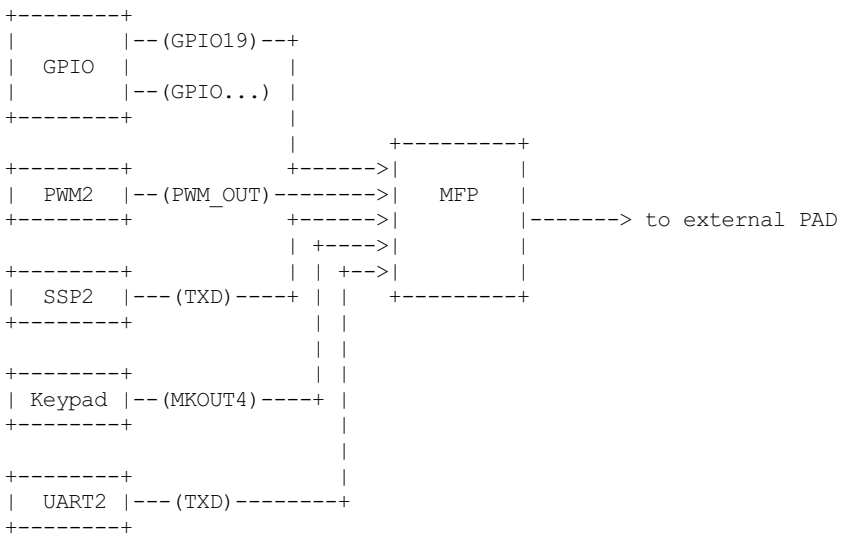
MFP Configuration for PXA2xx/PXA3xx Processors

Eric Miao <eric.miao@marvell.com>

MFP stands for Multi-Function Pin, which is the pin-mux logic on PXA3xx and later PXA series processors. This document describes the existing MFP API, and how board/platform driver authors could make use of it.

Basic Concept

Unlike the GPIO alternate function settings on PXA25x and PXA27x, a new MFP mechanism is introduced from PXA3xx to completely move the pin-mux functions out of the GPIO controller. In addition to pin-mux configurations, the MFP also controls the low power state, driving strength, pull-up/down and event detection of each pin. Below is a diagram of internal connections between the MFP logic and the remaining SoC peripherals:



NOTE: the external pad is named as MFP_PIN_GPIO19, it doesn't necessarily mean it's dedicated for GPIO19, only as a hint that internally this pin can be routed from GPIO19 of the GPIO controller.

To better understand the change from PXA25x/PXA27x GPIO alternate function to this new MFP mechanism, here are several key points:

1. GPIO controller on PXA3xx is now a dedicated controller, same as other internal controllers like PWM, SSP and UART, with 128 internal signals which can be routed to external through one or more MFPs (e.g. GPIO<0> can be routed through either MFP_PIN_GPIO0 as well as MFP_PIN_GPIO0_2, see arch/arm/mach-pxa/mfp-pxa300.h)
2. Alternate function configuration is removed from this GPIO controller, the remaining functions are pure GPIO-specific, i.e.
    - GPIO signal level control
    - GPIO direction control
    - GPIO level change detection
3. Low power state for each pin is now controlled by MFP, this means the PGSRx registers on PXA2xx are now useless on PXA3xx
4. Wakeup detection is now controlled by MFP, PWER does not control the wakeup from GPIO(s) any more, depending on the sleeping state, ADxER (as defined in pxa3xx-regs.h) controls the wakeup from MFP

NOTE: with such a clear separation of MFP and GPIO, by GPIO<xx> we normally mean it is a GPIO signal, and by MFP<xxx> or pin xxx, we mean a physical pad (or ball).

