Design Document for FTM Driver

1 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.

2 Existing code to be changed

All source code is newly written.

3 API of new functions

Define 6 APIs to control timer from driver.

3.1 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.

3.2 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.

3.3 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

3.4 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

3.5 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

3.6 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

4 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.

5 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);
```

6 Any other pertinent information

This driver is implemented by using framework of platform device.