ADPD143
Linux/Android Device Driver 1.0
USER Guide

Copyright Information

Disclaimer

Trademark and Service Mark Notice

Preface

Thank you for using ADPD143 product from Analog Devices, Inc.
This is ADPD143 driver implementation for Measuring Heart rate for Linux/Android platform.

Purpose of This Manual

The *ADPD143 Linux/Android Device Driver User Guide* document provides information about the design details of the driver implementation for ADPD143 – HRM measuring applications.

Revision History

Date (YYYY-MM-DD)	Author	Notes
2018-03-09	Joshua Yoon	Initial version

Table of Contents

1 Hardware Details

This section provides the hardware interface details for ADPD143 with the host board.

The main components of the hardware are,

- Host Board
- ADPD143 Hardware
- Windows/Linux host machine

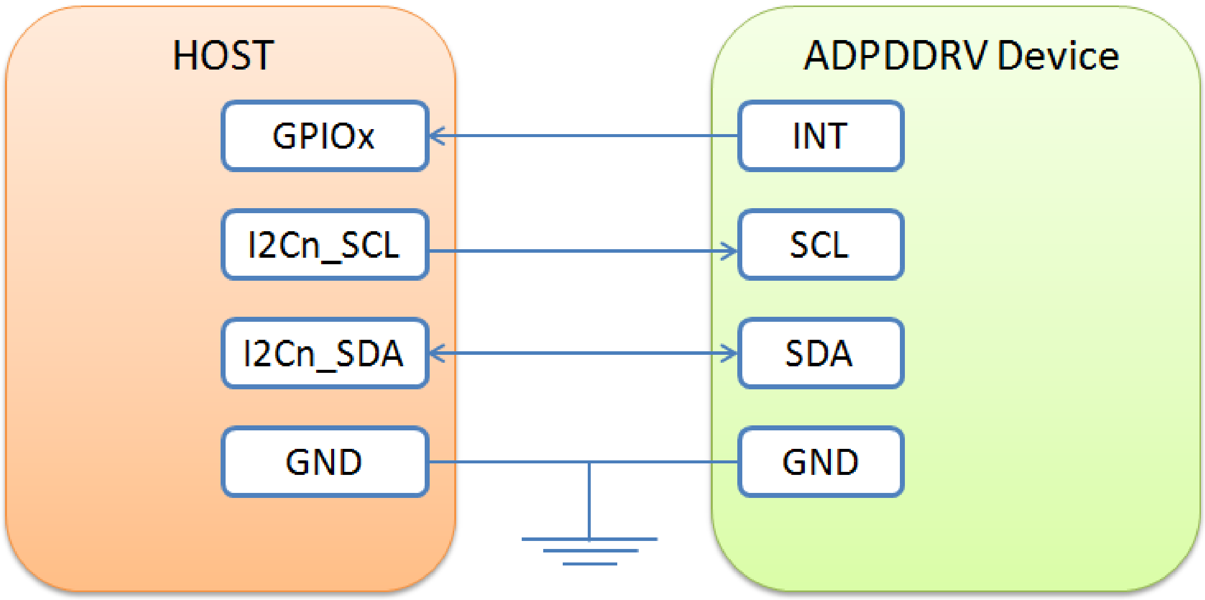

Figure 1 HW Interface Diagram

Detailed description of the hardware connection is explained below

- I2C interface is utilized for configuring ADPD143 Chip.
- Host Board will work as Master and ADPD143 will work as Slave.
- I2C Operating Clock Speed between Host & ADPD143 chip is 400 KHz.
- Hardware is connected through INT () , SCL, SDA.

- INT ( ) pin Generate Interrupts as per Interrupt register settings. Active high interrupt is used in ADPD143.
- ADPD143 supports I2C 8 bit addressing and 16 bit data.
- The chip address of ADPD143 is currently assumed as 0x64.

2 Software Details

This section provides information on the various software components used for the development and validation of ADPD143 driver.

