**Design Document for FTM Driver**

 

# Outline

This document describes the FTM (Flex Timer Module) driver in Linux kernel of MVF TOWER BOARD (XTWR-VF600) with MVF SoC. FTM driver provides highly-accurate 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.

## ftm\_alloc\_timer function

Assign FTM timer.

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

Prototype: int ftm\_alloc\_timer (fmt\_channel ch)

Argument: ch: FMT channel (described below)

Return value: Negative value: Error

Positive value: TimerHandle

* enum ftm\_channel

typedef enum {

FMT0,

FMT1,

FMT\_AVAILABLE\_CHANNEL

} ftm\_channel;

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

## ftm\_param\_set function

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

Prototype: int ftm\_param\_set (int timer\_handle, struct mvf\_ftm\_request req,

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

Argument: timer\_handle: Handle obtained by ftm\_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\_ftm\_request

Members of the structure are explained as below.

struct mvf\_ftm\_request{

unsigned long clocksource;

unsigned long divider;

unsigned short start;

unsigned short end;

};

・clocksource: Member to define clock source

Select from the following 4 parameters,

FTM\_PARAM\_CLK\_NOCLOCK No clock

FTM\_PARAM\_CLK\_SYSTEMCLOCK System clock

FTM\_PARAM\_CLK\_FIXEDFREQ Fixed clock

FTM\_PARAM\_CLK\_EXTERNAL External clock

・divider: Member to define frequency dividing for clock source

Select from the following 8 parameters.

FTM\_PARAM\_DIV\_BY\_1 Gate clock source

FTM\_PARAM\_DIV\_BY\_2 1/2 frequency

FTM\_PARAM\_DIV\_BY\_4 1/4 frequency

FTM\_PARAM\_DIV\_BY\_8 1/8 frequency

FTM\_PARAM\_DIV\_BY\_16 1/16 frequency

FTM\_PARAM\_DIV\_BY\_32 1/32 frequency

FTM\_PARAM\_DIV\_BY\_64 1/64 frequency

FTM\_PARAM\_DIV\_BY\_128 1/128 frequency

・start/end: Member to define start and end value of counter

Set the value of 0-0xffff.

Set start value to FTM\_CNTIN (Counter Initial Value) register and end value to FTM\_MOD (Modulo) register.

Limit value for above values comply with processor manual.

## ftm\_enable\_timer function

Start timer.

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

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

Argument: timer\_handle: Handle obtaind by ftm\_alloc\_timer

Return value: Negative value: Error

0: Start successfully

## ftm\_disable\_timer function

Stop timer.

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

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

Return value: Negative value: Error

0: Stop successfully

## ftm\_read\_counter function

Read counter value.

Counter value is 2 bytes and copy read-value of FTM\_CNT (Counter) register to variable.

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

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

Counter: Pointer of variable to obtain counter value

Return value: Negative value: Error

0: Read successfully

## ftm\_free\_timer function

Release timer assigned by ftm\_alloc\_timer.

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

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

Return value: Negative value: Error

0: Release successfully

# Expected register settings

Parameters settable for timer register are in the range of the ones settable using struct mvf\_ftm\_request req structure of 3.2 ftm\_param\_set function.

# Expected functionality and usage

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

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

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

For example, when defining FTM0;

static struct resource ftm\_resources[] = {

[0] = {

.start = MVF\_FTM0\_BASE\_ADDR,

.end = MVF\_FTM0\_BASE\_ADDR + 0x1000 -1,

.flags = IORESOURCE\_MEM,

},

[1] = {

.start = MXC\_INT\_FTM0,

.end = MXC\_INT\_FTM0,

.flags = IORESOURCE\_IRQ,

},

};

static struct platform\_device ftm\_device = {

.name = "ftm",

.id = 0,

.num\_resources = 2,

.resource = ftm\_resources,

};

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

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

# Any other pertinent information

This driver is implemented by using framework of platform device.