Mod5270 PinIO Class

Application Note
# Table of Contents

- Introduction .................................................. 3
- PinIO Class .................................................... 3
  - Pin Class Constants ...................................... 3
  - Pin Class Member Functions ............................ 7
- Program Examples ............................................. 8
**Introduction**

The PinIO Class provides an easy way to configure and operate the Freescale MCF5270 microprocessor GPIO signals. Each signal pin on the 5270 can have multiple functions. You can use the PinIO Class to control GPIO signals without having to explicitly configure the 5270 registers. Configuration of the processor registers are done in the member functions of the PinIO class. There are 46 pins on the Mod5270 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 sim5270.h and use the Freescale MCF5270 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 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 46 pins available for GPIO on the Mod5270, as well as their primary and alternate functions, if any:

<table>
<thead>
<tr>
<th>Connector</th>
<th>Pin</th>
<th>Definition</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1</td>
<td>5</td>
<td>PINJ1_5_CS1</td>
<td>1: Chip Select 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PINJ1_5_GPIO</td>
<td>0: GPIO</td>
</tr>
<tr>
<td>J1</td>
<td>6</td>
<td>PINJ1_6_CS2</td>
<td>1: Chip Select 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PINJ1_6_GPIO</td>
<td>0: GPIO</td>
</tr>
<tr>
<td>J1</td>
<td>7</td>
<td>PINJ1_7_CS3</td>
<td>1: Chip Select 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PINJ1_7_GPIO</td>
<td>0: GPIO</td>
</tr>
<tr>
<td>J2</td>
<td>3</td>
<td>PINJ2_3_URXD0</td>
<td>1: UART 0 Receive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PINJ2_3_GPIO</td>
<td>0: GPIO</td>
</tr>
<tr>
<td>J2</td>
<td>4</td>
<td>PINJ2_4_UTXD0</td>
<td>1: UART 0 Transmit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PINJ2_4_GPIO</td>
<td>0: GPIO</td>
</tr>
<tr>
<td>J2</td>
<td>6</td>
<td>PINJ2_PD_PDATA</td>
<td>1: P Data High 14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PINJ2_PD_GPIO</td>
<td>0: GPIO</td>
</tr>
</tbody>
</table>
| J2  | 7   | PINJ2_PD_PDATA  | 1: P Data High 13  
|     |     | PINJ2_PD_GPIO   | 0: GPIO            |
| J2  | 8   | PINJ2_PD_PDATA  | 1: P Data High 15  
|     |     | PINJ2_PD_GPIO   | 0: GPIO            |
| J2  | 9   | PINJ2_PD_PDATA  | 1: P Data High 11  
|     |     | PINJ2_PD_GPIO   | 0: GPIO            |
| J2  | 10  | PINJ2_PD_PDATA  | 1: P Data High 12  
|     |     | PINJ2_PD_GPIO   | 0: GPIO            |
| J2  | 11  | PINJ2_PD_PDATA  | 1: P Data High 10  
|     |     | PINJ2_PD_GPIO   | 0: GPIO            |
| J2  | 12  | PINJ2_PD_PDATA  | 1: P Data High 9   
|     |     | PINJ2_PD_GPIO   | 0: GPIO            |
| J2  | 13  | PINJ2_PD_PDATA  | 1: P Data High 8   
|     |     | PINJ2_PD_GPIO   | 0: GPIO            |
| J2  | 15  | PINJ2_PD_PDATA  | 1: P Data Low 0    
|     |     | PINJ2_PD_GPIO   | 0: GPIO            |
| J2  | 16  | PINJ2_PD_PDATA  | 1: P Data Low 1    
|     |     | PINJ2_PD_GPIO   | 0: GPIO            |
| J2  | 17  | PINJ2_PD_PDATA  | 1: P Data Low 4    
|     |     | PINJ2_PD_GPIO   | 0: GPIO            |
| J2  | 18  | PINJ2_PD_PDATA  | 1: P Data Low 2    
|     |     | PINJ2_PD_GPIO   | 0: GPIO            |
| J2  | 19  | PINJ2_PD_PDATA  | 1: P Data Low 5    
|     |     | PINJ2_PD_GPIO   | 0: GPIO            |
| J2  | 20  | PINJ2_PD_PDATA  | 1: P Data Low 6    
|     |     | PINJ2_PD_GPIO   | 0: GPIO            |
| J2  | 21  | PINJ2_21_URXD1  | 1: UART 1 Receive  
|     |     | PINJ2_21_GPIO   | 0: GPIO            |
| J2  | 22  | PINJ2_22_UTXD1  | 1: UART 1 Transmit 
|     |     | PINJ2_22_GPIO   | 0: GPIO            |