- Linux Kernel version 3.4
- ubuntu 10.04 & above

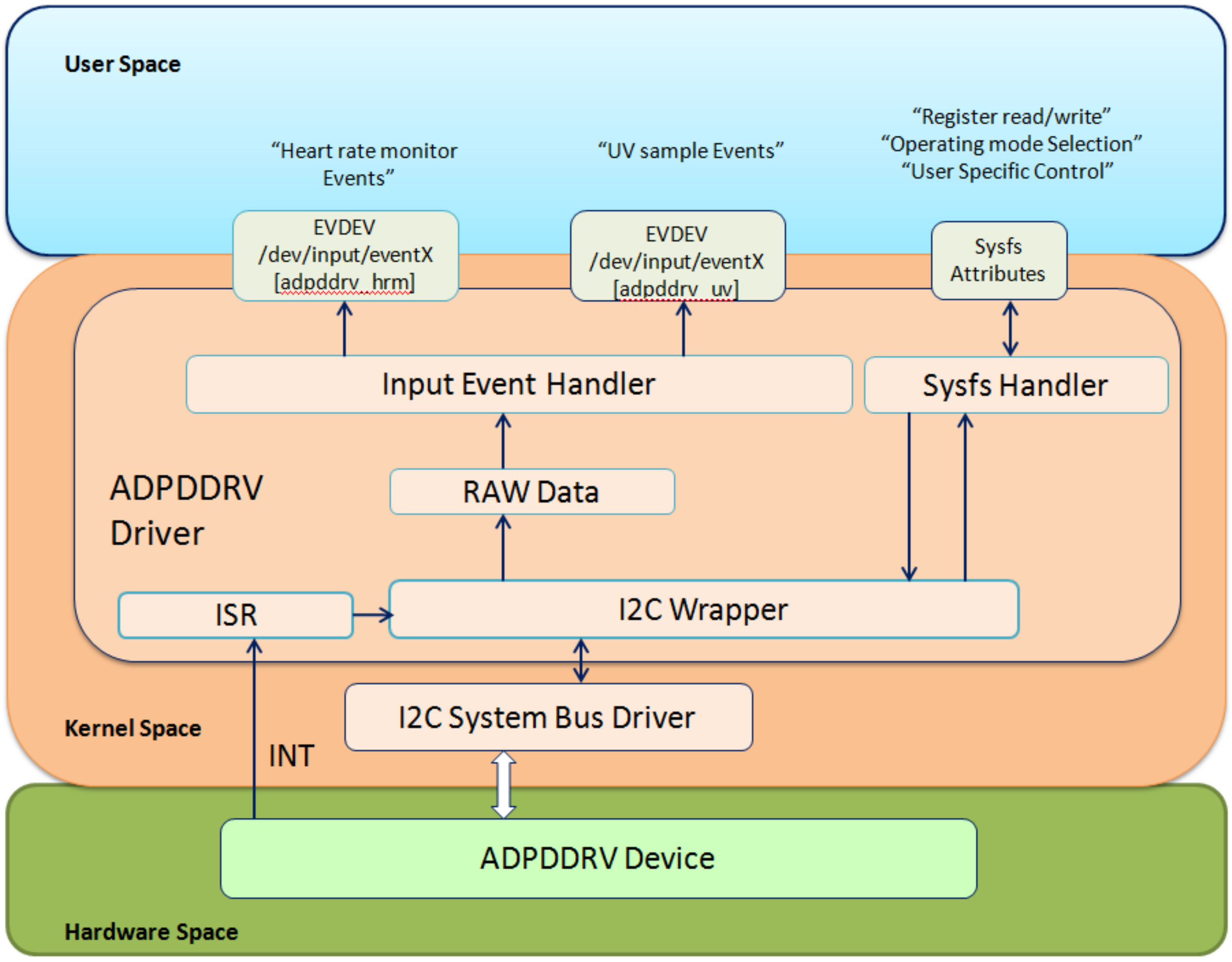

Figure 2 : Software Architecture diagram

- The device is registered as an input device (which would be identified as one of the event devices from user space /dev/input/eventX – X may be 0,1,…31.)
- Driver communicates with the device via I2C for register read/write.
- For every interrupt, the data for the respective modes will be packed and sent to user space via input event interface.
- Host board is connected to the ADPD143 chip through I2C Interface. The main software components of ADPD143 driver are I2C low level driver, Interrupt service routine, Input Event Handling and SYSFS handling.
- Interrupt Service routine is used to handle the interrupt from the ADPD143 chip. Here for every interrupt, driver communicates with device to get information on sensor and reports back to user space via Event interface.
- Event interface reports data to user space via normal Linux Events. Here the device will be registered as an input device to support event interface and it will be identified as "/dev/input/eventX" where 'X' can be 0,1,…31 which describes the number of the event device to which the chip has been associated. ADPD143 supports HRM related functionalities.
- SYSFS handling is available to access from the User Space. The various sysfs functionalities available are explained in a separate section.

2.1 *Software Architecture details*

ADPD143 is a sensor that offers Heart rate measurement functions. Depending upon the operating mode selection, the driver module receives data from ADPD143 chip through I2C. The following are the mode description in general.

