SBL2E PinIO Class

Application Note

Revision 1.0
March 31, 2011
Document Status: Initial Release

Table of Contents

Introduction

The PinIO class provides an easy way to configure and operate the Freescale MCF52236 microprocessor GPIO signals. Each signal pin on the processor can have multiple functions. You can use the PinIO class to control GPIO signals without having to explicitly configure the processor registers. Configuration of the processor registers are done in the member functions of the PinIO class. There are twelve pins on the SBL2E that are made available for GPIO. This document will list the pins that can be used for GPIO and how to use them. Note that the terms "PinIO class" and "Pins class" may be used interchangeably in this document.

If you do wish to access these registers directly, we recommend you use the GPIO register structure defined in the "sim52234.h" header file of the \Nburn\<platform>\include directory and use the Freescale MCF52235 reference manual (Chapter 14 – General Purpose I/O Module) to learn the operation of each register.

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 depends on each pin's drive strength configuration through the pin drive strength register of the GPIO module (see Chapter 14.6.5.4 of the MCF52235 reference manual). If a pin's drive strength control bit is set low, then it is 2 mA. If its respective control bit is set high, then it is 10 mA.

PinIO Class

This class is defined in the header file "pins.h", which is located in the \Nburn\include_sc directory. With this class, the pins can be configured for GPIO or some other function. If the pins are set for GPIO, then you can set, clear, read the state of the pins, drive the pins, or set them for high impedance by simply using the appropriate member function.

Since the number and type of pins are unique to each NetBurner module, the definition of the pins (\Nburn\<platform>\include\pinconstant.h) and the functions to use those pins (\Nburn\<platform>\system\pins.cpp) are located within each applicable platform directory. The numbering system in "pins.cpp" are listed according to the MCF52236's processor pin numbers, so the "PinMap" array constant in "pin_map.cpp" is used to map the SBL2E's JP1 header pins to the MCF52236's processor pins (index zero in the array does not correlate to any pin number, so this is reserved with a zero value to indicate no mapped connection).

Pins Class Constants

The table below lists the fifteen pins available for GPIO on the SBL2E, as well as their primary and alternate functions, if any:

Pin	Definition	Function
1	PIN1_UTXD0 PIN1_GPIO	1: UART 0 - Transmit 0: GPIO
2	PIN2_URXD0 PIN2_GPIO	1: UART 0 - Receive 0: GPIO
3	PIN3_URTS0 PIN3_GPIO	1: UART 0 – Request to Send 0: GPIO
4	PIN4_UCTS0 PIN4_GPIO	1: UART 0 – Clear to Send 0: GPIO
7	PIN7_AN0 PIN7_GPIO	1: Analog Input 0 0: GPIO
8	PIN8_AN1 PIN8_GPIO	1: Analog Input 1 0: GPIO
9	PIN9_AN2 PIN9_GPIO	1: Analog Input 2 0: GPIO
10	PIN10_AN3 PIN10_GPIO	1: Analog Input 3 0: GPIO
12	PIN12_URXD1 PIN12_GPIO	1: UART 0 - Receive 0: GPIO
13	PIN13_UTXD1 PIN13_GPIO	1: UART 1 - Transmit 0: GPIO
14	PIN14_SCL PIN14_CANTX PIN14_UTXD2 PIN14_GPIO	1: I2C Serial Clock 2: CAN – Transmit 3: UART 2 - Transmit 0: GPIO
15	PIN15_SDA PIN15_CANRX PIN15_URXD2 PIN15_GPIO	1: I2C Serial Data 2: CAN – Receive 3: UART 2 - Receive 0: GPIO

Pin Constants Table

The "Definition" column in the table above describes the values available for each pin when used with the PinIO class member function "function". For example, if pin JP1-15 needs to be configured for GPIO, then it would be written as:

```
Pins[15].function( PIN15_GPIO );
```

Or, if the I^2C serial data signal functionality is needed, then it would be written as:

```
Pins[15].function( PIN17_SDA );
```

The "Function" column in the table describes the primary, alternate, and GPIO functions for each pin. The numbers to the left represent the following:

0 = GPIO
1 = Primary Function
2 = Alternate Function 1 (applicable to select pins)
3 = Alternate Function 2 (applicable to select pins)

Pins Class Member Functions

Using the Pins class member functions to configure and use the GPIO pins eliminates the time and complexity of having to look up the proper documentation and use the right register and bits for a desired pin or set of pins. For example, if one were to use pin JP1-12 (UART 1 – Receive) for GPIO and set it high without the PinIO class, then it would be written like this:

```
#include <sim52234.h>

sim.gpio.pubpar &= ~0x0C;    // Configure pin JP1-12 for GPIO
sim.gpio.setub = 0x02;       // Set bit to be driven out on pin
sim.gpio.ddrub |= 0x02;      // Set signal direction as output
```

Knowing the right register and bits are not required with the PinIO class, thus making it more convenient:

```
#include <pins.h>

Pins[12].function( PIN12_GPIO ); // Configure pin JP1-12 for GPIO
Pins[12] = 1;                    // Set pin as output high
```