| J2  | 23  | PINJ2_PD_PDATA  | 1: P Data Low 3    
|     |     | PINJ2_PD_GPIO   | 0: GPIO            |
| J2  | 24  | PINJ2_PD_PDATA  | 1: P Data Low 7    
|     |     | PINJ2_PD_GPIO   | 0: GPIO            |
| J2  | 25  | PINJ2_25_SPI_CLK| 1: SPI Clock       
|     |     | PINJ2_25_SCL    | 2: I2C Serial Clock|
|     |     | PINJ2_25_GPIO   | 0: GPIO            |
| J2  | 26  | PINJ2_26_DOUT3  | 1: DMA Timer Output 3  
|     |     | PINJ2_26_SPI_CS3| 2: SPI Chips Select 3|
|     |     | PINJ2_26_UART2_RTS| 3: UART 2 Request to Send |
|     |     | PINJ2_26_GPIO   | 0: GPIO            |
| J2  | 27  | PINJ2_27_SPI_DIN| 1: SPI Data In     
|     |     | PINJ2_27_SDA    | 2: I2C Serial Data  
|     |     | PINJ2_27_GPIO   | 0: GPIO            |
| J2  | 28  | PINJ2_28_SPI_DOUT| 1: SPI Data Out    
|     |     | PINJ2_28_GPIO   | 0: GPIO            |
| J2  | 29  | PINJ2_29_UART0_CTS| 1: UART 0 Clear to Send  
|     |     | PINJ2_29_GPIO   | 0: GPIO            |
| J2  | 30  | PINJ2_30_SPI_CS0| 1: SPI Chip Select 0 
|     |     | PINJ2_30_GPIO   | 0: GPIO            |
| J2  | 31  | PINJ2_31_TIN0   | 1: DMA Timer Input 0 
|     |     | PINJ2_31_DREQ0  | 2: DMA Request 0    
|     |     | PINJ2_31_GPIO   | 0: GPIO            |
| J2  | 32  | PINJ2_32_UART1_RTS| 1: UART 1 Request to Send  
|     |     | PINJ2_32_UART2_RTS| 2: UART 2 Request to Send  
|     |     | PINJ2_32_GPIO   | 0: GPIO            |
| J2  | 33 | PINJ2_33_UCTS1 | 1: UART 1 Clear to Send  
|     |    | PINJ2_33_UCTS2 | 2: UART 2 Clear to Send  
|     |    | PINJ2_33_GPIO  | 0: GPIO  
| J2  | 34 | PINJ2_34_DTOUT1 | 1: DMA Timer Output 1  
|     |    | PINJ2_34_DACK1 | 2: DMA Transfer Ack 1  
|     |    | PINJ2_34_GPIO  | 0: GPIO  
| J2  | 35 | PINJ2_35_TIN2 | 1: DMA Timer Input 2  
|     |    | PINJ2_35_DTOUT2 | 2: DMA Timer Output 2  
|     |    | PINJ2_35_DREQ2 | 3: DMA Request 2  
|     |    | PINJ2_35_GPIO  | 0: GPIO  
| J2  | 36 | PINJ2_36_DTOUT0 | 1: DMA Timer Output 0  
|     |    | PINJ2_36_DACK0 | 2: DMA Transfer Ack 0  
|     |    | PINJ2_36_GPIO  | 0: GPIO  
| J2  | 37 | PINJ2_37_TIN1 | 1: DMA Timer Input 1  
|     |    | PINJ2_37_DTOUT1 | 2: DMA Timer Output 1  
|     |    | PINJ2_37_DREQ1 | 3: DMA Request 1  
|     |    | PINJ2_37_GPIO  | 0: GPIO  
| J2  | 38 | PINJ2_38_UART0_RTS | 1: UART 0 Request to Send  
|     |    | PINJ2_38_GPIO  | 0: GPIO  
| J2  | 39 | PINJ2_39_SDA | 1: I2C Serial Data  
|     |    | PINJ2_39_GPIO  | 0: GPIO  
| J2  | 40 | PINJ2_40_SPI_CS1 | 1: SPI Chip Select 1  
|     |    | PINJ2_40_SDRAMC | 2: SDRAMC SCKE  
|     |    | PINJ2_40_GPIO  | 0: GPIO  
| J2  | 41 | PINJ2_41_UART2_RX | 1: UART 2 Receive  
|     |    | PINJ2_41_GPIO  | 0: GPIO  
| J2  | 42 | PINJ2_42_SCL | 1: I2C Serial Clock  
|     |    | PINJ2_42_GPIO  | 0: GPIO  
| J2  | 43 | PINJ2_43_IRQ1_LVS | 1: Level-Sensitive  
|     |    | PINJ2_43_IRQ1_RET | 2: Rising-Edge Triggered  
|     |    | PINJ2_43_IRQ1_FET | 3: Falling-Edge Triggered  
|     |    | PINJ2_43_IRQ1_FRT | 4: Fall and Rise Edge Triggered  
| J2  | 44 | PINJ2_44_UART2_TX | 1: UART 2 Transmit  
|     |    | PINJ2_44_GPIO  | 0: GPIO  
| J2  | 45 | PINJ2_45_IRQ3_LVS | 1: Level-Sensitive  
|     |    | PINJ2_45_IRQ3_RET | 2: Rising-Edge Triggered  
|     |    | PINJ2_45_IRQ3_FET | 3: Falling-Edge Triggered  
|     |    | PINJ2_45_IRQ3_FRT | 4: Fall and Rise Edge Triggered  
| J2  | 47 | PINJ2_47_IRQ5_LVS | 1: Level-Sensitive  
|     |    | PINJ2_47_IRQ5_RET | 2: Rising-Edge Triggered  
|     |    | PINJ2_47_IRQ5_FET | 3: Falling-Edge Triggered  
|     |    | PINJ2_47_IRQ5_FRT | 4: Fall and Rise Edge Triggered  
| J2  | 48 | PINJ2_48_IRQ7_LVS | 1: Level-Sensitive  
|     |    | PINJ2_48_IRQ7_RET | 2: Rising-Edge Triggered  
|     |    | PINJ2_48_IRQ7_FET | 3: Falling-Edge Triggered  
|     |    | PINJ2_48_IRQ7_FRT | 4: Fall and Rise Edge Triggered  