MFP API Usage

For board code writers, here are some guidelines:

1. include ONE of the following header files in your <board>.c:
    - #include "mfp-pxa25x.h"
    - #include "mfp-pxa27x.h"
    - #include "mfp-pxa300.h"

- #include "mfp-pxa320.h"
- #include "mfp-pxa930.h"

NOTE: only one file in your <board>.c, depending on the processors used, because pin configuration definitions may conflict in these file (i.e. same name, different meaning and settings on different processors). E.g. for zylonite platform, which support both PXA300/PXA310 and PXA320, two separate files are introduced: zylonite_pxa300.c and zylonite_pxa320.c (in addition to handle MFP configuration differences, they also handle the other differences between the two combinations).

NOTE: PXA300 and PXA310 are almost identical in pin configurations (with PXA310 supporting some additional ones), thus the difference is actually covered in a single mfp-pxa300.h.

2. prepare an array for the initial pin configurations, e.g.:

```
static unsigned long mainstone_pin_config[] __initdata = {
   /* Chip Select */
   GPIO15_nCS_1,

   /* LCD - 16bpp Active TFT */
   GPIOxx_TFT_LCD_16BPP,
   GPIO16_PWM0_OUT,        /* Backlight */

   /* MMC */
   GPIO32_MMC_CLK,
   GPIO112_MMC_CMD,
   GPIO92_MMC_DAT_0,
   GPIO109_MMC_DAT_1,
   GPIO110_MMC_DAT_2,
   GPIO111_MMC_DAT_3,

   ...

   /* GPIO */
   GPIO1_GPIO | WAKEUP_ON_EDGE_BOTH,
};
```

a) once the pin configurations are passed to pxa{2xx,3xx}_mfp_config(), and written to the actual registers, they are useless and may discard, adding '__initdata' will help save some additional bytes here.

b) when there is only one possible pin configurations for a component, some simplified definitions can be used, e.g. GPIOxx_TFT_LCD_16BPP on PXA25x and PXA27x processors

c) if by board design, a pin can be configured to wake up the system from low power state, it can be 'OR'ed with any of:

WAKEUP_ON_EDGE_BOTH WAKEUP_ON_EDGE_RISE WAKEUP_ON_EDGE_FALL
WAKEUP_ON_LEVEL_HIGH - specifically for enabling of keypad GPIOs,

to indicate that this pin has the capability of wake-up the system, and on which edge(s). This, however, doesn't necessarily mean the pin _will_ wakeup the system, it will only when set_irq_wake() is invoked with the corresponding GPIO IRQ (GPIO_IRQ(xx) or gpio_to_irq()) and eventually calls gpio_set_wake() for the actual register setting.

d) although PXA3xx MFP supports edge detection on each pin, the internal logic will only wakeup the system when those specific bits in ADxER registers are set, which can be well mapped to the corresponding peripheral, thus set_irq_wake() can be called with the peripheral IRQ to enable the wakeup.

MFP on PXA3xx

Every external I/O pad on PXA3xx (excluding those for special purpose) has one MFP logic associated, and is controlled by one MFP register (MFPR).

The MFPR has the following bit definitions (for PXA300/PXA310/PXA320):

```
31                      16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
 +-------------------------+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 |         RESERVED        |PS|PU|PD|  DRIVE |SS|SD|SO|EC|EF|ER|--| AF_SEL |
 +-------------------------+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

 Bit 3:   RESERVED
 Bit 4:   EDGE_RISE_EN - enable detection of rising edge on this pin
 Bit 5:   EDGE_FALL_EN - enable detection of falling edge on this pin
 Bit 6:   EDGE_CLEAR   - disable edge detection on this pin
 Bit 7:   SLEEP_OE_N   - enable outputs during low power modes
 Bit 8:   SLEEP_DATA   - output data on the pin during low power modes
 Bit 9:   SLEEP_SEL    - selection control for low power modes signals
 Bit 13:  PULLDOWN_EN  - enable the internal pull-down resistor on this pin
 Bit 14:  PULLUP_EN    - enable the internal pull-up resistor on this pin
 Bit 15:  PULL_SEL     - pull state controlled by selected alternate function
                         (0) or by PULL{UP,DOWN}_EN bits (1)
```

```
Bit 0 - 2: AF_SEL - alternate function selection, 8 possibilities, from 0-7
Bit 10-12: DRIVE  - drive strength and slew rate
                    0b000 - fast 1mA
                    0b001 - fast 2mA
                    0b002 - fast 3mA
                    0b003 - fast 4mA
                    0b004 - slow 6mA
                    0b005 - fast 6mA
                    0b006 - slow 10mA
                    0b007 - fast 10mA
```