The following lists the member functions that can be used with the PinIO class:

Member Function Name	Description	Example
`void set()`	Set output high	`Pins[7].set();` `Pins[7] = 1;`
`void clr()`	Set output low	`Pins[4].clr();` `Pins[4] = 0;`

`BOOL read()`	Read pin high/low state	`BOOL bpinstate = Pins[13];`
		`if ( !Pins[13] )`
		`  iprintf ( "The pin is low" );`

`void hiz()`	Set output to tri-state (high impedance input)	`Pins[9].hiz();`
`void drive`	Turn output on (opposite of tri-state)	`Pins[15].drive();`
`void function()`	Set pin to special function or GPIO	`Pins[3].function( PIN3_URTS0 );` `Pins[3].function( PIN3_GPIO );`

Program Examples

```
/*****************************************************************
 * SIMPLE ALTERNATING HIGH/LOW OUTPUT PIN:
 *
 * This program configures pin JP1-15 as GPIO output. In an infinite
 * loop, alternating high and low signals are driven out on the pin
 * every second. The change in state of the pin can be confirmed by
 * using a multimeter, oscilloscope, or connecting an LED between
 * JP1-15 and ground. Another purpose for this example is to
 * demonstrate the usage of the set() and clr() functions. In the
 * next example, assigning '1' and '0' in place of set() and clr()
 * are used respectively, but basically perform the same function.
 *
 * Note:  The SBL2E module's JP1 pins are accessible through the
 * adapter board's J8, J9, and J10 headers.
 *****************************************************************/

#include "predef.h"
#include <stdio.h>
#include <ctype.h>
#include <basictypes.h>
#include <serialirq.h>
#include <system.h>
#include <constants.h>
#include <ucos.h>
#include <netif.h>
#include <autoupdate.h>
#include <pins.h>

extern "C" {
   void UserMain( void *pd );
}

const char *AppName = "SBL2E-PinsClassExample";

/* The main task. */
void UserMain( void *pd ) {
   SimpleUart( 0, SystemBaud );
   assign_stdio( 0 );
   InitializeStack();
   OSChangePrio( MAIN_PRIO );
   EnableAutoUpdate();

   iprintf( "Application started\r\n" );

   Pins[15].function( PIN15_GPIO );   // Configure JP1-15 for GPIO

   while ( 1 )
   {
      OSTimeDly( 1 * TICKS_PER_SECOND );
      Pins[15].set();                     // Set pin high
      OSTimeDly( 1 * TICKS_PER_SECOND );
      Pins[15].clr();                     // Set pin low
   }
}
```

```
/*********************************************************************
 * SENDING SIGNALS FROM AN OUTPUT PIN TO AN INPUT PIN:
 *
 * This program configures pins JP1-14 and JP1-15 as GPIO output and
 * GPIO input, respectively. In order for this program to properly
 * work, a jumper wire is needed to connect JP1-14 and JP1-15 together
 * on the SBL2E module.
 *
 * In an infinite loop, alternating high and low signals are driven
 * out on JP1-14, where JP1-15 will then be read. If the signal read
 * from JP1-15 is high, then the message "Hit!" will be outputted
 * through the serial port to MTTTY. If the signal read from JP1-15
 * is low, then the message "Miss!" will be outputted. After each
 * send/read, there is a one-second delay.
 *
 * Note:  The SBL2E module's JP1 pins are also accessible through the
 * adapter board's J8, J9, and J10 headers.
 *********************************************************************/

#include "predef.h"
#include <stdio.h>
#include <ctype.h>
#include <basictypes.h>
#include <serialirq.h>
#include <system.h>
#include <constants.h>
#include <ucos.h>
#include <netif.h>
#include <autoupdate.h>
#include <pins.h>

extern "C" {
   void UserMain( void *pd );
}

const char *AppName = "SBL2E-PinsClassExample2";

/* The main task. */
void UserMain( void *pd ) {
   SimpleUart( 0, SystemBaud );
   assign_stdio( 0 );
   InitializeStack();
   OSChangePrio( MAIN_PRIO );
   EnableAutoUpdate();

   iprintf( "Application started\r\n" );

   Pins[14].function( PIN14_GPIO );   // Configure JP1-14 for GPIO
   Pins[15].function( PIN15_GPIO );   // Configure JP1-15 for GPIO

   while ( 1 )
   {
      OSTimeDly( 1 * TICKS_PER_SECOND );

      Pins[14] = 1;               // Set JP1-14 output high
      if ( Pins[15] )             // Read JP1-15 input pin state
```

```
            iprintf( "Hit!\r\n" );
        else
            iprintf( "Miss!\r\n" );

        OSTimeDly( 1 * TICKS_PER_SECOND );

        Pins[14] = 0;               // Set JP1-14 output low
        if ( Pins[15] )             // Read JP1-15 input pin state
            iprintf( "Hit!\r\n" );
        else
            iprintf( "Miss!\r\n" );
    }
}
```