Mod5213 PinIO Class

Application Note

Revision 1.2
February 28, 2005
Document Status: Third Release

Table of Contents

Introduction

The PinIO Class provides an easy way to configure and operate the Freescale MCF5213 microprocessor GPIO signals. Each signal pin on the 5213 can have multiple functions. You can use the PinIO Class to control GPIO signals without having to explicitly configure the 5213 registers. Configuration of the processor registers are done in the member functions of the PinIO class. There are 33 pins on the Mod5213 that are made available for GPIO. This document will list the pins that can be used for GPIO and how to use them.

If you do wish to access these registers directly, we recommend you use the register structure defined in sim5213.h and use the Freescale MCF5213 reference manual to learn the operation of each register.

PinIO Class

This class is defined in the header file "pins.h" located in the \nburn\include_nn 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.

Pin Class Constants

The table below lists the 33 pins available for GPIO on the Mod5213, as well as their primary and alternate functions, if any:

Pin	Definition	Function
2	PIN2_UART0_RX PIN2_GPIO	1: UART 0 Receive 0: GPIO
3	PIN3_UART0_TX PIN3_GPIO	1: UART 0 Transmit 0: GPIO
4	PIN4_SDA PIN4_CANRX PIN4_UART2_RX PIN4_GPIO	1: I2C Serial Data 2: CAN Receive 3: UART 2 Receive 0: GPIO
5	PIN5_SCL PIN5_CANTX PIN5_UART2_TX PIN5_GPIO	1: I2C Serial Clock 2: CAN Transmit 3: UART 2 Transmit 0: GPIO

6	PIN6_IRQ1 PIN6_SYNCA PIN6_PWM1 PIN6_GPIO	1: External Interrupt 1 2: External Timer Clock Input A 3: Pulse Width Modulation Channel 1 0: GPIO
7	PIN7_IRQ4 PIN7_GPIO	1: External Interrupt 4 0: GPIO
8	PIN8_IRQ7 PIN8_GPIO	1: External Interrupt 7 0: GPIO
11	PIN11_AN2 PIN11_GPIO	1: Analog Input 2 0: GPIO
12	PIN12_AN1 PIN12_GPIO	1: Analog Input 1 0: GPIO
13	PIN13_AN0 PIN13_GPIO	1: Analog Input 0 0: GPIO
14	PIN14_AN3 PIN14_GPIO	1: Analog Input 3 0: GPIO
15	PIN15_AN7 PIN15_GPIO	1: Analog Input 7 0: GPIO
16	PIN16_AN6 PIN16_GPIO	1: Analog Input 6 0: GPIO
17	PIN17_AN5 PIN17_GPIO	1: Analog Input 5 0: GPIO
18	PIN18_AN4 PIN18_GPIO	1: Analog Input 4 0: GPIO
21	PIN21_DTIN3 PIN21_DTOUT3 PIN21_PWM6 PIN21_GPIO	1: DMA Timer Input 3 2: DMA Timer Output 3 3: Pulse Width Modulation Channel 6 0: GPIO
22	PIN22_DTIN2 PIN22_DTOUT2 PIN22_PWM4 PIN22_GPIO	1: DMA Timer Input 2 2: DMA Timer Output 2 3: Pulse Width Modulation Channel 4 0: GPIO
23	PIN23_DTIN1 PIN23_DTOUT1 PIN23_PWM2 PIN23_GPIO	1: DMA Timer Input 1 2: DMA Timer Output 1 3: Pulse Width Modulation Channel 2 0: GPIO
24	PIN24_DTIN0 PIN24_DTOUT0 PIN24_PWM0 PIN24_GPIO	1: DMA Timer Input 0 2: DMA Timer Output 0 3: Pulse Width Modulation Channel 0 0: GPIO
25	PIN25_GPT3 PIN25_PWM7 PIN25_GPIO	1: General Purpose Timer 3 3: Pulse Width Modulation Channel 7 0: GPIO
26	PIN26_GPT2 PIN26_PWM5 PIN26_GPIO	1: General Purpose Timer 2 3: Pulse Width Modulation Channel 5 0: GPIO
27	PIN27_GPT1 PIN27_PWM3 PIN27_GPIO	1: General Purpose Timer 1 3: Pulse Width Modulation Channel 3 0: GPIO
28	PIN28_GPT0 PIN28_PWM1 PIN28_GPIO	1: General Purpose Timer 0 3: Pulse Width Modulation Channel 1 0: GPIO
29	PIN29_UART1_RX PIN29_GPIO	1: UART 1 Receive 0: GPIO

