MediaTek

# Maui Device Driver Document

Keypad Driver Design Specification

**Documents Number:**

**Preliminary Information**

**Revision: 0.01**

**Release Date: Sep, 28, 2003**

## Revision History

| Revision | Date | Author | Comments |
|---|---|---|---|
| 0.01 | 06/30/2003 | Kumar Chen | Draft version |
| | | | |
| | | | |

## Table of contents

# 1 Introduction

## 1.1 Overview

This document describes the design concept of the keypad driver & task.

## 1.2 References

- MT6205B Baseband spec.
- Kwypad_API.doc

# 2 Architecture

## 2.1 Block Diagram

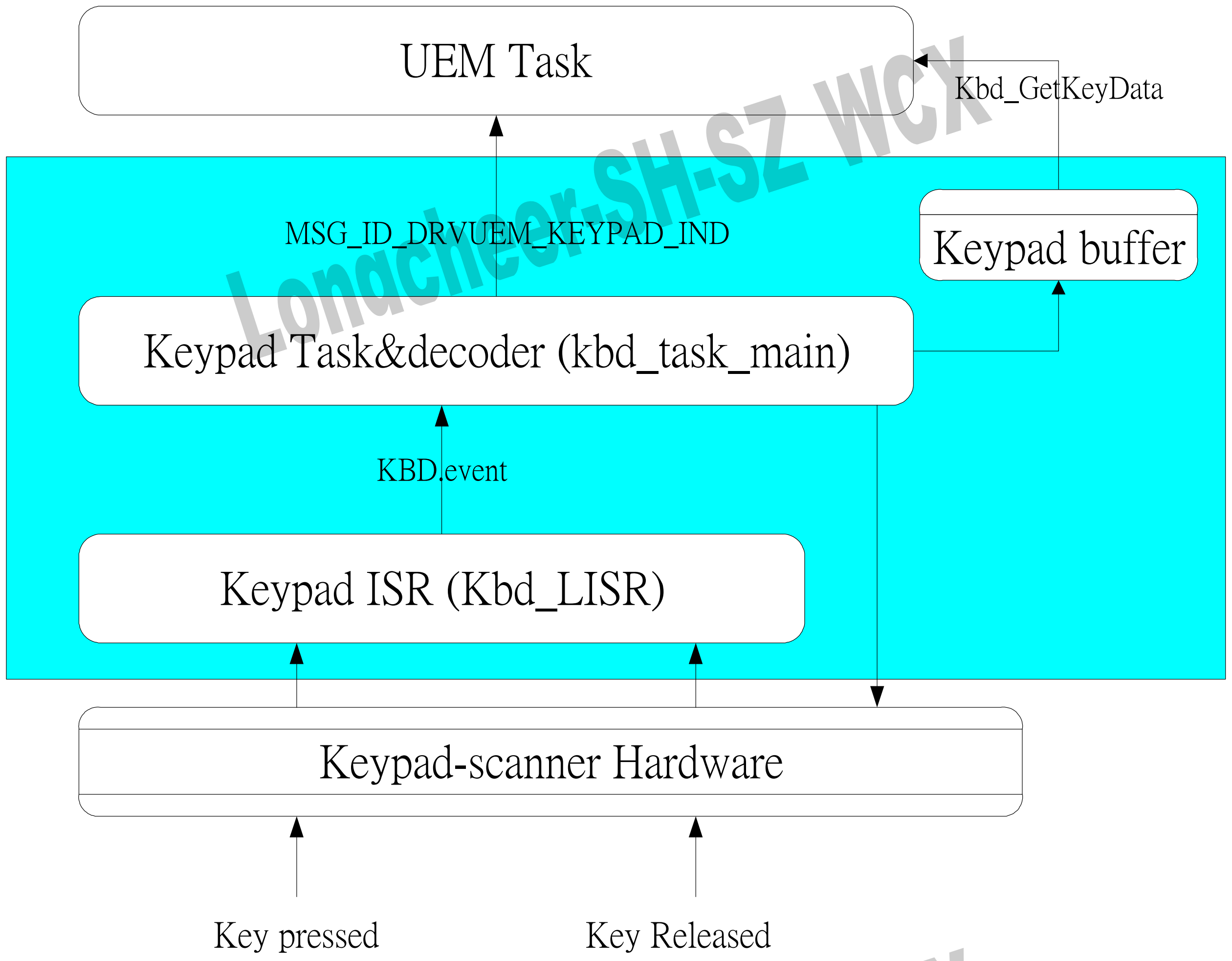

*Figure 1 The keypad architecture*

## 2.2 Functional overview

Keypad driver contains a keypad ISR and a keypad task. When user press or release keys, keypad ISR will generate an event to keypad task. The keypad-decoder is executed in keypad task. The keypad task will detect different key events and put these key events to keypad buffer.

