Freescale Semiconductor
Application Note

Document Number: AN2855
Rev. 0, 2/2008

Pad Configuration and GPIO Driver for MPC5500

by: Martin Kaspar, EMEAGTM, Roznov
Daniel McKenna, MSG Applications, East Kilbride

Contents

1 Introduction

This document describes an application programming interface (API) that provides a functional set for pad configuration and access on the MPC5500 family. The API software (AN2855SW) is available to download from www.freescale.com.

2 Target Device

This API was designed for MPC5500 family devices; however, it can be adapted for use with MPC5510 family devices by making small changes; for example, by modifying the SIU header file to suit the target device.

3 C Compiler Compliance

The API was tested using the following C compilers:

- Metrowerks CodeWarrior for MPC5500 v1.5
- Diab Data Compiler 5.1.1

4 General Description

All GPIO functionality is provided by the SIU for MPC55xx. Each MPC55xx pad that has GPIO functionality has an associated pad configuration register (SIU_PCR*n*) in the SIU where the GPIO function is selected. The GPIO pads are typically multiplexed with other I/O pad functions. In addition, each MPC55xx pad with GPIO functionality has an input data register (SIU_GPDI*n*) and an output data register (SIU_GPDO*n*).

5 Quick Reference

This section is intended to be a source of quick access information; the API functions are described in detail in Section 6, “Detailed API Description”.

5.1 Required Header Files

The following header files are needed in order to use this peripheral software driver.

```
#include “siu_struct.h” /* header file for SIU module */
#include “fs_gpio.h”
```

5.2 Public Data Structure(s)

```
typedef struct
{
        uint16_t pad;    /* This is the pad number */
        uint16_t config;  /* This is the configuration value for the pad */
}fs_gpio_config_array_ts;
```

5.3 Configuration Items

Not applicable.

5.4 API Specification

Function arguments for each routine are described as **in**, **out**, or **inout**.

1. **in** argument means that the parameter value is an input only to the function.
2. **out** argument means that the parameter value is an output only from the function.
3. **inout** argument means that a parameter value is an input to the function, but the same parameter is also an output from the function.

NOTE

inout parameters are typically input pointer variables in which the caller passes the address of a pre-allocated data structure to a function. The function stores its results within that data structure. The actual value of the inout pointer parameter is not changed.

Table 1 lists the peripheral software driver routines.

Table 1. GPIO API

Routine	Description
void **fs_gpio_config_input**(uint16_t pad, uint16_t hysteresis, uint16_t weak_pull);	This function configures a pad for use as general purpose input.
void **fs_gpio_config_output**(uint16_t pad, uint16_t drive_strength, uint16_t slew_rate, uint16_t open_drain, uint16_t readback);	This function configures a pad for use as general purpose output.
void **fs_gpio_config**(uint16_t pad, uint16_t config);	This function configures a pad (a general configuration).
void **fs_gpio_config_array**(uint16_t size, fs_gpio_config_array_ts *config_array);	This function configures an array of pads (a general configuration).
uint8_t **fs_gpio_read_data**(uint16_t pad);	This function returns the current state of a pad.
void **fs_gpio_write_data**(uint16_t pad, uint8_t value);	This function sets the state of a pad.
uint8_t **fs_gpio_read_byte**(uint16_t pad_msb);	This function returns the current states of the pads composing a byte.
void **fs_gpio_write_byte**(uint16_t pad_msb, uint8_t value);	This function sets the states of the pads composing a byte.

6 Detailed API Description

The detailed functionality of all driver routines is explained in this section. The code examples illustrate the usage of the API.

6.1 fs_gpio_config_input

Configure a pad for use as general purpose input.

Call:

```
void fs_gpio_config_input(uint16_t pad, uint16_t hysteresis, uint16_t weak_pull );
```

Arguments:

pad	in	This is the pad number. The pad number will be in the range 0–511. Note that not all pad numbers in the range are implemented on silicon.
hysteresis	in	This defines if hysteresis should be enabled for the pad. Use FS_GPIO_HYSTERESIS_ENABLE to enable it, or FS_GPIO_HYSTERESIS_DISABLE to disable it.
weak_pull	in	This defines the weak pull device for the pad. It should be set to FS_GPIO_WEAK_PULL_UP, FS_GPIO_WEAK_PULL_DOWN, or FS_GPIO_WEAK_PULL_DISABLE.

