SB70LC GPIO Configuration

Application Note

Revision 1.0
October 29, 2009
Document Status: Initial Release

Table of Contents

Introduction

Most of the Freescale MCF5270 processor pins can be configured as general purpose input/output (GPIO) pins. These pins have multiple functions, and the specific pins that make sense to use as GPIO will depend on which peripherals are needed for the targeted application. For example, the QSPI pins can be used for their primary function, or they can be configured as GPIO. This document will examine each pin and show how it can be configured as GPIO. Additional information contained in this document can be found in "Chapter 12 – General Purpose I/O Module" of the MCF5271 reference manual.

Electrical Specifications

The current drive capabilities of the GPIO pins are the same for all pins. The instantaneous maximum current for a single pin is 25 mA. The sustained current drive is 5 mA. Please see the "MCF5271 Integrated Microprocessor Hardware Specification" PDF document for more information, which can be found at the Freescale web site.

Pin Assignment and GPIO Control Registers

Any pin that can be used for GPIO functionality is configurable with the following five registers: a port output data register, a port data direction register, a port pin data/set data register, a port clear output data register, and a pin assignment register. All GPIO pins mentioned here have a sixth register called a drive strength control register that allows control of the current to be driven high or low on the pins by setting the corresponding bits.

To access the pin assignment and GPIO control registers, the following preprocessor directive must be included:

```
#include <sim5270.h>
```

In the SIM header file (\Nburn\SB70LC\include\sim5270.h), GPIO registers are grouped into a struct type definition called gpiostruct. Each GPIO register is prefixed with a tag to identify whether it is a pin assignment, port output data, port data direction, port pin data/set data, port clear output data, or drive strength control register (additional information on each type of register can be found in the following sections). The following table provides the name of the prefix tags and the full name of the register type associated with each tag:

Tag Prefix	Type of Register
`par`	Pin Assignment Register
`podr`	Port Output Data Register
`pddr`	Port Data Direction Register
`ppdsdr`	Port Pin Data/Set Data Register
`pclrr`	Port Clear Output Data Register
`dscr`	Drive Strength Control Register

Pin Assignment Register (PAR)

The pin assignment register controls the functionality of the pins. Pins can have either one or two assigned bits for configuration. One-bit assignments allow pins to be configured between their primary function and GPIO, and two-bit assignments allow pins to be configured between their primary function, alternate function, and GPIO (some may also have a second alternate function). Setting a '0' (1-bit) or "00" (2-bit) will configure any pin for GPIO, while a '1' or "11" will configure any pin for their primary function.

In the charts that follow for each port (pin groups), the pin assignment bits column helps identify the bits in the pin assignment register that control the corresponding pin's functionality. Pin assignment registers can be eight bits wide or sixteen bits wide.

Port Data Direction Register (PDDR)

The port data direction registers control the direction of the pins when they are configured for GPIO. The registers are eight bits wide, but not all ports mentioned here will use all eight bits.

The registers are read/write capable. At reset, all bits in the PDDR registers are cleared. Setting any bit to '1' in a PDDR register configures the corresponding GPIO pin as an output. Setting any bit to '0' in a PDDR register configures the corresponding pin as an input.

Port Output Data Register (PODR)

The port output data registers store the data to be driven on the corresponding port pins when the pins are configured for general purpose output. The registers are each eight bits wide, but not all of them use all eight bits.

The registers are read/write capable. At reset, all implemented bits in the PODR registers are set. Reserved bits always remain cleared. Reading a PODR register returns the current values in the register. To set bits in a PODR register, write '1' to the bits, or write '1' to the corresponding bits in the port pin data/set data register. To clear bits in a PODR register, write '0' to the bits, or write '0' to the corresponding bits in the port clear output data register.

Port Pin Data/Set Data Register (PPDSDR)

The port pin data/set data register reflects the current pin states and control the setting of output pins when the pins are configured for GPIO. The registers are each eight bits wide, but not all ports mentioned here will use all eight bits.

The registers are read/write capable. At reset, the bits in the PPDSDR registers are set to the current pin states. Reading a PPDSDR register returns the current state of the associated pins. Setting bits in this register sets the corresponding bits in the port output data register. Writing a '0' has no effect.

