b. **Frame rate** - The driver shall allow the application to specify the frame rate for capture.

c. **ISO** - The driver shall allow an application to set a specific ISO value for the capture as specified by the ISO 12232 standard – saturation based ISO value. The driver shall advertise the minimum and maximum values of the ISO range that is supported, and allow the application to set the ISO to any integer value in that range.

In the auto exposure mode, the driver shall determine the appropriate ISO value to be used if the application does not specify an ISO value. If an ISO value is specified, the driver shall attempt to set the ISO as close to the specified value as possible while computing the exposure parameters.

d. **Aperture** – This is optional. If the underlying camera hardware supports variable aperture, the driver shall advertise the allowed values and allow applications to specify the aperture.

e. **Exposure bracketing** - The driver shall allow an application to enable exposure compensation based on the exposure parameters computed by the auto-exposure control. The exposure compensation shall be specified as a fraction of the Exposure Value (EV). Microsoft recommends a step size of one sixth of an EV.

   The driver shall advertise the minimum and maximum exposure compensation possible. Microsoft recommends the minimum and maximum values are 3EV above and below the exposure value computed by the auto exposure control.

   While compensating the exposure, the driver shall attempt to remain within the recommended exposure time. For example, when the pixel integration time is at 66 ms, to increase exposure value, the driver shall first attempt to increase the ISO before increasing the integration time. Microsoft recommends that the shortest possible exposure time is used, taking motion blur and image quality parameters such as signal to noise ratio into consideration.

   It shall be possible for an application to bracket ISO and exposure time separately in increments of a sixth of an EV.

f. **Anti-banding** - The driver shall support the ability to set exposure parameters in a way that fluorescent light flicker will not lead to banding in the image. It is not expected that anti-banding is supported for taking images of electronic displays. Anti-banding shall be implemented in the following modes:

(1) Auto flicker detection – This feature is optional. The driver shall provide an automatic flicker detection algorithm to detect powerline frequency flicker and set the exposure appropriately.

(2) Manual anti-banding – The driver shall support manually setting the flicker frequency control to AE.

(3) Powerline frequency detection – The driver shall support identifying the powerline frequency detection based on location information to set the flicker frequency correction.

7.3.6 White balance requirements

The driver shall support the following white balance modes.

Auto white balance The driver shall automatically determine the scene illuminant and render neutral colors (whites, grays, blacks) as accurately as possible. The other colors shall be rendered in a pleasing manner without noticeable color bias. The following auto modes shall be supported.

a. **Scene-based white balance** - The driver detects the illuminant using statistics collected from the entire scene. The illuminant detection shall be correct in the presence of solid color backgrounds. Skin tones in a color checker chart shall be rendered with an error of less than 8 Delta E and the neutral patches with an error of less than 5 Delta E.

b. **Face-based white balance** - If face detection is available and a face is detected in the scene, the driver shall have the ability to use the facial information such as the white of the eye to set the white balance. The driver shall use the information contained in the face ROI to enable face based white balance.

c. **ROI-based white balance** - The white balance is computed based on the color statistics collected from the specified ROI.

Manual white balance The driver shall allow the application to set the white balance to a specific color temperature between 1800 K and 10000 K. In addition, the following white balance modes as specified by the color temperature shall be supported:

a. Horizon: 2200 K

b. Incandescent: 3050 K

c. Fluorescent: 4300 K

d. Daylight: 6500 K

e. Shade: 7500 K

White balance lock The driver shall support the ability to lock the white balance at any given instant. To unlock the white balance, the application shall instruct the driver to switch to either the auto or manual mode. Note that in the manual mode, the white balance stays locked at the specified color temperature.

Capture mode requirements The viewfinder shall maintain colors as close as possible to the captured image/video. In the still capture mode, the white balance is expected to converge as quickly as possible. Microsoft recommends a latency of a single frame.

During video recording, the rate of convergence of the white balance algorithm shall be set based on a tuning parameter. Microsoft recommends avoiding sharp fluctuations in white balance during video recording when brightly colored objects are in the frame for brief intervals.

In variable photo sequence mode, the white balance shall be locked at the first frame unless otherwise instructed by the application.

7.3.7 Region of interest (ROI) specifications

The driver shall support the interface to specify a region of interest (ROI) within the scene.

ROI definition The driver shall support the ability to specify a region of interest using the top left corner and right bottom corner of the bounding region. These coordinates must be normalized with respect to the maximum camera resolution, where (0,0) is the top left and (1,1) is the bottom right coordinates of the entire frame.

In addition to specifying the coordinates of the ROI, it shall be possible to specify if the ROI shall be used to compute each of the following:

a. Auto focus

b. Auto exposure

c. Auto white balance

The ROI shall contain a property that indicates whether a face has been detected inside it. It shall be possible to specify a confidence level from 0-100 to set the weight with which the ROI statistics must be used. The default value for the confidence level shall be set to 100, indicating ROI only statistics.