Description:

Returns:	None
Range Issues:	None
Special Issues:	None
Implementation:	This routine is implemented as a function call.

Example:

```
fs_gpio_config_input( 179, FS_GPIO_HYSTERESIS_ENABLE, FS_GPIO_WEAK_PULL_UP );   /* EMIOS0 */
fs_gpio_config_input( FS_GPIO_EMIOS1, FS_GPIO_HYSTERESIS_DISABLE, FS_GPIO_WEAK_PULL_DISABLE );
                                                                                    /* 180 */
```

6.2 fs_gpio_config_output

Configure a pad for use as general purpose output.

Call:

```
void fs_gpio_config_output( uint16_t pad, uint16_t drive_strength, uint16_t slew_rate,
                                   uint16_t open_drain, uint16_t readback );
```

Arguments:

pad	in	This is the pad number. The pad number will be in the range 0–511. Note that not all pad numbers in the range are implemented on silicon.
drive_strength	in	This defines the drive strength. It should be set to FS_GPIO_DRIVE_STRENGTH_10PF, FS_GPIO_DRIVE_STRENGTH_20PF, FS_GPIO_DRIVE_STRENGTH_30PF or FS_GPIO_DRIVE_STRENGTH_50PF.
slew_rate	in	This defines the slew rate. It should be set to FS_GPIO_MINIMUM_SLEW_RATE, FS_GPIO_MEDIUM_SLEW_RATE or FS_GPIO_MAXIMUM_SLEW_RATE.
open_drain	in	This enables open drain mode. It should be set to FS_GPIO_OUTPUT_DRAIN_ENABLE to enable open drain mode or to FS_GPIO_OUTPUT_DRAIN_DISABLE to disable it.
readback	in	This enables readng of the actual value of the pad by fs_gpio_read_data() function. Use FS_GPIO_READBACK_ENABLE to enable such functionality, or FS_GPIO_READBACK_DISABLE to disable it and thus to reduce noise and power consumption.

Description:

This function configures a pad for use as general purpose output. It modifies the corresponding SIU_PCR*n* register according to the selected arguments when the GPIO function and the output buffer are always enabled.

Returns:	None
Range Issues:	None
Special Issues:	None
Implementation:	This routine is implemented as a function call.

Example:

```
fs_gpio_config_output( 183, FS_GPIO_DRIVE_STRENGTH_50PF, FS_GPIO_MEDIUM_SLEW_RATE,
                 FS_GPIO_OUTPUT_DRAIN_DISABLE, FS_GPIO_READBACK_DISABLE ); /* EMIOS4 */
fs_gpio_config_output( FS_GPIO_EMIOS5, FS_GPIO_DRIVE_STRENGTH_30PF, 0, 0,
                          FS_GPIO_READBACK_ENABLE );  /* 184 */
```

6.3 fs_gpio_config

Configure a pad (a general configuration).

Call:

```
void fs_gpio_config( uint16_t pad, uint16_t config );
```

Arguments:

pad	in	This is the pad number. The pad number will be in the range 0–511. Note that not all pad numbers in the range are implemented on silicon.
config	in	This is the configuration value for the pad. The best way to determine this is to add the appropriate configuration values together from the fs_gpio.h file.

Description:

This function configures a pad (a general configuration). It modifies the corresponding SIU_PCR*n* register according to the specified configuration value.

Returns:	None
Range Issues:	None
Special Issues:	User must be fully familiar with the GPIO configuration.
Implementation:	This routine is implemented as a function call.

Example:

```
fs_gpio_config( 186, FS_GPIO_IO_FUNCTION+FS_GPIO_OUTPUT_MODE+FS_GPIO_DRIVE_STRENGTH_50PF );
                                                                      /* EMIOS7 */
fs_gpio_config( FS_GPIO_EMIOS8, FS_GPIO_IO_FUNCTION | FS_GPIO_INPUT_MODE |
                 FS_GPIO_HYSTERESIS_ENABLE | FS_GPIO_WEAK_PULL_DISABLE ); /* 187 */

fs_gpio_config( 209, FS_GPIO_ALTERNATE_FUNCTION2 ); /* configured as SOUDT */
fs_gpio_config( 83, FS_GPIO_PRIMARY_FUNCTION ); /* configured as CNTXA */
```

6.4 fs_gpio_config_array

Configure an array of pads (a general configuration).