Application	Operating Mode		INPUT_EVENT_NAME			Actual Device Parameter being sent from the ADPD143
	NAME	SYSFS VALUE	NAME	TYPE	CODE	
	IDLE_OFF	0x0	NIL	NIL	NIL	None
Heart Rate Monitor	Red (Slot A)	0x30	ADPD143	EV_REL	REL_X	X1_a + X2_a + Y1_a + Y2_a
				EV_MSC	MSC_RAW	X1 Value
					MSC_RAW	X2 Value
					MSC_RAW	Y1 Value
					MSC_RAW	Y2 Value
	IR (Slot B)	0x32	ADPD143	EV_REL	REL_Y	X1_b + X2_b + Y1_b + Y2_b
				EV_MSC	MSC_RAW	X1 Value
					MSC_RAW	X2 Value
					MSC_RAW	Y1 Value
					MSC_RAW	Y2 Value
	RI (Slot AB)	0x31	ADPD143	EV_REL	REL_X	X1_a + X2_a + Y1_a + Y2_a
				EV_MSC	MSC_RAW	X1 Value
					MSC_RAW	X2 Value
					MSC_RAW	Y1 Value
					MSC_RAW	Y2 Value
				EV_REL	REL_Y	X1_b + X2_b + Y1_b + Y2_b
				EV_MSC	MSC_RAW	X1 Value
					MSC_RAW	X2 Value
					MSC_RAW	Y1 Value
					MSC_RAW	Y2 Value

The ADPD143 device is registered with input devices and would be identified with one of the event devices from the user space. For e.g., /dev/input/eventX where X is the number of the event device to which the chip has been associated. 'X' can be 0, 1,31. Through event interface the data will be returned in the form of Linux events to user space. In ADPD143 one Input device named ADPD143 is present in the system which would be recognized as

/dev/input/event**A**,

– say **eventA** for **HRM input**

The driver communicates with the device via I2C for read/write registers. Based on the selection of the operating mode interrupts are triggered.

For every interrupt, the sensor information data will be packed and will be reported to the user space via Input Event interface through Input core driver. The ADPD143 driver reports input event data to the user space via various combinations of event type and code based on the operating mode selection.

In event interface, timestamp will be provided by EVDEV driver and therefore sensor information alone will be sent to user space from driver.

2.2 Algorithm Description

This algorithm has three major components: This section describes the same in separate flowcharts.