There is an important rule to wakeup UEM task: When keypad task find this buffer is empty and then detect a key event(MSG_ID_DRVUEM_KEYPAD_IND), the keypad task will send a primitive to notify UEM wake up. **UEM must read all of the key events in this keypad buffer by using "Kbd_GetKeyData" function when UEM receive keypad primitive.** If UEM doesn't read out all events in keypad buffer, keypad task will not send primitive to wake up UEM again. This rule can reduce the system context-switching time, and keep the keypad detection efficiently.

Basically our keypad decoder can detect two key pressed concurrently. If this feature is needed, this TWO_KEY_ENABLE local compiler option should be opened. The default setting doesn’t enable this compiler option, and keypad task (decoder) will filter the multiple keys (key number > 1 concurrently) not to put into keypad buffer. Besides, keypad task will also detect long pressed key and repeated key for upper layers.

kbd_onekey_longpress and kbd_onekey_repeated are supported. While one key is pressed util long press timer is reached, additional long pressed key event is pushed into the key buffer. In the meantime, if the one keep pressing the key without release, repeating key event is pushsed into the key buffer each repeat timer period.

Sometimes the power key is defined at more than one physical positon such as total column 0. The keypad task will detect this issue and assume only one key event is pushed.

Some platforms use END key as power key. Correctly define the corresponding keypad array is very important. Don’t be confused with these two keys.

While UEM layer is too busy to consume all of the key events in the key buffer, key events are lost after the key buffer is full. Otherwise, keypad task will reserve enough key buffers for the corresponding key release event. Keypad task will guarantee key pressed envets and key released events are in pairs no matter long pressed keys or repeated keys are generated.

# 3 Data Structure

## 3.1 kbd_event

**Definition:**

```
typedef enum {
     kbd_onekey_press=0,
     kbd_onekey_release,
#ifdef TWO_KEY_ENABLE
     kbd_twokey_press,
     kbd_twokey_release,
#endif   /*TWO_KEY_ENABLE*/
     kbd_onekey_longpress,
     kbd_onekey_repeated
} kbd_event;
```

**Members:**

| Members | Description |
|---|---|
| kbd_onekey_press | one key pressed event |
| kbd_onekey_release | one key released event |
| kbd_twokey_press | two key pressed concurrently event |
| kbd_twokey_release | two key released concurrently event |
| kbd_onekey_longpress | long pressed key event |
| kbd_onekey_repeated | repeated key event |

## 3.2 kbd_data

**Definition:**

```
typedef struct
{
   kbd_event   Keyevent;
#ifdef TWO_KEY_ENABLE
   kal_uint8   Keydata[2];
#else /*!TWO_KEY_ENABLE*/
   kal_uint8   Keydata[1];
#endif   /*TWO_KEY_ENABLE*/
} kbd_data;
```

**Members:**

| Members | Description |
|---|---|
| Keyevent | Keypad event listed above |
| Keydata | Keypad data array |

## 3.3 kbd_struct

**Definition:**

```
typedef struct
{
   kal_hisrid            hisr;
   kal_eventgrpid        event;
   kal_uint8             gpthandle;
   kal_uint32            longpress_timeout;
   kal_uint32            repeat_time;
   kal_uint32            kbdmap_reg;
} kbd_struct;
```

**Members:**

| Members | Description |
|---|---|
| hisr | KAL hisr id |
| event | KAL event group id |
| gpthandle | GPT timer handle for long-pressed & repeated key detection |
| longpress_timeout | Time out value for long pressed key(unit:10ms) |
| repeat_time | Time out value for repeated key(unit:10ms) |
| kbdmap_reg | Keypad map HW register backup for keypad-decoder |

# 4 Function

## 4.1 Kbd_SetLongPressTime

void Kbd_SetLongPressTime(kal_uint32 ticks)

This function is to notify driver what time means long press!! At initialize, the default value of the timeout value is 2s.
Note that: the unit of tick is **10ms** in this function.

**Parameters:**

| Members | Description |
|---|---|
| ticks | Time out value for long pressed key |

**Return value:**
None

**Example:**
None

## 4.2 Kbd_SetRepeatTime

void Kbd_SetRepeatTime(kal_uint32 ticks)

This function is to notify driver what time means repeated key!! The default value of the timeout value is 1s
Note that: the unit of tick is **10ms** in this function.

**Parameters:**

| Members | Description |
|---|---|
| ticks | Time out value for repeated key |

**Return value:**
None

**Example:**
None

## 4.3 Kbd_GetKeyData

kal_bool Kbd_GetKeyData(kbd_data *keydata)

This function should be called by upper layer to read out the key event when receiving a keypad primitive.

**Parameters:**

| Members | Description |
|---|---|
| keydata | Data pointer for kbd_data structure |

**Return value:**
The keydata is valid only when return value is KAL_TRUE. When this function returns KAL_FALSE, the buffer is empty and the content of keydata is useless.

**Example:**
None

# 5 Message

## 5.1 MSG_ID_DRVUEM_KEYPAD_IND

**Description:**

This primitive is used to wake up UEM task to read out the keypad events from keypad buffer.

**Local Parameter:**

N/A

**Reference:**

N/A

## Figures index