Mod5234 PinIO Class

Application Note
# Table of Contents

Introduction 3

Electrical Specifications 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 MCF5234 microprocessor GPIO signals. Each signal pin on the 5234 can have multiple functions. You can use the PinIO Class to control GPIO signals without having to explicitly configure the 5243 registers. Configuration of the processor registers are done in the member functions of the PinIO class. There are 49 pins on the Mod5234 that are made available for GPIO (16 of those pins are ETU channel pins, which are not applicable to this application note). 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 sim5234.h and use the Freescale MCF5235 reference manual 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 is 5 mA. Please see the document, "MCF523x Integrated Microprocessor Hardware Specification" for more information.

PinIO Class

This class is defined in the header file “pins.h” located in the \nburn\include directory, and it is used by the Mod5234, 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 33 pins available for GPIO on the Mod5234, as well as their primary and alternate functions, if any (the ETU channel pins are not included, since they are not relevant to PinIO Class usage):
<table>
<thead>
<tr>
<th>Connector</th>
<th>Pin</th>
<th>Definition</th>
<th>Function</th>
</tr>
</thead>
</table>
| J1        | 5   | PINJ1_5_CS1 | 1: Chip Select 1  
PINJ1_5_GPIO | 0: GPIO |
| J1        | 6   | PINJ1_6_CS2 | 1: Chip Select 2  
PINJ1_6_GPIO | 0: GPIO |
| J1        | 7   | PINJ1_7_CS3 | 1: Chip Select 3  
PINJ1_7_GPIO | 0: GPIO |
| J1        | 13  | PINJ1_13_TA | 1: Transfer Acknowledge  
PINJ1_13_GPIO | 0: GPIO |
| J2        | 3   | PINJ2_3_U0RXD | 1: UART 0 Receive  
PINJ2_3_GPIO | 0: GPIO |
| J2        | 4   | PINJ2_4_U0TXD | 1: UART 0 Transmit  
PINJ2_4_GPIO | 0: GPIO |
| J2        | 21  | PINJ2_21_U1RXD | 1: UART 1 Receive  
PINJ2_21_GPIO | 2: CAN 0 Receive  
PINJ2_21_CAN0RX | 0: GPIO |
| J2        | 22  | PINJ2_22_U1TXD | 1: UART 1 Transmit  
PINJ2_22_GPIO | 2: CAN 0 Transmit  
PINJ2_22_CAN0TX | 0: GPIO |
| J2        | 23  | PINJ2_23_U1RTS | 1: UART 1 Request to Send  
PINJ2_23_GPIO | 2: UART 2 Request to Send  
PINJ2_23_U2RTS | 0: GPIO |
| J2        | 24  | PINJ2_24_U1CTS | 1: UART 1 Clear to Send  
PINJ2_24_GPIO | 2: UART 2 Clear to Send  
PINJ2_24_U2CTS | 0: GPIO |
| J2        | 25  | PINJ2_25_SPI_CLK | 1: SPI Clock  
PINJ2_25_GPIO | 2: I2C Serial Clock  
PINJ2_25_SCL | 0: GPIO |
| J2        | 26  | PINJ2_26_TCRCLK | 1: TPU Time Base Clock  
PINJ2_26_GPIO | 0: GPIO |
| J2        | 27  | PINJ2_27_SPI_DIN | 1: SPI Data In  
PINJ2_27_GPIO | 2: I2C Serial Data  
PINJ2_27_SDA | 0: GPIO |
| J2        | 28  | PINJ2_28_SPI_DOUT | 1: SPI Data Out  
PINJ2_28_GPIO | 0: GPIO |
| J2        | 29  | PINJ2_29_U0CTS | 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_DT0IN | 1: DMA Timer Input 0  
PINJ2_31_GPIO | 2: DMA Request 0  
PINJ2_31_DREQ0 | 0: GPIO |
| J2        | 32  | PINJ2_32_UTPUUDIS | 1: Upper TPU Channel Output Disable  
PINJ2_32_GPIO | 0: GPIO |
| J2        | 33  | PINJ2_33_DT2OUT | 1: DMA Timer Output 2  
PINJ2_33_GPIO | 2: DMA Transfer Acknowledge 2  
PINJ2_33_DACK2 | 0: GPIO |
| J2        | 34  | PINJ2_34_DTIOUT | 1: DMA Timer Output 1  
PINJ2_34_GPIO | 2: DMA Transfer Acknowledge 1  
PINJ2_34_DACK1 | 0: GPIO |
| J2        | 35  | PINJ2_35_LTPUUDIS | 1: Lower TPU Channel Output Disable  
PINJ2_35_GPIO | 0: GPIO |
| J2        | 36  | PINJ2_36_DTIOUT | 1: DMA Timer Output 0  
PINJ2_36_GPIO | 2: DMA Transfer Acknowledge 0  
PINJ2_36_DACK0 | 0: GPIO |
<table>
<thead>
<tr>
<th>Pin</th>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>J2</td>
<td>PINJ2_36_GPIO</td>
<td>0: GPIO</td>
</tr>