1. Initialization routine
2. Sequence flow for the Interrupt Service Routine [ISR]
3. Operating Modes

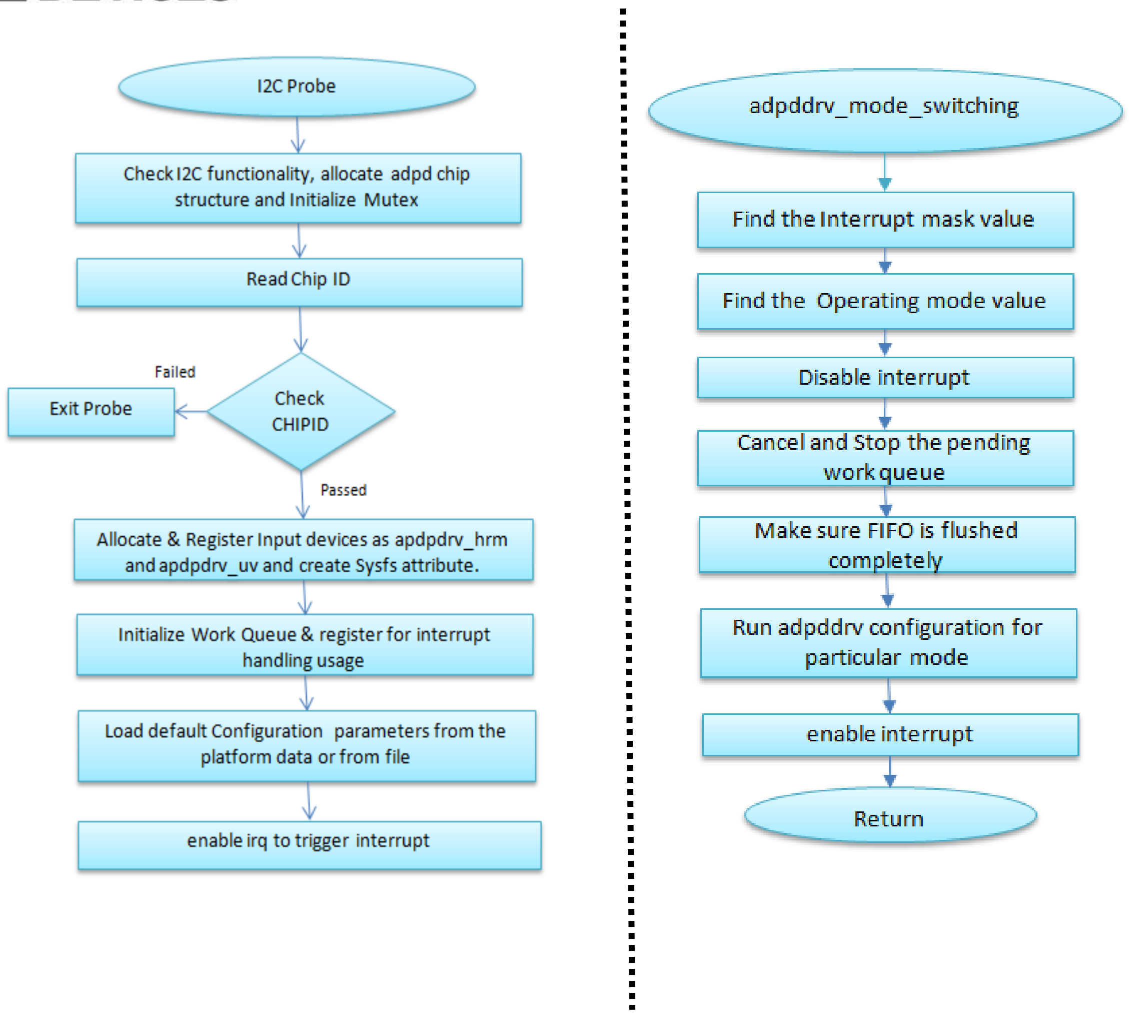
I2C Probe
Check I2C functionality, allocate adpd chip structure and Initialize Mutex
Read Chip ID
Check CHIPID
Failed
Exit Probe
Passed
Allocate & Register Input devices as apdpdrv_hrm and apdpdrv_uv and create Sysfs attribute.
Initialize Work Queue & register for interrupt handling usage
Load default Configuration parameters from the platform data or from file
enable irq to trigger interrupt
adpddrv_mode_switching
Find the Interrupt mask value
Find the Operating mode value
Disable interrupt
Cancel and Stop the pending work queue
Make sure FIFO is flushed completely
Run adpddrv configuration for particular mode
enable interrupt
Return