Port Clear Output Data Register (PCLRR)

The port clear output data register clears the corresponding bits in the port output data register. The registers are each eight bits wide, but not all ports mentioned here will use all eight bits.

The registers are read/write capable. Setting it has no effect. Writing '0' to bits in this register clears the corresponding bits in the port output data register. Reading the PCLRR register returns zeros.

Drive Strength Control Register (DSCR)

The drive strength control register sets the output pin drive strength of the current. All drive strength control registers are read/write capable. For more information on how to configure and use this register for each port, please refer to the Freescale MCF5271 reference manual, section 12.3.1.7.

Timer Port

The timer pin assignment register (PAR_TIMER) controls the functions of the timer pins (MCF5271 reference manual, table 12-17). The GPIO control bits correspond to the bit positions in the GPIO control registers (PDDR_TIMER, PODR_TIMER, PPDSDR_TIMER, and PCLRR_TIMER).

SB70LC Pin No.	SB70LC Signal Name	Signal Description	Pin Assign Bit(s) [PAR_TIMER]	GPIO Control Bit [***_TIMER]
JP1-15	TIN3	Timer Input 3	15-14	7

Code examples:

```
// Timer Pin Assignment Register – Set TIN3 as GPIO.
sim.gpio.par_timer &= ~PAR_TIN3_BIT;

// Timer Port Data Direction Register – Set TIN3 as a GPIO output.
sim.gpio.pddr_timer |= CTRL_TIN3_BIT;

// Timer Port Output Data Register – Set GPIO signal TIN3 high.
sim.gpio.podr_timer |= CTRL_TIN3_BIT;

// Timer Port Pin Data/Set Data Register – Read the current pin state
// value for GPIO signal TIN3.
BYTE value_timer = sim.gpio.ppdsdr_timer & CTRL_TIN3_BIT;

// Timer Drive Strength Control Register – Set a high output drive
// strength (writing to bit 0 affects all timer pins configured for
// GPIO).
sim.gpio.dscr_timer |= 0x01;
```

Reference:

```
#define PAR_TIN3_BIT    ( 0xC000 )   // 1100 0000 0000 0000
#define CTRL_TIN3_BIT   (   0x80 )   //           1000 0000
```

Inter-Integrated Circuit (I^2C) Port

The inter-integrated circuit pin assignment register (PAR_FECI2C) controls the functions of the I^2C pins (MCF5271 reference manual, table 12-14). The GPIO control bits correspond to bit positions in the GPIO control registers (PDDR_FECI2C, PODR_FECI2C, PPDSDR_FECI2C, and PCLRR_FECI2C).

SB70LC Pin No.	SB70LC Signal Name	Signal Description	Pin Assign Bit(s) [PAR_FECI2C]	GPIO Control Bit [***_FECI2C]
JP1-16	SDA	I^2C Serial Data	1-0	0
JP1-17	SCL	I^2C Serial Clock	3-2	1

Code examples:

```
// I2C Pin Assignment Register – Set SDA and SCL as GPIO.
sim.gpio.par_feci2c &= ~( PAR_SDA_BIT | PAR_SCL_BIT );

// I2C Port Data Direction Register – Set SDA as a GPIO output and SCL
// as a GPIO input.
sim.gpio.pddr_feci2c |= CTRL_SDA_BIT;
sim.gpio.pddr_feci2c &= ~CTRL_SCL_BIT;

// I2C Port Output Data Register – Set GPIO signal SDA low.
sim.gpio.podr_feci2c |= CTRL_SDA_BIT;

// I2C Port Pin Data/Set Data Register – Read the current pin state
// value for GPIO signal SCL.
BYTE value_feci2c = sim.gpio.ppdsdr_feci2c & CTRL_SCL_BIT;

// I2C Drive Strength Control Register – Set a low output drive
// strength (writing to bit 0 affects all I2C pins configured for
// GPIO).
sim.gpio.dscr_feci2c &= ~0x01;
```

Reference:

```
#define PAR_SDA_BIT    ( 0x03 )   // 0000 0011
#define PAR_SCL_BIT    ( 0x0C )   // 0000 1100
#define CTRL_SDA_BIT   ( 0x01 )   // 0000 0001
#define CTRL_SCL_BIT   ( 0x02 )   // 0000 0010
```

Queued Serial Peripheral Interface (QSPI) Port

The queued serial peripheral interface pin assignment register (PAR_QSPI) controls the functions of the QSPI pins (MCF5271 reference manual, table 12-16). The GPIO control bits correspond to bit positions in the GPIO control registers (PDDR_QSPI, PODR_QSPI, PPDSDR_QSPI, and PCLRR_QSPI).