MFP Design for PXA2xx/PXA3xx

Due to the difference of pin-mux handling between PXA2xx and PXA3xx, a unified MFP API is introduced to cover both series of processors.

The basic idea of this design is to introduce definitions for all possible pin configurations, these definitions are processor and platform independent, and the actual API invoked to convert these definitions into register settings and make them effective there-after.

Files Involved

- arch/arm/mach-pxa/include/mach/mfp.h

for

1. Unified pin definitions - enum constants for all configurable pins
2. processor-neutral bit definitions for a possible MFP configuration

- arch/arm/mach-pxa/mfp-pxa3xx.h

for PXA3xx specific MFPR register bit definitions and PXA3xx common pin configurations

- arch/arm/mach-pxa/mfp-pxa2xx.h

for PXA2xx specific definitions and PXA25x/PXA27x common pin configurations

- arch/arm/mach-pxa/mfp-pxa25x.h arch/arm/mach-pxa/mfp-pxa27x.h arch/arm/mach-pxa/mfp-pxa300.h arch/arm/mach-pxa/mfp-pxa320.h arch/arm/mach-pxa/mfp-pxa930.h

for processor specific definitions

- arch/arm/mach-pxa/mfp-pxa3xx.c
- arch/arm/mach-pxa/mfp-pxa2xx.c

for implementation of the pin configuration to take effect for the actual processor.

Pin Configuration

The following comments are copied from mfp.h (see the actual source code for most updated info):

```
/*
 * a possible MFP configuration is represented by a 32-bit integer
 *
 * bit  0.. 9 - MFP Pin Number (1024 Pins Maximum)
 * bit 10..12 - Alternate Function Selection
 * bit 13..15 - Drive Strength
 * bit 16..18 - Low Power Mode State
 * bit 19..20 - Low Power Mode Edge Detection
 * bit 21..22 - Run Mode Pull State
 *
 * to facilitate the definition, the following macros are provided
 *
 * MFP_CFG_DEFAULT - default MFP configuration value, with
 *            alternate function = 0,
 *            drive strength = fast 3mA (MFP_DS03X)
 *            low power mode = default
 *            edge detection = none
 *
 * MFP_CFG  - default MFPR value with alternate function
 * MFP_CFG_DRV      - default MFPR value with alternate function and
 *            pin drive strength
 * MFP_CFG_LPM      - default MFPR value with alternate function and
 *            low power mode
 * MFP_CFG_X        - default MFPR value with alternate function,
 *            pin drive strength and low power mode
 */

Examples of pin configurations are::

  #define GPIO94_SSP3_RXD           MFP_CFG_X(GPIO94, AF1, DS08X, FLOAT)
```

```
which reads GPIO94 can be configured as SSP3_RXD, with alternate function
selection of 1, driving strength of 0b101, and a float state in low power
modes.

NOTE: this is the default setting of this pin being configured as SSP3_RXD
which can be modified a bit in board code, though it is not recommended to
do so, simply because this default setting is usually carefully encoded,
and is supposed to work in most cases.
```

Register Settings

Register settings on PXA3xx for a pin configuration is actually very straight-forward, most bits can be converted directly into MFPR value in a easier way. Two sets of MFPR values are calculated: the run-time ones and the low power mode ones, to allow different settings.

The conversion from a generic pin configuration to the actual register settings on PXA2xx is a bit complicated: many registers are involved, including GAFRx, GPDRx, PGSRx, PWER, PKWR, PFER and PRER. Please see mfp-pxa2xx.c for how the conversion is made.