Figure 3- Initialization Routine & Mode switch service routine

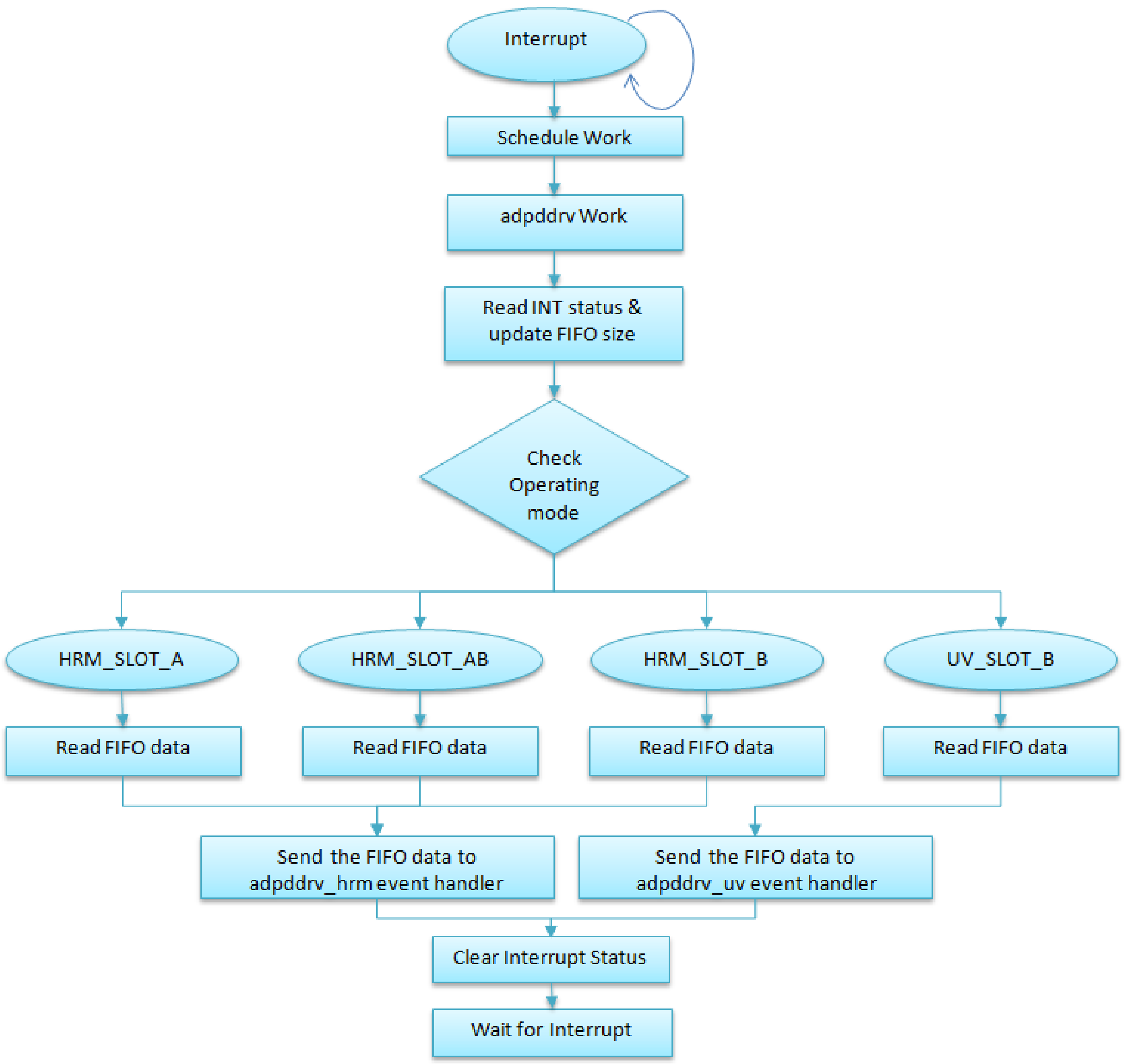

Figure 4: Interrupt Service Routine

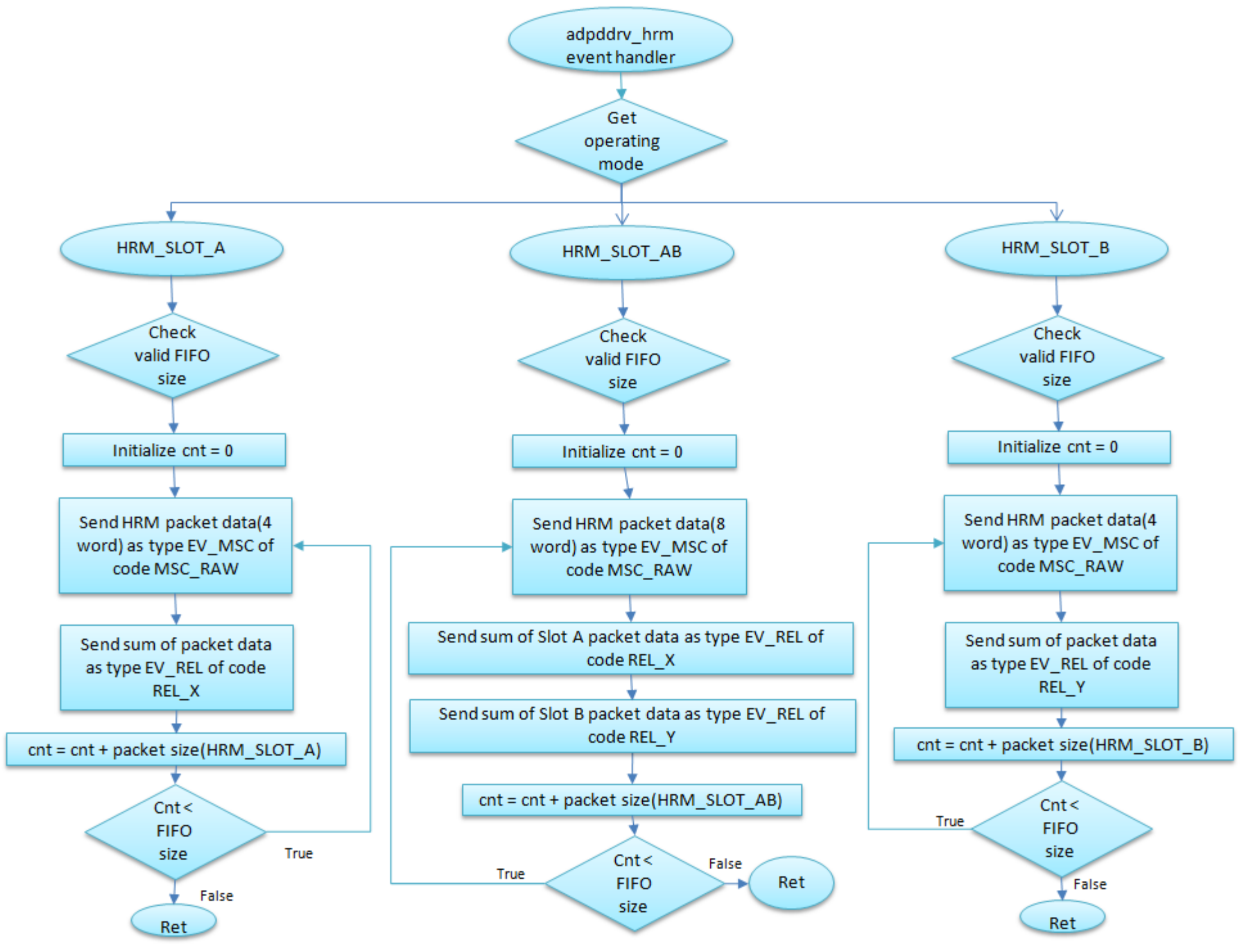
adpddrv_hrm event handler
Get operating mode
HRM_SLOT_A
HRM_SLOT_AB
HRM_SLOT_B
Check valid FIFO size
Check valid FIFO size
Check valid FIFO size
Initialize cnt = 0
Initialize cnt = 0
Initialize cnt = 0
Send HRM packet data(4 word) as type EV_MSC of code MSC_RAW
Send HRM packet data(8 word) as type EV_MSC of code MSC_RAW
Send HRM packet data(4 word) as type EV_MSC of code MSC_RAW
Send sum of packet data as type EV_REL of code REL_X
Send sum of Slot A packet data as type EV_REL of code REL_X
Send sum of Slot B packet data as type EV_REL of code REL_Y
Send sum of packet data as type EV_REL of code REL_Y
cnt = cnt + packet size(HRM_SLOT_A)
cnt = cnt + packet size(HRM_SLOT_AB)
cnt = cnt + packet size(HRM_SLOT_B)
Cnt < FIFO size
True
False
Ret
Cnt < FIFO size
True
False
Ret
Cnt < FIFO size
True
False
Ret