Minimum ROI size It should be possible to specify a single point of interest, and the driver can use the minimum possible ROI permitted by the underlying hardware to specify the ROI.

Multiple ROI scenarios In the case of multiple ROIs, the last set ROI shall override the previous ones. The driver shall not support optimization for multiple ROIs. That shall be the responsibility of the application.

7.3.8 Zoom specifications

Optical zoom support Optical zoom is optional. If implemented, this shall be internal to the driver. The driver shall first increment the optical zoom to the maximum possible value before incrementing the digital zoom.

Digital zoom during still capture It shall be possible for the application to send down zoom values in small increments so that the visual zooming looks smooth on the viewfinder. After the final zoom value is received by the driver, the driver shall converge to that zoom value in less than 5 frames.

Digital zoom during video capture During video recording, the zoom implementation shall be similar to the still capture mode described above. If the cropped sensor resolution falls below the video resolution required by the application, further zooming shall be disabled.

7.3.9 Camera flash requirements

Flash off The flash is switched off during still image capture and video recording.

Flash on The flash is switched on during still image capture and video recording.

Flash on adjustable power The flash is switched on during still image capture and video recording at a power level specified by the application.

Flash auto The flash is switched on during still image capture only if the auto exposure algorithm determines that additional illumination is needed in the scene. The exposure parameters should be recomputed to account for the flash.

Flash auto adjustable power The flash is switched on during still image capture only if the Auto Exposure algorithm determines that additional illumination

is needed in the scene. The power level of the flash is specified by the application, and the exposure parameters should be recomputed based on this.

Focus assist illumination In low light situations where the AF algorithm has difficulty finding the best focus, the flash is used for additional illumination. The driver shall support a property that would enable an application to turn focus assist on and off even in the flash off mode.

Red eye reduction A pre-flash is fired before the actual capture to provide red eye reduction. The driver shall support a property to turn this on and off.

Flash during photo sequence In the photo sequence mode, the flash must be fired only on a single frame as the default. The metadata should indicate whether a flash was fired on not on each frame.

If the driver supports multi-flash capabilities, the flash can be triggered on every sequential frame capture after the trigger is received. The metadata should indicate whether a flash was fired or not on each frame. The default flash mode shall be single flash, unless the application specifically requests multiple flashes.

7.3.10 Hardware calibration support

The driver shall support characterizing each camera module for the following properties and use the generated calibration data in image quality tuning.

a. Noise profile

b. Color spectral response

c. Lens shading profile

d. Defective pixel mapping

The calibration data shall be generated at the manufacturing facility, and stored either in a non-volatile memory location in the camera module or in the camera phone file system. On camera launch, the driver shall load the calibration data and verify that it is valid by matching the camera module ID. If valid, the driver shall use the calibration data to modify the various algorithms to improve image quality. If the calibration data is not found to be valid, or if calibration data is not available, the driver shall use a default calibration file in lieu of the device calibration file.

7.3.11 Noise and sharpness control

The driver shall provide a means for noise filtering prior to demosaic in the RAW Bayer domain, as well as after demosaic. After demosaic, it shall be possible to apply noise filtering and sharpness enhancement algorithms separately in the luminance and chrominance channels. Microsoft recommends that the driver support advanced noise reduction algorithms such as those based on wavelet theory.

It shall be possible to set the strength of each filter as a tuning parameter during camera tuning.

7.3.12 Scene mode specifications

It shall be possible for an application to request the driver to bias various driver settings for specific scene mode captures. Microsoft recommends supporting the following scene modes:

a. Auto

b. Macro

c. Portrait

d. Sport

e. Snow

f. Night

g. Beach

h. Sunset

i. Candlelight

j. Landscape

k. Night portrait

l. Backlit

m. Manual

Chapter 8

Conclusion

The developed battery charging control driver are efficient in term of reduction in battery charging time and it compliance with Battery specification 1.2 which insure that system will not overload charging source.

The battery charging driver also compliant with JEITA standard which reduces risk of damaging battery while charging in all type of environment.

Bibliography

[1] "Battery Charging specification 1.2," USB Implementers Forum, 2010.

[2] "Windows Driver Framework," Microsoft, [Online]. Available: http://msdn.microsoft.com/en-us/library/windows/hardware/ff557565(v=vs.85).aspx. [Accessed 30 november 2014].

[3] W. Oney, "Programming the Microsoft®Windows®Driver Model.," in Microsoft Press, Redmon, Washington, 1999.

[4] Microsoft, "Architecture of the windows driver foundation," White Paper, may 10,2006.

[5] Microsoft, "Architecture of Kernel-Mode Driver Framework," White Paper, September 12, 2006.

[6] J. Qian, "Li-ion battery-charger solutions for," Texas Instruments Incorporated, 2010.