30	PIN30_UART1_TX PIN30_GPIO	1: UART 1 Transmit 0: GPIO
31	PIN31_UART1_CTS PIN31_SYNCA PIN31_UART2_RX PIN31_GPIO	1: UART 1 Clear to Send 2: External Timer Clock Input A 3: UART 2 Receive 0: GPIO
32	PIN32_UART1_RTS PIN32_SYNCB PIN32_UART2_TX PIN32_GPIO	1: UART 1 Request to Send 2: External Timer Clock Input B 3: UART 2 Transmit 0: GPIO
33	PIN33_QSPI_CS2 PIN33_GPIO	1: QSPI Chip Select 2 0: GPIO
34	PIN34_QSPI_CS1 PIN34_GPIO	1: QSPI Chip Select 1 0: GPIO
35	PIN35_QSPI_CS0 PIN35_SDA PIN35_UART1_CTS PIN35_GPIO	1: QSPI Chip Select 0 2: I2C Serial Data 3: UART 1 Clear to Send 0: GPIO
36	PIN36_QSPI_DOUT PIN36_CANTX PIN36_UART1_TX PIN36_GPIO	1: QSPI Data Out 2: CAN Transmit 3: UART 1 Transmit 0: GPIO
37	PIN37_QSPI_DIN PIN37_CANRX PIN37_UART1_RX PIN37_GPIO	1: QSPI Data In 2: CAN Receive 3: UART 1 Receive 0: GPIO
38	PIN38_QSPI_CLK PIN38_SCL PIN38_UART1_RTS PIN38_GPIO	1: QSPI Clock 2: I2C Serial Clock 3: UART 1 Request to Send 0: GPIO

Pin Constants Table

The Definition column in the Pin Constants Table describes the values available for each pin when used with the PinIO class member function "function". For example, if pin 38 needs to be configured for GPIO it would be written as:

```
Pins[5].function( PIN5_GPIO );
```

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

```
Pins[5].function( PIN5_SCL );
```

The Function column in the Pin Constants 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
3: Alternate Function 2

Pin Class Member Functions

Using the Pin 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 30 (UART 1 – Transmit) for GPIO and set it high without the PinIO class, then it would be written like this:

```
#include <..\MOD5213\system\sim5213.h>

sim.gpio.pubpar &= ~0x03;    // Configure pin 30 for GPIO
sim.gpio.setub = 0x01;       // Set bit to be driven out on pin
sim.gpio.ddrub |= 0x01;      // 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[30].function( PIN30_GPIO );    // Configure pin 30 for GPIO
Pins[30] = 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[38].set(); Pins[38] = 1;
void clr()	Set output low	Pins[4].clr(); Pins[4] = 0;
BOOL read()	Read pin high/low state	BOOL bpinstate = Pins[30]; if (!Pins[30]) iprintf ("The pin is low");
void hiz()	Set output to tristate (high impedance input)	Pins[15].hiz();
void drive	Turn output on (opposite of tristate)	Pins[22].drive();
void function()	Set pin to special function or GPIO	Pins[5].function(PIN5_GPIO); Pins[5].function(PIN5_CANTX);

Program Examples

```
//////////////////////////////////////////////////////////////////////
// SIMPLE ALTERNATING HIGH/LOW OUTPUT PIN:                          //
//                                                                  //
// This program configures pin 38 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   //
// pin 38 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 performs the same function. //
//////////////////////////////////////////////////////////////////////

#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 <serialupdate.h>
#include <pins.h>

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

const char *AppName = "Mod5213PinIOTest";

void UserMain( void *pd )
{
   OSChangePrio( MAIN_PRIO );
   EnableSerialUpdate();

   SimpleUart( 0, SystemBaud );
   assign_stdio( 0 );

   Pins[38].function( PIN38_GPIO );   // Configure pin 38 for GPIO

   while ( 1 )
   {
      OSTimeDly( 1 * TICKS_PER_SECOND );
      Pins[38].set();                 // Set pin high
      OSTimeDly( 1 * TICKS_PER_SECOND );
      Pins[38].clr();                 // Set pin low
   }
}
```

```
////////////////////////////////////////////////////////////////////////
// SENDING SIGNALS FROM AN OUTPUT PIN TO AN INPUT PIN:                //
//                                                                    //
// This program configures pins 4 and 21 as GPIO output and GPIO      //
// input, respectively. In order for this program to properly work,   //
// a jumper wire is needed to connect header pins 4 and 21 on the     //
// carrier development board.                                         //
//                                                                    //
// In an infinite loop, alternating high and low signals are driven   //
// out on pin 4, where pin 21 will then be read. If the signal read   //
// from pin 21 is high, then the message "Hit!" will be outputted     //
// through the serial port to MTTTY. If the signal read from pin 21   //
// is low, then the message "Miss!" will be outputted. After each     //
// send/read, there is a 1-second delay.                              //
////////////////////////////////////////////////////////////////////////

#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 <serialupdate.h>
#include <pins.h>

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

const char *AppName = "Mod5213PinIOTest";

void UserMain( void *pd )
{
   OSChangePrio( MAIN_PRIO );
   EnableSerialUpdate();

   SimpleUart( 0, SystemBaud );
   assign_stdio( 0 );

   Pins[4].function( PIN4_GPIO );      // Configure pin 4 for GPIO
   Pins[21].function( PIN21_GPIO );    // Configure pin 21 for GPIO

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

      Pins[4] = 1;                     // Set 4 output high
      if ( Pins[21] )                  // Read 21 input pin state
         iprintf( "Hit!\r\n" );
      else
         iprintf( "Miss!\r\n" );

      OSTimeDly( 1 * TICKS_PER_SECOND );
```

```
      Pins[4] = 0;                    // Set 4 output low
      if ( Pins[21] )                 // Read 21 input pin state
         iprintf( "Hit!\r\n" );
      else
         iprintf( "Miss!\r\n" );

   }
}
```