Figure 5- HRM event Handling

3 ADPD143 Kernel Module Description

The following are the features that would be supported in ADPD143 driver.

1) Event interface – Data will be returned in the form of Linux events to user space
2) SYSFS support – For run time device access such as device re – configuration, read/write single registers, configure interrupt mode, get stages information etc.

The device driver implementation procedure is as follows.

3.1 I2C DRIVER

In Linux/Android kernel, any i2c device cannot be probed by itself automatically as any hot pluggable device like USB. So, the i2c device needs to be instantiated. There are multiple ways of I2C instantiation.

3.1.1 I2C Device Instantiation for Device Tree

An I2C device with name "*ADPD_SENSOR*" and its corresponding address say, "0x64" is registered in the device tree file.

For example, the below illustration is based on the Snapdragon 800 series. (APQ8074 platform: ***<kernel>/arch/arm/boot/dts/apq8074-dragonboard.dtsi***).

```
i2c@f9928000{
         status = "ok";
         adpd143@64{
              compatible = "ad,adpd143";
               reg = <0x64>;
               interrupt-parent = <&msmgpio>;
               interrupts = <59 0x2>;
               /*interrupt-names = "adpd143_irq";*/
               interrupt-names = "hrm_sensor_irq";
               ad,irq-gpio = <&msmgpio 59 0x00>;
         };
     };
```

3.2 DRIVER PREPARATION AS MODULE

The driver can be built outside the kernel as a standalone Loadable Kernel module (LKM). During runtime, the module can be inserted and removed any number of times. This module driver is generic and thereby can be supported across any platform which supports standard I2C bus interface and GPIO compatible for interrupt handling.

3.2.1 COMPILE TIME OPTIONS SUMMARY FOR EXTERNAL MODULE

To build the driver below specified command can be used.

make

To enable the DEBUG log information of the driver add the following along with make

DEBUG=ON (i.e. make DEBUG=ON)

This will help generate the kernel level logs.

To configure the interrupt as Edge, the following option can be used with make,

INT=EDGE

This will configure the Driver to generate Interrupt based on RISING EDGE model. To configure LEVEL interrupt, the following option can be used with make,

INT=LEVEL

To display a brief configuration setting summary, use the following command

make help

3.2.2 DRIVER PREPARATION AS PART OF KERNEL

Modify the kconfig and the Makefile of the "$(KERNELDIR_PATH)\driver\input\misc" folder to include the ADPD143 driver file.

