Mod5272 PinIO Class

Application Note

Revision 1.0
December 14, 2005
Document Status: Initial Release

Table of Contents

Introduction

The PinIO Class provides an easy way to configure and operate the Freescale MCF5272 microprocessor GPIO signals. Each signal pin on the 5272 can have multiple functions. You can use the PinIO Class to control GPIO signals without having to explicitly configure the 5272 registers. Configuration of the processor registers are done in the member functions of the PinIO class. There are 27 pins on the Mod5272 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 sim5272.h and use the Freescale MCF5272 user 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 directory, and it is used by the Mod5270, Mod5272, and Mod5282. With this class, the pins associated with each module 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 27 pins available for GPIO on the Mod5272, as well as their primary and alternate functions, if any:

Connector	Pin	Definition	Function
J2	3	PINJ2_3_URXD0 PINJ2_3_GPIO	1: UART 0 Receive / DMA Timer In 2 0: GPIO
J2	4	PINJ2_4_UTXD0 PINJ2_4_GPIO	1: UART 0 Transmit 0: GPIO
J2	6	(Not Applicable)	0: GPIO (Enabled by Default)
J2	7	(Not Applicable)	0: GPIO (Enabled by Default)
J2	8	(Not Applicable)	0: GPIO (Enabled by Default)
J2	9	(Not Applicable)	0: GPIO (Enabled by Default)
J2	10	(Not Applicable)	0: GPIO (Enabled by Default)
J2	11	(Not Applicable)	0: GPIO (Enabled by Default)
J2	12	(Not Applicable)	0: GPIO (Enabled by Default)
J2	13	(Not Applicable)	0: GPIO (Enabled by Default)
J2	15	(Not Applicable)	0: GPIO (Enabled by Default)

J2	16	(Not Applicable)	0: GPIO (Enabled by Default)
J2	17	(Not Applicable)	0: GPIO (Enabled by Default)
J2	18	(Not Applicable)	0: GPIO (Enabled by Default)
J2	19	(Not Applicable)	0: GPIO (Enabled by Default)
J2	20	(Not Applicable)	0: GPIO (Enabled by Default)
J2	23	(Not Applicable)	0: GPIO (Enabled by Default)
J2	24	(Not Applicable)	0: GPIO (Enabled by Default)
J2	26	PINJ2_26_SPI_CS3 PINJ2_26_DOUT3 PINJ2_26_GPIO	1: SPI Chip Select 3 2: DMA Timer Output 3 0: GPIO
J2	29	PINJ2_29_URT0_CTS PINJ2_29_GPIO	1: UART 0 Clear to Send 0: GPIO
J2	33	PINJ2_33_USB_TXEN PINJ2_33_GPIO	1: USB Transmit Enable 0: GPIO
J2	36	PINJ2_36_TOUT0 PINJ2_36_GPIO	1: DMA Timer Output 0 0: GPIO
J2	38	PINJ2_38_URT0_RTS PINJ2_38_GPIO	1: UART 0 Request to Send 0: GPIO
J2	39	PINJ2_39_USB_TP PINJ2_39_GPIO	1: USB Transmit Serial Data Output 0: GPIO
J2	40	PINJ2_40_SPI_CS1 PINJ2_40_GPIO	1: SPI Chip Select 1 0: GPIO
J2	42	PINJ2_42_USB_RP PINJ2_42_GPIO	1: USB Receive Serial Data Input 0: GPIO
J2	48	PINJ2_48_DGNT1 PINJ2_48_GPIO	1: D-Channel Grant 1 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 J2-4 needs to be configured for GPIO it would be written as:

```
J2[4].function( PINJ2_4_GPIO );
```

Or, if UART 0 Transmit signal functionality is needed, then it would be written as:

```
J2[4].function( PINJ2_4_UTXD0 );
```

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

Pins J2-6 to J2-13, J2-15 to J2-20, J2-23, and J2-24 are dedicated GPIO pins. They do not have a pin assignment or control register to configure them for GPIO since they have no primary or alternate functions.