<tr>
<td></td>
<td>PINJ2_37_DTIN</td>
<td>1: DMA Timer Input 1</td>
</tr>
<tr>
<td></td>
<td>PINJ2_37_DREQ1</td>
<td>2: DMA Request 1</td>
</tr>
<tr>
<td></td>
<td>PINJ2_37_DTOUT</td>
<td>3: DMA Timer Output 1</td>
</tr>
<tr>
<td></td>
<td>PINJ2_37_GPIO</td>
<td>0: GPIO</td>
</tr>
<tr>
<td>J2</td>
<td>PINJ2_38_UORTS</td>
<td>1: UART 0 Request to Send</td>
</tr>
<tr>
<td></td>
<td>PINJ2_38_GPIO</td>
<td>0: GPIO</td>
</tr>
<tr>
<td>J2</td>
<td>PINJ2_39_SDA</td>
<td>1: I2C Serial Data</td>
</tr>
<tr>
<td></td>
<td>PINJ2_39_CAN0TX</td>
<td>2: CAN 0 Transmit</td>
</tr>
<tr>
<td></td>
<td>PINJ2_39_GPIO</td>
<td>0: GPIO</td>
</tr>
<tr>
<td>J2</td>
<td>PINJ2_40_SPI_CS1</td>
<td>1: SPI Chip Select 1</td>
</tr>
<tr>
<td></td>
<td>PINJ2_40_SCKE</td>
<td>2: SDRAMC SCKE</td>
</tr>
<tr>
<td></td>
<td>PINJ2_40_GPIO</td>
<td>0: GPIO</td>
</tr>
<tr>
<td>J2</td>
<td>PINJ2_41_U2RXD</td>
<td>1: UART 2 Receive</td>
</tr>
<tr>
<td></td>
<td>PINJ2_41_GPIO</td>
<td>0: GPIO</td>
</tr>
<tr>
<td>J2</td>
<td>PINJ2_42_SCL</td>
<td>1: I2C Serial Clock</td>
</tr>
<tr>
<td></td>
<td>PINJ2_42_CAN0RX</td>
<td>2: CAN 0 Receive</td>
</tr>
<tr>
<td></td>
<td>PINJ2_42_GPIO</td>
<td>0: GPIO</td>
</tr>
<tr>
<td>J2</td>
<td>PINJ2_43_IRQ1_LVS</td>
<td>1: Level-Sensitive</td>
</tr>
<tr>
<td></td>
<td>PINJ2_43_IRQ1_RET</td>
<td>2: Rising-Edge Triggered</td>
</tr>
<tr>
<td></td>
<td>PINJ2_43_IRQ1_FET</td>
<td>3: Falling-Edge Triggered</td>
</tr>
<tr>
<td></td>
<td>PINJ2_43_IRQ1_FRT</td>
<td>4: Fall and Rise Edge Triggered</td>
</tr>
<tr>
<td>J2</td>
<td>PINJ2_44_U2TXD</td>
<td>1: UART 2 Transmit</td>
</tr>
<tr>
<td></td>
<td>PINJ2_44_GPIO</td>
<td>0: GPIO</td>
</tr>
<tr>
<td>J2</td>
<td>PINJ2_45_IRQ3_LVS</td>
<td>1: Level-Sensitive</td>
</tr>
<tr>
<td></td>
<td>PINJ2_45_IRQ3_RET</td>
<td>2: Rising-Edge Triggered</td>
</tr>
<tr>
<td></td>
<td>PINJ2_45_IRQ3_FET</td>
<td>3: Falling-Edge Triggered</td>
</tr>
<tr>
<td></td>
<td>PINJ2_45_IRQ3_FRT</td>
<td>4: Fall and Rise Edge Triggered</td>
</tr>
<tr>
<td>J2</td>
<td>PINJ2_47_IRQ5_LVS</td>
<td>1: Level-Sensitive</td>
</tr>
<tr>
<td></td>
<td>PINJ2_47_IRQ5_RET</td>
<td>2: Rising-Edge Triggered</td>
</tr>
<tr>
<td></td>
<td>PINJ2_47_IRQ5_FET</td>
<td>3: Falling-Edge Triggered</td>
</tr>
<tr>
<td></td>
<td>PINJ2_47_IRQ5_FRT</td>
<td>4: Fall and Rise Edge Triggered</td>
</tr>
<tr>
<td>J2</td>
<td>PINJ2_48_IRQ7_LVS</td>
<td>1: Level-Sensitive</td>
</tr>
<tr>
<td></td>
<td>PINJ2_48_IRQ7_RET</td>
<td>2: Rising-Edge Triggered</td>
</tr>
<tr>
<td></td>
<td>PINJ2_48_IRQ7_FET</td>
<td>3: Falling-Edge Triggered</td>
</tr>
<tr>
<td></td>
<td>PINJ2_48_IRQ7_FRT</td>
<td>4: Fall and Rise Edge Triggered</td>
</tr>
</tbody>
</table>

**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:

\[ J2[30].function( \text{PINJ2}_30\_GPIO ) ; \]

Or, if \( ^2 \text{C} \) serial clock signal functionality is needed, then it would be written as:

\[ J2[42].function( \text{PINJ2}_42\_SCL ) ; \]

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 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 Mod5234 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 <..\MOD5234\system\sim5234.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>
</table>
| void set()           | Set output high | J1[7].set();
|                      |              | J1[7] = 1;       |
| void clr()           | Set output low | J2[21].clr();
|                      |              | J2[21] = 0;       |
| BOOL read()          | Read pin high/low state | BOOL bpinnstate = 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[27].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
  }
}
#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 = "Mod5234PinsTest";

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

}