In the kernel add the object to the Makefile along with the ccflags

Similarly the **Makefile** has to be modified to add ADPD143 in the kernel build.

```
/*************************************************************************/
ccflags-$(CONFIG_ADPD143_DBG)          += -DADPD_DBG
obj-$(CONFIG_INPUT_ADPD143)          += adpd143.o
/*************************************************************************/
```

These options would be available once the ADPD143 Sensor support option is enabled by either selecting as Y (part of Kernel) or M (Module).

Similarly the **Kconfig** has to be modified to add ADPD143 in the kernel build.

Go to the Kernel Menuconfig & go to the Device Driver option to select ADPD143 to be built as part of the kernel image.

Device Drivers ---> Input Device support ---> Miscellaneous devices ---> <*> ADPD143 HRM Sensor Support

Or

Go to the Kernel Menuconfig & go to the Device Driver option to select ADPD143 as a kernel module.

Device Drivers ---> Input Device support ---> miscellaneous devices ---> <M> ADPD143 HRM Sensor Support

3.2.3 Test Environment

The following is the hardware setup used to test the driver module.

1. Target hardware and host board needs to be connected as mentioned in the previous section
2. For adb logs, appropriate USB cable must be connected to the PC
3. Power connections to Host board
4. Target hardware is powered via USB

3.2.4 Test Procedure

This section describes the steps to be followed for running the application binaries to validate the driver module in the target hardware. Prior to testing the driver module, the following steps are needed to be executed **ONCE**. These steps are based on the following assumptions,

- The target hardware is already burned with required Android image (4.0.4 – 4.3)
- Host machine (either Windows/Linux) has *'adb'* and '*fastboot*' tools installed and these are accessible from any directory.
- In the host machine, switch to the directory that has ADPD143.ko file. From this directory, the user should be able to access '*adb*' tool. So, host machine's PATH environment variable has to be set accordingly.

Step 1: After powering on the boards, open the command prompt or command terminal. Multiple command prompts can be opened for handling the commands and monitoring the events.

Step 2: Once the Android has finished booting, the necessary files needs to be pushed to the board via adb. The following commands need to be given from host machine (either Windows/Linux).

:> adb start-server

```
* daemon not running. starting it now on port 5037 *

* daemon started successfully *
```

:> adb devices

```
List of devices attached

XXXXXXXXXXXXX          device
```

:> adb shell /* give the following commands in shell prompt */

```
shell@android:/ $ su

shell@android:/ # chmod 777 /data/misc/
```

```
shell@android:/ # exit

shell@android:/ $ exit
```

:> adb push ADPD143.ko /data/misc/.

```
434 KB/s (366906 bytes in 0.824s)
```

3.2.5 ADPD143 Driver Module Validation

This section provides details on the commands need to be given to install the driver and run the binaries. In target side, on the 'adb' terminal, execute the following at the command prompt

```
/*******************************************************************/
 :> su
:> cd /data/misc/
/*******************************************************************/
```

3.2.6 Dynamic module insertion

Transfer the build ADPD143.ko file to the Target. In the prompt give the following command to insert the driver module.

```
/*******************************************************************/
:> insmod adpd143.ko
/*******************************************************************/
```

3.2.6.1 Dynamic Options for the Module

While inserting the ADPD143 kernel module, the following options are available for the host.

1) Defining the interrupt number for ADPD143 device

3.2.6.2 Interrupt to Host

An option to intimate the GPIO number for host interrupt is provided via module parameter. The parameter is 'gpio_num'.

```
insmod ADPD143.ko gpio_num=97
```

Note: The interrupt number would vary across the customer platforms. The customer could define the interrupt number via ***'gpio_num'*** *module parameter.*

3.3 SYSFS Support & the ADPD143 Driver Control

3.3.1 Sysfs Path

ADPD143 driver provides the support for several sysfs functions for the user space in order to talk to the driver. The driver illustrates with an example of how to create the sysfs attributes specific to the ADPD143 device in a pre-determined path.

For example, a separate class called 'adpd_sensor' is first created by the driver. Under this class, a device is created under the name 'ADPD143_sensor'. The attributes are created under the device 'adpd143. The sysfs path may look like the below,

```
/sys/class/sensors/adpd143 /
```

3.3.2 Input Devices

The following input device is created for the ADPD143 device.

ADPD143 - input device for the mode related to Heart Rate Monitor.

3.3.3 Purpose of the Sysfs

The following are the functionalities that are available for access from user space,

- Mode selection.
- Reading & Writing Registers of ADPD143 directly.
- Configuration files for particular mode written to register directly.