Pin Class Member Functions

Using the Pin Class member functions to configure and use the GPIO pins eliminate 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 J2-36 (DMA Timer Output 0) for GPIO and set it high without the PinIO class, then it would be written like this:

```
#include <../MOD5272/system/sim5272.h>

sim.pbcnt &= ~0x0000C000;  // Configure pin J2-36 for GPIO
sim.pbdat |= 0x0080;       // Set bit to be driven out on pin
sim.pbddr |= 0x0080;       // 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>

J2[36].function( PINJ2_36_GPIO ); // Configure pin J2-36 for GPIO
J2[36] = 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	J1[7].set(); J1[7] = 1;
void clr()	Set output low	J2[20].clr(); J2[20] = 0;
BOOL read()	Read pin high/low state	BOOL bpinstate = J2[29]; if (!J2[29]) iprintf ("The pin is low");
void hiz()	Set output to tristate (high impedance input)	J2[38].hiz();
void drive	Turn output on (opposite of tristate)	J2[7].drive();
void function()	Set pin to special function or GPIO	J2[40].function(PINJ2_40_SPI_CS1); J2[4].function(PINJ2_4_UTXD0);

Program Examples

```
//////////////////////////////////////////////////////////////////////
// SIMPLE ALTERNATING HIGH/LOW OUTPUT PIN:                          //
//                                                                  //
// This program configures pin J2-39 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   //
// J2-39 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 <startnet.h>
#include <autoupdate.h>
#include <dhcpclient.h>
#include <pins.h>

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

const char *AppName = "Mod5272PinsTest";

void UserMain( void *pd )
{
   InitializeStack();
   if ( EthernetIP == 0 ) GetDHCPAddress();
   OSChangePrio( MAIN_PRIO );
   EnableAutoUpdate();
   StartHTTP();

   J2[39].function( PINJ2_39_GPIO );   // Configure pin J2-39 for GPIO

   while ( 1 )
   {
      OSTimeDly( 1 * TICKS_PER_SECOND );
      J2[39].set();                        // Set pin high
      OSTimeDly( 1 * TICKS_PER_SECOND );
      J2[39].clr();                        // Set pin low
   }
}
```

```
////////////////////////////////////////////////////////////////////////
// SENDING SIGNALS FROM AN OUTPUT PIN TO AN INPUT PIN:                //
//                                                                    //
// This program configures pins J2-3 and J2-48 as GPIO output and     //
// GPIO input, respectively. In order for this program to properly    //
// work, a jumper wire is needed to connect the J2-3 and J2-48        //
// header pins on the carrier development board.                      //
//                                                                    //
// In an infinite loop, alternating high and low signals are driven   //
// out on J2-3, where J2-48 will then be read. If the signal read     //
// from J2-48 is high, the message "Hit!" will be outputted through   //
// the serial port to MTTTY. If the signal read from J2-48 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 <startnet.h>
#include <autoupdate.h>
#include <dhcpclient.h>
#include <pins.h>

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

const char *AppName = "Mod5272PinsTest";

void UserMain( void *pd )
{
   InitializeStack();
   if ( EthernetIP == 0 ) GetDHCPAddress();
   OSChangePrio( MAIN_PRIO );
   EnableAutoUpdate();
   StartHTTP();

   J2[3].function( PINJ2_3_GPIO );     // Configure pin J2-3 for GPIO
   J2[48].function( PINJ2_48_GPIO );   // Configure pin J2-48 for GPIO

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

      J2[3] = 1;                       // Set J2-3 output high
      if ( J2[48] )                    // Read J2-48 input pin state
         iprintf( "Hit!\r\n" );
      else
         iprintf( "Miss!\r\n" );

      OSTimeDly( 1 * TICKS_PER_SECOND );

      J2[3] = 0;                       // Set J2-3 output low
      if ( J2[48] )                    // Read J2-48 input pin state
```

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

      }
   }
```