Call:

```
void fs_gpio_config_array( uint16_t size, fs_gpio_config_array_ts *config_array );
```

Arguments:

size	in	This the number of elements in the config_array, that is. the number of pads to be configured.
config_array	in	This is an array of pad and configuration data.

Description:

This function configures an array of pads (a general configuration). It modifies the corresponding SIU_PCR*n* registers according to the specified array of pads and configuration values.

Returns:	None
Range Issues:	None
Special Issues:	User must be fully familiar with the GPIO configuration.
Implementation:	This routine is implemented as a function call.

Example:

```
/* static structures for configuring several pins */
static fs_gpio_config_array_ts byte_in[] =
   { 114, FS_GPIO_IO_FUNCTION | FS_GPIO_INPUT_MODE | FS_GPIO_HYSTERESIS_ENABLE |
           FS_GPIO_WEAK_PULL_DISABLE, /* ETPUA0 */
     115, FS_GPIO_IO_FUNCTION | FS_GPIO_INPUT_MODE | FS_GPIO_HYSTERESIS_ENABLE |
           FS_GPIO_WEAK_PULL_DISABLE, /* ETPUA1 */
     116, FS_GPIO_IO_FUNCTION | FS_GPIO_INPUT_MODE | FS_GPIO_HYSTERESIS_ENABLE |
           FS_GPIO_WEAK_PULL_DISABLE, /* ETPUA2 */
     117, FS_GPIO_IO_FUNCTION | FS_GPIO_INPUT_MODE | FS_GPIO_HYSTERESIS_ENABLE |
           FS_GPIO_WEAK_PULL_DISABLE, /* ETPUA3 */
     118, FS_GPIO_IO_FUNCTION | FS_GPIO_INPUT_MODE | FS_GPIO_HYSTERESIS_ENABLE |
           FS_GPIO_WEAK_PULL_DISABLE, /* ETPUA4 */
     119, FS_GPIO_IO_FUNCTION | FS_GPIO_INPUT_MODE | FS_GPIO_HYSTERESIS_ENABLE |
           FS_GPIO_WEAK_PULL_DISABLE, /* ETPUA5 */
     120, FS_GPIO_IO_FUNCTION | FS_GPIO_INPUT_MODE | FS_GPIO_HYSTERESIS_ENABLE |
           FS_GPIO_WEAK_PULL_DISABLE, /* ETPUA6 */
     121, FS_GPIO_IO_FUNCTION | FS_GPIO_INPUT_MODE | FS_GPIO_HYSTERESIS_ENABLE |
           FS_GPIO_WEAK_PULL_DISABLE /* ETPUA7 */
   };

static fs_gpio_config_array_ts byte_out[] =
   { 122, FS_GPIO_IO_FUNCTION + FS_GPIO_OUTPUT_MODE + FS_GPIO_DRIVE_STRENGTH_50PF, /* ETPUA8 */
     123, FS_GPIO_IO_FUNCTION + FS_GPIO_OUTPUT_MODE + FS_GPIO_DRIVE_STRENGTH_50PF, /* ETPUA9 */
    124, FS_GPIO_IO_FUNCTION + FS_GPIO_OUTPUT_MODE + FS_GPIO_DRIVE_STRENGTH_50PF, /* ETPUA10 */
    125, FS_GPIO_IO_FUNCTION + FS_GPIO_OUTPUT_MODE + FS_GPIO_DRIVE_STRENGTH_50PF, /* ETPUA11 */
    126, FS_GPIO_IO_FUNCTION + FS_GPIO_OUTPUT_MODE + FS_GPIO_DRIVE_STRENGTH_50PF, /* ETPUA12 */
    127, FS_GPIO_IO_FUNCTION + FS_GPIO_OUTPUT_MODE + FS_GPIO_DRIVE_STRENGTH_50PF, /* ETPUA13 */
    128, FS_GPIO_IO_FUNCTION + FS_GPIO_OUTPUT_MODE + FS_GPIO_DRIVE_STRENGTH_50PF, /* ETPUA14 */
     129, FS_GPIO_IO_FUNCTION + FS_GPIO_OUTPUT_MODE + FS_GPIO_DRIVE_STRENGTH_50PF /* ETPUA15 */
   };

fs_gpio_config_array( 8, byte_in );
fs_gpio_config_array( 8, byte_out );
```