The various sysfs attribute are as follows,

3.3.4 Sysfs Attributes

The following SYSFS attributes are provided to the user to access the device during run time.

1) *enable*

2) *mode*
3) *reg_read*
4) *reg_write*
5) *configuration*

3.3.4.1 Operating mode support:

“***mode***” sysfs attribute is available for the user to change the driver mode. As per register mapping the possible modes the user can access and control are as follows,

- HRM_SLOT_A,
- HRM_SLOT_AB,
- HRM_SLOT_B and

The following is the command format to be given to change the operating mode.

```
echo [sysfs_value] > /sys/class/sensors/adpd143/mode
```

The following is the reference table for mode configuration.

Application	Operating Mode		INPUT_EVENT_NAME			Actual Device Parameter being sent from the ADPD143
	NAME	SYSFS VALUE	NAME	TYPE	CODE	
	IDLE_OFF	0x0	NIL	NIL	NIL	None
Heart Rate Monitor	Red (Slot A)	0x30	ADPD143	EV_REL	REL_X	X1_a + X2_a + Y1_a + Y2_a
				EV_MSC	MSC_RAW	X1 Value
					MSC_RAW	X2 Value
					MSC_RAW	Y1 Value
					MSC_RAW	Y2 Value
	IR (Slot B)	0x32	ADPD143	EV_REL	REL_Y	X1_b + X2_b + Y1_b + Y2_b
				EV_MSC	MSC_RAW	X1 Value
					MSC_RAW	X2 Value
					MSC_RAW	Y1 Value
					MSC_RAW	Y2 Value
	RI (Slot AB)	0x31	ADPD143	EV_REL	REL_X	X1_a + X2_a + Y1_a + Y2_a
				EV_MSC	MSC_RAW	X1 Value
					MSC_RAW	X2 Value
					MSC_RAW	Y1 Value

					MSC_RAW	Y2 Value
				EV_REL	REL_Y	X1_b + X2_b + Y1_b + Y2_b
				EV_MSC	MSC_RAW	X1 Value
					MSC_RAW	X2 Value
					MSC_RAW	Y1 Value
					MSC_RAW	Y2 Value

3.3.4.2 Read register

"reg_read" attribute is available for the user to read a register data during runtime.

The following is the command format to be given to access this attribute.

```
echo [Reg Addr] > /sys/class/sensors/adpd143/reg_read
```

The following is the command format to read the register after the above command is given

```
cat /sys/class/sensors/adpd143/reg_read
```

3.3.4.3 Write register

"adpd_reg_write" attribute is available for the user to write to a register during runtime. The following is the command format to be given to access this attribute.

```
echo [Reg Addr] [Data]> /sys/class/sensors/adpd143/reg_write
```

3.3.4.4 DEVICE CONFIGURATION

When the driver is inserted as a module, the initialization routine is called. In the initialization routine, the default hardware configuration which is the general configuration (i.e., hrm configuration) for the device is extracted using the filp if the file is available in the directory specified or uses default platform data declared in the driver. Depending upon the current mode the driver loads the suitable mode configuration when initiated. The device configuration could be controlled by the '*adpd_configuration*' sysfs parameter.

There are two methods by which the ADPD143 platform data could be configured.
They are,

1) Configuration using a File

2) Compile time

3.3.4.4.1 Configuration using a File

A HRM configuration file could be kept at a selected directory. For ex, /data/misc /.
The format of the file is string based. Extension of the file is *.dcfg. ADPD143 driver will read this file and convert it to the required platform data and then load the device registers. This method is chosen by writing '0' to the '*adpd_configuration*' sysfs parameter. Below are the names of the configuration used by different modes.

ADPD143_ config.dcfg	- Configuration file name related to HRM modes.

Note: - The configuration file should not contain any special characters other than # that is used for commenting.

3.3.4.4.2 Configuring device at compile time

The much needed platform data is provided as a static 'C' array inside the driver. This method is chosen by writing a '1' to the '*configuration*' sysfs parameter.

```
$echo 1 > /sys/class/sensors/adpd143/configuration
```

Note: The default configuration method is the compile time method. i.e 'configuration' parameter would be set to '1' as default.

3.4 Application Executables Usage

Android 'getevent' tool can be used to validate the driver for event interface. The getevent tool runs on the device and provides information about input devices and a live dump of kernel input events. It is very useful tool for ensuring that device drivers are reporting the expected set of capabilities for each input device and are generating the desired stream of input events.

```
:> getevent –t –l
```