**Design Document for PIT Driver**

 

# Outline

This document describes the PIT (Periodic Interrupt Timer) driver in Linux kernel of MVF TOWER BOARD (XTWR-VF600) with MVF SoC.

PIT driver provides highly-accurate timer function by API for various drivers in kernel.

Out of 8 PIT timers, PIT0 can be used as long as it is not defined by the kernel as TICK timer.

# Existing code to be changed

All source code is newly written.

# API of new functions

Define 6 APIs to control timer from driver.

## pit\_alloc\_timer function

Assign PIT timer.

Drivers employing this timer use this function to obtain and control TimerHandle.

Prototype: int pit\_alloc\_timer (pit\_channel ch)

Argument: ch:PIT channel (refer to enum below)

Return value: Negative value: Error

Positive value: TimerHandle

Note: If the kernel assigns PIT0 as System Timer, this allocation causes an error.

* enum pit\_channel

typedef enum {

PIT0, PIT1, PIT2, PIT3, PIT4,PIT5,PIT6,PIT7

PIT\_AVAILABLE\_CHANNEL

} pit\_channel;

PIT\_AVAILABLE\_CHANNEL is used to obtain available channel of PIT.

## pit\_param\_set function

Set timer by parameter, and register callback function for timer interrupt.

Prototype: int pit\_param\_set (int timer\_handle, unsigned long load\_value,

void (\*event\_handler)(int ch))

Argument: timer\_handle: Handle obtained by pit\_alloc\_timer

load\_value: 32bit timer setting value (default value of down counter)

event\_handler: Event handler (NULL can be specified)

Return value: Negative value: Error

0: Set successfully

## pit\_enable\_timer function

Start timer.

An error occurs if load\_value is not set by pit\_param\_set function.

Prototype: int pit\_enable\_timer (int timer\_handle)

Argument: timer\_handle: Handle obtained by pit\_alloc\_timer

Return value: Negative value: Error

0: Start successfully

## pit\_disable\_timer function

Stop timer.

Prototype: int pit\_disable\_timer (int timer\_handle)

Argument: timer\_handle: Handle obtained by pit\_alloc\_timer

Return value: Negative value: Error

0: Stop successfully

## pit\_read\_counter function

Read counter value.

Counter value is 4 bytes and copy read-value of PIT\_CVALn (Current timer value) register to buffer.

Prototype: int pit\_read\_counter (int timer\_handle, unsigned long \*counter)

Argument: timer\_handle: Handle obtained by pit\_alloc\_timer

Counter: Pointer of variable to obtain counter value

Return value: Negative value: Error

0: Read successfully

## pit\_free\_timer function

Release timer assigned by pit\_alloc\_timer.

Prototype: int pit\_free\_timer (int timer\_handle)

Argument: timer\_handle: Handle obtained by pit\_alloc\_timer

Return value: Negative value: Error

0: Release successfully

# Expected register settings

Parameters settable for Timer Load Value of 3.2 pit\_param\_set function comply with processor manual.

# Expected functionality and usage

This driver assumes that the following operations are done as a sequence from device driver.

1. Obtain handle by pit\_alloc\_timer
2. Set parameter and register callback function by pit\_param\_set
3. Start timer by pit\_enable\_timer
4. Timer processing by callback function, or timer read and such
5. Stop timer by pit\_disable\_timer
6. Release timer by pit\_free\_timer at the time of driver unload

PIT driver employs platform framework and enables it by resource definition.

For example, when defining PIT;

static struct resource pit\_resources[] = {

[0] = {

.start = MVF\_PIT\_BASE\_ADDR,

.end = MVF\_PIT\_BASE\_ADDR + SZ\_4K-1,

.flags = IORESOURCE\_MEM,

},

[1] = {

.start =MXC\_INT\_PIT,

.end =MXC\_INT\_PIT,

.flags = IORESOURCE\_IRQ,

},

};

static struct platform\_device pit\_device = {

.name = "pit",

.id = 0,

.num\_resources = 2,

.resource = pit\_resources,

};

Describe these definitions and define as platform resource by the following at startup initialization function of the kernel.

platform\_device\_register(&pit\_device);

# Any other pertinent information

This driver is implemented by using framework of platform device.