Mod5234 PinIO Class

Application Note

Revision 1.2
September 25, 2006
Document Status: Initial Release

Table of Contents

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 5234 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 ETPU 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 ETPU channel pins are not included, since they are not relevant to PinIO Class usage):

Connector	Pin	Definition	Function
J1	5	PINJ1_5_CS1 PINJ1_5_GPIO	1: Chip Select 1 0: GPIO
J1	6	PINJ1_6_CS2 PINJ1_6_GPIO	1: Chip Select 2 0: GPIO
J1	7	PINJ1_7_CS3 PINJ1_7_GPIO	1: Chip Select 3 0: GPIO
J1	13	PINJ1_13_TA PINJ1_13_GPIO	1: Transfer Acknowledge 0: GPIO
J2	3	PINJ2_3_U0RXD PINJ2_3_GPIO	1: UART 0 Receive 0: GPIO
J2	4	PINJ2_4_U0TXD PINJ2_4_GPIO	1: UART 0 Transmit 0: GPIO
J2	21	PINJ2_21_U1RXD PINJ2_21_CAN0RX PINJ2_21_GPIO	1: UART 1 Receive 2: CAN 0 Receive 0: GPIO
J2	22	PINJ2_22_U1TXD PINJ2_22_CAN0TX PINJ2_22_GPIO	1: UART 1 Transmit 2: CAN 0 Transmit 0: GPIO
J2	23	PINJ2_23_U1RTS PINJ2_23_U2RTS PINJ2_23_GPIO	1: UART 1 Request to Send 2: UART 2 Request to Send 0: GPIO
J2	24	PINJ2_24_U1CTS PINJ2_24_U2CTS PINJ2_24_GPIO	1: UART 1 Clear to Send 2: UART 2 Clear to Send 0: GPIO
J2	25	PINJ2_25_SPI_CLK PINJ2_25_SCL PINJ2_25_GPIO	1: SPI Clock 2: I2C Serial Clock 0: GPIO
J2	26	PINJ2_26_TCRCLK PINJ2_26_GPIO	1: TPU Time Base Clock 0: GPIO
J2	27	PINJ2_27_SPI_DIN PINJ2_27_SDA PINJ2_27_GPIO	1: SPI Data In 2: I2C Serial Data 0: GPIO
J2	28	PINJ2_28_SPI_DOUT PINJ2_28_GPIO	1: SPI Data Out 0: GPIO
J2	29	PINJ2_29_U0CTS PINJ2_29_GPIO	1: UART 0 Clear to Send 0: GPIO
J2	30	PINJ2_30_SPI_CS0 PINJ2_30_GPIO	1: SPI Chip Select 0 0: GPIO
J2	31	PINJ2_31_DT0IN PINJ2_31_DREQ0 PINJ2_31_GPIO	1: DMA Timer Input 0 2: DMA Request 0 0: GPIO
J2	32	PINJ2_32_UTPUODIS PINJ2_32_GPIO	1: Upper TPU Channel Output Disable 0: GPIO
J2	33	PINJ2_33_DT2OUT PINJ2_33_DACK2 PINJ2_33_GPIO	1: DMA Timer Output 2 2: DMA Transfer Acknowledge 2 0: GPIO
J2	34	PINJ2_34_DT1OUT PINJ2_34_DACK1 PINJ2_34_GPIO	1: DMA Timer Output 1 2: DMA Transfer Acknowledge 1 0: GPIO
J2	35	PINJ2_35_LTPUODIS PINJ2_35_GPIO	1: Lower TPU Channel Output Disable 0: GPIO
J2	36	PINJ2_36_DT0OUT PINJ2_36_DACK0	1: DMA Timer Output 0 2: DMA Transfer Acknowledge 0

		PINJ2_36_GPIO	0: GPIO
J2	37	PINJ2_37_DT1IN PINJ2_37_DREQ1 PINJ2_37_DT1OUT PINJ2_37_GPIO	1: DMA Timer Input 1 2: DMA Request 1 3: DMA Timer Output 1 0: GPIO
J2	38	PINJ2_38_U0RTS PINJ2_38_GPIO	1: UART 0 Request to Send 0: GPIO
J2	39	PINJ2_39_SDA PINJ2_39_CAN0TX PINJ2_39_GPIO	1: I2C Serial Data 2: CAN 0 Transmit 0: GPIO
J2	40	PINJ2_40_SPI_CS1 PINJ2_40_SCKE PINJ2_40_GPIO	1: SPI Chip Select 1 2: SDRAMC SCKE 0: GPIO
J2	41	PINJ2_41_U2RXD PINJ2_41_GPIO	1: UART 2 Receive 0: GPIO
J2	42	PINJ2_42_SCL PINJ2_42_CAN0RX PINJ2_42_GPIO	1: I2C Serial Clock 2: CAN 0 Receive 0: GPIO
J2	43	PINJ2_43_IRQ1_LVS PINJ2_43_IRQ1_RET PINJ2_43_IRQ1_FET PINJ2_43_IRQ1_FRT	1: Level-Sensitive 2: Rising-Edge Triggered 3: Falling-Edge Triggered 4: Fall and Rise Edge Triggered
J2	44	PINJ2_44_U2TXD PINJ2_44_GPIO	1: UART 2 Transmit 0: GPIO
J2	45	PINJ2_45_IRQ3_LVS PINJ2_45_IRQ3_RET PINJ2_45_IRQ3_FET PINJ2_45_IRQ3_FRT	1: Level-Sensitive 2: Rising-Edge Triggered 3: Falling-Edge Triggered 4: Fall and Rise Edge Triggered
J2	47	PINJ2_47_IRQ5_LVS PINJ2_47_IRQ5_RET PINJ2_47_IRQ5_FET PINJ2_47_IRQ5_FRT	1: Level-Sensitive 2: Rising-Edge Triggered 3: Falling-Edge Triggered 4: Fall and Rise Edge Triggered
J2	48	PINJ2_48_IRQ7_LVS PINJ2_48_IRQ7_RET PINJ2_48_IRQ7_FET PINJ2_48_IRQ7_FRT	1: Level-Sensitive 2: Rising-Edge Triggered 3: Falling-Edge Triggered 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:

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

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

```
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 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:

```
#include <..\MOD5234\system\sim5234.h>

sim.gpio.par_uart &= ~0x1000;  // Configure pin J2-44 for GPIO
sim.gpio.ppdsdr_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:

```
#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:

Member Function Name	Description	Example

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 bpinstate = J2[30];
		if (!J2[30])
		   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[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
   }
}
```

```
///////////////////////////////////////////////////////////////////////
// SENDING SIGNALS FROM AN OUTPUT PIN TO AN INPUT PIN:               //
//                                                                   //
// 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 MTTTY. 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.                                        //
///////////////////////////////////////////////////////////////////////

#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" );

   }
}
```