SB70LC Pin No.	SB70LC Signal Name	Signal Description	Pin Assign Bit(s) [PAR_QSPI]	GPIO Control Bit [***_QSPI]
JP1-7	SPI_CLK	SPI Clock	1-0	2
JP1-4	SPI_DOUT	SPI Data Out	2	0
JP1-6	SPI_DIN	SPI Data In	4-3	1
JP1-3	SPI_CS0	SPI Chip Select 0	5	3

Code examples:

```
// QSPI Pin Assignment Register – Set SPI_CLK and SPI_DIN as GPIO.
sim.gpio.par_qspi &= ~( PAR_SPI_CLK_BIT | PAR_SPI_DIN_BIT );

// QSPI Port Data Direction Register – Set SPI_CLK and SPI_DIN as GPIO
// inputs.
sim.gpio.pddr_qspi &= ~( CTRL_SPI_CLK_BIT | CTRL_SPI_DIN_BIT );

// QSPI Port Output Data Register – Set GPIO signal SPI_CS0 high.
sim.gpio.podr_qspi |= CTRL_SPI_CS0_BIT;

// QSPI Port Pin Data/Set Data Register – Read the current pin state
// values for GPIO signals SPI_CLK and SPI_DIN.
BYTE value_qspi = sim.gpio.ppdsdr_qspi & ( CTRL_SPI_CLK_BIT |
                                           CTRL_SPI_DIN_BIT );

// QSPI Drive Strength Control Register – Set a high output drive
// strength (writing to bit 0 affects all QSPI pins configured for
// GPIO).
sim.gpio.dscr_qspi |= 0x01;
```

Reference:

```
#define PAR_SPI_CLK_BIT     ( 0x03 )   // 0000 0011
#define PAR_SPI_DOUT_BIT    ( 0x04 )   // 0000 0100
#define PAR_SPI_DIN_BIT     ( 0x18 )   // 0001 1000
#define PAR_SPI_CS0_BIT     ( 0x20 )   // 0010 0000
#define CTRL_SPI_CLK_BIT    ( 0x04 )   // 0000 0100
#define CTRL_SPI_DOUT_BIT   ( 0x01 )   // 0000 0001
#define CTRL_SPI_DIN_BIT    ( 0x02 )   // 0000 0010
#define CTRL_SPI_CS0_BIT    ( 0x08 )   // 0000 1000
```

Universal Asynchronous Receive and Transmit (UART) Port

The universal asynchronous receive and transmit pin assignment register (PAR_UART) controls the functions of the UART pins (MCF5271 reference manual, table 12-15). The GPIO control bits correspond to bit positions in the GPIO control registers (PDDR_UARTL, PODR_UARTL, PPDSDR_UARTL, and PCLRR_UARTL).

SB70LC Pin No.	SB70LC Signal Name	Signal Description	Pin Assign Bit(s) [PAR_UART]	GPIO Control Bit [***_UARTL]
JP1-5	U0_3VRTS	UART 0 Request to Send	0	2
JP1-14	U0_3VCTS	UART 0 Clear to Send	1	3
JP1-10	U0_3VTX	UART 0 Transmit	2	1
JP1-11	U0_3VRX	UART 0 Receive	3	0
JP1-9	U1_3VRTS	UART 1 Request to Send	5-4	6
JP1-8	U1_3VCTS	UART 1 Clear to Send	7-6	7
JP1-12	U1_3VTX	UART 1 Transmit	9-8	5
JP1-13	U1_3VRX	UART 1 Receive	11-10	4