6.5 fs_gpio_read_data

Return the current state of a pad.

Call:

```
uint8_t fs_gpio_read_data( uint16_t pad );
```

Arguments:

pad	in	This is the pad number. The pad number will be between 0-511. Note that not all pad numbers in the range are implemented on silicon.

Description:

This function returns the current state the of a pad. It reads the corresponding SIU_GPDIn register.

Returns:	State of the specified pad as uint8_t.
Range Issues:	None
Special Issues:	This function assumes that the pad is already configured for input.
Implementation:	This routine is implemented as a function call.

Example:

```
uint8_t a;

a = fs_gpio_read_data(179);
a = fs_gpio_read_data(FS_GPIO_EMIOS1);
```

6.6 fs_gpio_write_data

Set the state of a pad.

Call:

```
void fs_gpio_write_data( uint16_t pad, uint8_t value );
```

Arguments:

pad	in	This is the pad number. The pad number will be in the range 0–511. Note that not all pad numbers in the range are implemented on silicon.
value	in	This is the value (0 or 1) to be written to the output pad.

Description:

This function sets the state of a pad. It writes the specified argument value to the corresponding SIU_GPDOn register.

Returns:	None
Range Issues:	None
Special Issues:	This function assumes that the pad is already configured for output.
Implementation:	This routine is implemented as a function call.

Example:

```
uint8_t a;
a = 1;

fs_gpio_write_data( 183, a );
fs_gpio_write_data( 183, 0 );

fs_gpio_write_data( FS_GPIO_EMIOS5, 1 );
fs_gpio_write_data( FS_GPIO_EMIOS5, 0 );

fs_gpio_write_data( FS_GPIO_EMIOS5, !fs_gpio_read_data(FS_GPIO_EMIOS5) );
```

6.7 fs_gpio_read_byte

Return the current states the of the pads composing a byte.

Call:

```
uint8_t fs_gpio_read_data( uint16_t pad );
```

Arguments:

pad_msb	in	This is the pad number of the MSB bit from a byte. The pad number will be in the range 0–504. Note that not all pad numbers in the range are implemented on silicon.

Description:

This function returns the current states the of the pads forming a byte. It reads the corresponding SIU_GPDIn registers.

Returns:	State of the eight subsequent pads as uint8_t.
Range Issues:	Pads forming a byte must be continuous.
Special Issues:	This function assumes that the pads are already configured for input.
Implementation:	This routine is implemented as a function call.

Example:

```
uint8_t c;

c = fs_gpio_read_byte( 114 );
c = fs_gpio_read_byte( byte_in[0].pad );
```

6.8 fs_gpio_write_byte

Set the states of the pads composing a byte.

Call:

```
void fs_gpio_write_byte( uint16_t pad_msb, uint8_t value );
```

Arguments:

pad_msb	in	This is the pad number. The pad number will be in the range 0–504. Note that not all pad numbers in the range are implemented on silicon.
value	in	This is the 8-bit (= 1 byte) value to be written to the outputs.

Description:

This function sets the states of the pads forming a byte. It writes the specified argument value to the corresponding SIU_GPDOn registers.

Returns:	None
Range Issues:	Pads forming a byte must be continuous.
Special Issues:	This function assumes that the pads are already configured for output.
Implementation:	This routine is implemented as a function call.

Example:

```
uint8_t b;

fs_gpio_write_byte( 122, 0xe0 );

b = 0x5a;
fs_gpio_write_byte( 122, b );

fs_gpio_write_byte( byte_out[0].pad, 0xf1 );
```

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN2855
Rev. 0
2/2008