**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 J2-30 needs to be configured for GPIO it would be written as:

```cpp
J2[30].function( PINJ2_30_GPIO );
```

Or, if I²C serial clock signal functionality is needed, then it would be written as:

```cpp
J2[42].function( PINJ2_42_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

The following rules apply to the assignment of a pin as GPIO:

1. The P Data Low[0:7] and P Data High[8:15] pins (J2-6 to J2-13, J2-15 to J2-20, J2-23, and J2-24) must be assigned as a group. In other words, if any one pin is assigned as GPIO, they must all be used as GPIO. For example, the following line of code (where 'x' is any P Data pin number) will configure ALL P Data pins as GPIO:

```cpp
J[x].function( PINJ2_PD_GPIO );
```

2. The IRQ pins (J2-43, J2-45, J2-47, and J2-48) are GPIO by default; they do not have a GPIO configuration register. They only require configuration if you want to use them as interrupt inputs instead of GPIO. Once configured as an interrupt input, they cannot be reconfigured as GPIO.

Note on Chip Select[1:3] pins (J1-5 to J1-7): It is not recommended that they be configured for GPIO when using the Mod5270 with the MOD-DEV-100 development carrier board. The three chip select signals are ANDed together with the TIP (Transfer in Progress) signal, which in turn is connected to the external buffer on the carrier board. Doing so may enable the external buffer and cause a crash on the bus, thus resulting in trap errors. It is preferred that a carrier board without an external buffer be used to test the chip select GPIO pins, such as the MOD-DEV-50.
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 J2-44 (UART 2 – Transmit) for GPIO and set it high without the PinIO class, then it would be written like this:

```c
#include <../MOD5270/system/sim5270.h>

sim.gpio.par_uart &= ~0x1000;  // Configure pin J2-44 for GPIO
sim.gpio.pddsdr_uarth = 0x02;  // Set bit to be driven out on pin
sim.gpio.pddr_uarth |= 0x02;   // Set signal direction as output
```

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

```c
#include <pins.h>

J2[44].function( PINJ2_44_GPIO );  // Configure pin J2-44 for GPIO
J2[44] = 1;                     // Set pin as output high
```

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

<table>
<thead>
<tr>
<th>Member Function Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void set()</code></td>
<td>Set output high</td>
<td>J1[7].set(); J1[7] = 1;</td>
</tr>
<tr>
<td><code>void clr()</code></td>
<td>Set output low</td>
<td>J2[20].clr(); J2[20] = 0;</td>
</tr>
</tbody>
</table>
| `BOOL read()`        | Read pin high/low state      | BOOL bpinstate = J2[30];,
|                      |                              | if ( !J2[30] )
|                      |                              |     printf ( “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[44].function( PINJ2_44_GPIO ); J2[42].function( PINJ2_42_SCL ); |
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 = "Mod5270PinsTest";

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
    }
}
This program configures pins J1-5 and J2-44 as GPIO output and GPIO input, respectively. In order for this program to properly work, a jumper wire is needed to connect the J1-5 and J2-44 header pins on the carrier development board (if you are using a carrier board with an external buffer (i.e., MOD-DEV-100), then you can change all references to J1-5 to any available GPIO pin on the J2 connector).

In an infinite loop, alternating high and low signals are driven out on J1-5, where J2-44 will then be read. If the signal read from J2-44 is high, the message “Hit!” will be outputted through the serial port to MTTY. If the signal read from J2-44 is low, then the message “Miss!” will be outputted. After each send/read, there is a 1-second delay.

```c
#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 = "Mod5270PinsTest";

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

    J1[5].function( PINJ1_5_GPIO );  // Configure pin J1-5 for GPIO
    J2[44].function( PINJ2_44_GPIO ); // Configure pin J2-44 for GPIO

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

        J1[5] = 1;  // Set J1-5 output high
        if ( J2[44] )  // Read J2-44 input pin state
            printf( "Hit!\r\n" );
        else
            printf( "Miss!\r\n" );

        OSTimeDly( 1 * TICKS_PER_SECOND );
    }
}
```
J1[5] = 0;  // Set J1-5 output low
if ( J2[44] )  // Read J2-44 input pin state
    iprintf("Hit!\r\n");
else
    iprintf("Miss!\r\n");
}