Code examples:

```
// UART Pin Assignment Register – Set U0_3VRTS as GPIO.
sim.gpio.par_uart &= ~PAR_U0_3VRTS_BIT;

// UART Port Data Direction Register – Set U0_3VRTS as a GPIO output.
sim.gpio.pddr_uartl |= CTRL_U0_3VRTS_BIT;

// UART Port Output Data Register – Set GPIO signal U0_3VRTS low.
sim.gpio.podr_uartl &= ~CTRL_U0_3VRTS_BIT;

// UART Port Pin Data/Set Data Register – Read the current pin state
// value for GPIO signal U0_3VRTS.
BYTE value_uartl = sim.gpio.ppdsdr_uartl & CTRL_U0_3VRTS_BIT;

// UART Drive Strength Control Register – Set a high output drive
// strength for UART 0 pins, and a low output drive strength for UART 1
// pins (writing to bit 0 and bit 2 affects all UART 0 and UART 1 pins
// configured for GPIO, respectively).
sim.gpio.dscr_uart |= 0x01;
sim.gpio.dscr_uart &= ~0x04;
```

Reference:

```
#define PAR_U0_3VRTS_BIT    ( 0x0001 )   // 0000 0000 0000 0001
#define PAR_U0_3VCTS_BIT    ( 0x0002 )   // 0000 0000 0000 0010
#define PAR_U0_3VTX_BIT     ( 0x0004 )   // 0000 0000 0000 0100
#define PAR_U0_3VRX_BIT     ( 0x0008 )   // 0000 0000 0000 1000
#define PAR_U1_3VRTS_BIT    ( 0x0030 )   // 0000 0000 0011 0000
#define PAR_U1_3VCTS_BIT    ( 0x00C0 )   // 0000 0000 1100 0000
#define PAR_U1_3VTX_BIT     ( 0x0300 )   // 0000 0011 0000 0000
#define PAR_U1_3VRX_BIT     ( 0x0C00 )   // 0000 1100 0000 0000
```

```
#define CTRL_U0_3VRTS_BIT   (   0x04 )   //          0000 0100
#define CTRL_U0_3VCTS_BIT   (   0x08 )   //          0000 1000
#define CTRL_U0_3VTX_BIT    (   0x02 )   //          0000 0010
#define CTRL_U0_3VRX_BIT    (   0x01 )   //          0000 0001
#define CTRL_U1_3VRTS_BIT   (   0x40 )   //          0100 0000
#define CTRL_U1_3VCTS_BIT   (   0x80 )   //          1000 0000
#define CTRL_U1_3VTX_BIT    (   0x20 )   //          0010 0000
#define CTRL_U1_3VRX_BIT    (   0x10 )   //          0001 0000
```

Programming Multiple Pins Simultaneously

Programming multiple pins in one line of code is possible as long as they belong to the same port. For example, you can program all four pins associated with the QSPI port at once, but you cannot program a pin from the QSPI port and a pin from the I^2C port at once. The following example shows how this is done with the QSPI port (use the QSPI port chart to see what pins are affected by the bit configurations) :

SB70LC Pin No.	SB70LC Signal Name	Signal Description	Pin Assign Bit(s) [PAR_QSPI]	GPIO Control Bit [***_QSPI]
JP1-7	SPI_CLK	SPI Clock	1-0	2
JP1-4	SPI_DOUT	SPI Data Out	2	0
JP1-6	SPI_DIN	SPI Data In	4-3	1
JP1-3	SPI_CS0	SPI Chip Select 0	5	3

```
// QSPI Pin Assignment Register - Set all four pins as GPIO (bits 6 and
// 7 are irrelevant).
sim.gpio.par_qspi = 0xC0;   // 1100 0000

// QSPI Port Data Direction Register – Set SPI_CLK as a GPIO output,
// and set the rest of the signals as GPIO inputs (bits 4, 5, 6, and 7
// are irrelevant).
sim.gpio.pddr_qspi = 0x04;   // 0000 0100
```

In the examples above, all four pins belonging to the QSPI port were programmed for use as GPIO pins simultaneously, followed by configuring the SPI_CLK as a GPIO output and the rest as GPIO inputs simultaneously. Overall, configuring four GPIO pins and their signal directions were done in only two lines of code.