**Design Document for LPTMR Driver**

 

# Outline

This document describes the LPTMR (Low-Power Timer) driver in Linux kernel of MVF TOWER BOARD (XTWR-VF600) with MVF SoC. LPTMR driver provides Low Power Timer function by API for various drivers in kernel.

# Existing code to be changed

All source code is newly written.

# API of new functions

Define 6 APIs to control timer from driver.

## lpt\_alloc\_timer function

Assign LP timer.

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

Prototype: int lpt\_alloc\_timer(void)

Return value: Negative value: Error

Positive value: TimerHandle

## lpt\_param\_set function

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

Prototype: int lpt\_param\_set (int timer\_handle, struct mvf\_lpt\_request req,,

void (\*event\_handler)(void))

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

Req: Timer parameters (described below)

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

Return value: Negative value: Error

0: Set successfully

* struct mvf\_lpt\_request

Members of the structure are explained as below.

struct mvf\_ftm\_request{

unsigned long compare\_value;

unsigned short timer\_mode;

unsigned short pulse\_pin\_polarity;

unsigned short pulse\_pin\_select;

unsigned short prs\_clock\_sel;

unsigned short prs\_bypass;

unsigned short prs\_value;

};

・compare\_value: Member to define maximum counter value of timer

Valid up to 16 bit.

・timer\_mode: Member to define timer mode

Select from the following 2 parameters.

LPT\_PARAM\_TM\_TIMECOUNTER (Timer counter mode)

LPT\_PARAM\_TM\_PULSECOUNTER (Pulse counter mode)

・pulse\_pin\_polarity: Member to define pin polarity for pulse counter mode

Use when timer\_mode is LPT\_TM\_PARAM\_PULSECOUNTER.

Select from the following 2 parameters.

LPT\_PARAM\_PPP\_ACTIVEHIGH

LPT\_PARAM\_PPP\_ACTIVELOW

・pulse\_pin\_select: Member to define pin source setting in timer source.

Use when timer\_mode is LPT\_TM\_PARAM\_PULSECOUNTER.

Select from the following 4 parameters.

LPT\_PARAM\_PPS\_INPUT0

LPT\_PARAM\_PPS\_INPUT1

LPT\_PARAM\_PPS\_INPUT2

LPT\_PARAM\_PPS\_INPUT3

・prs\_clock\_sel: Member to define clock of prescaler

Select from the following 4 parameters

LPT\_PARAM\_PCS\_CLOCK0

LPT\_PARAM\_PCS\_CLOCK1

LPT\_PARAM\_PCS\_CLOCK2

LPT\_PARAM\_PCS\_CLOCK3

・prs\_bypass: Member to define Glitch Filter

Select from the following 2 parameters

LPT\_PARAM\_PB\_GF\_ENABLE

LPT\_PARAM\_PB\_GF\_BYPASS

・prs\_value: Member to define divider of prescaler/Glitch Filter detection threshold

Set divider when timer\_mode is LPT\_PARAM\_TM\_TIMECOUNTER, and set chattering elimination time when timer\_mode is LPT\_PARAM\_TM\_PULSECOUNTER.

Select from the following 16 parameters

LPT\_PARAM\_PV\_DIV2\_NA

LPT\_PARAM\_PV\_DIV4\_RISE2

LPT\_PARAM\_PV\_DIV8\_RISE4

LPT\_PARAM\_PV\_DIV16\_RISE8

LPT\_PARAM\_PV\_DIV32\_RISE16

LPT\_PARAM\_PV\_DIV64\_RISE32

LPT\_PARAM\_PV\_DIV128\_RISE64

LPT\_PARAM\_PV\_DIV256\_RISE128

LPT\_PARAM\_PV\_DIV512\_RISE256

LPT\_PARAM\_PV\_DIV1024\_RISE512

LPT\_PARAM\_PV\_DIV2048\_RISE1024

LPT\_PARAM\_PV\_DIV4096\_RISE2048

LPT\_PARAM\_PV\_DIV8192\_RISE4096

LPT\_PARAM\_PV\_DIV16384\_RISE8192

LPT\_PARAM\_PV\_DIV32768\_RISE16384

LPT\_PARAM\_PV\_DIV65536\_RISE32768

## lpt\_enable\_timer function

Start timer.

An error occurs if it is not set by lpt\_param\_set function.

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

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

Return value: Negative value: Error

0: Start successfully

## lpt\_disable\_timer function

Stop timer.

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

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

Return value: Negative value: Error

0: Stop successfully

## lpt\_read\_counter function

Read counter value.

Counter value is 2 bytes and copy read-value of LPTMR\_CNR (Counter Value) register to buffer.

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

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

Counter: Pointer of variable to obtain counter value

Return value: Negative value: Error

0: Read successfully

## lpt\_free\_timer function

Release timer assigned by lpt\_alloc\_timer.

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

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

Return value: Negative value: Error

0: Release successfully

# Expected register settings

Parameters settable for 3.2 lpt\_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 lpt\_alloc\_timer
2. Set parameter and register callback function by lpt\_param\_set
3. Start timer by lpt\_enable\_timer
4. Timer processing by callback function, or timer read and such
5. Stop timer by lpt\_disable\_timer
6. Release timer by lpt\_free\_timer at the time of driver unload

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

For example, when defining LTP;

static struct resource lpt\_resources[] = {

[0] = {

.start = MVF\_LPT\_BASE\_ADDR,

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

.flags = IORESOURCE\_MEM,

},

[1] = {

.start =MXC\_INT\_LPT,

.end =MXC\_INT\_LPT,

.flags = IORESOURCE\_IRQ,

},

};

static struct platform\_device lpt\_device = {

.name = "lpt",

.id = 0,

.num\_resources = 2,

.resource = \_resources,

};

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

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

# Any other pertinent information

This driver is implemented by using framework of platform device.