NANO54415 PinIO Class

Application Note

Revision 1.0 | 2014-05-05 | Initial Release

Table of Contents

Introduction

The PinIO class provides an easy way to configure and operate the Freescale MCF54415 microprocessor GPIO signals. Each signal pin on the MCF54415 can have multiple functions. You can use the PinIO class to control GPIO signals without having to explicitly configure the MCF54415 registers. Configuration of the processor registers are done in the member functions of the PinIO class. There are 30 pins on the NANO54415 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, and assuming you have the NNDK tools installed, we recommend you use the register structure defined in:

```
\nburn\NANO54415\include\sim5441x.h
```

and use the “MCF5441x Reference Manual” located in:

```
\nburn\docs\FreescaleManuals\MCF54415RM.pdf
```

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 "MCF5441x ColdFire Microprocessor Data Sheet" located in:

```
\nburn\docs\FreescaleManuals\MCF54415DataSheet.pdf
```

for more information.

PinIO Class

This class is defined in the header file `\nburn\include\pins.h`. With this class, the pins can be configured for their primary function, alternate function(s), or GPIO. 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. The supported functions defined for each pin and the member functions to use those pins (when configured for GPIO) are respectively listed in the following files:

```
\nburn\NANO54415\include\pinconstant.h
\nburn\NANO54415\system\pins.cpp
```

PinIO Class Constants

The table below lists the 30 pins available for GPIO on the NANO54415, as well as their primary and alternate functions, if any.

Pin	Definition	Function
9	PIN_9_IRQ7	1: External Interrupt 7
	PIN_9_GPIO	0: Port C-6 GPIO
10	PIN_10_UART2_CTS	3: UART 2 – Clear to Send
	PIN_10_UART6_TXD	2: UART 6 – Transmit
	PIN_10_SSI1_BCLK	1: SSI 1 – Serial Bit Clock
	PIN_10_GPIO	0: Port E-6 GPIO / Rapid GPIO 14
12	PIN_12_OW_DAT	3: 1-Wire Data Signal
	PIN_12_DACK0	2: DMA Acknowledge 0
	PIN_12_GPIO	0: Port D-3 GPIO / Rapid GPIO 0
13	PIN_13_UART2_RXD	3: UART 2 – Receive
	PIN_13_PWM_A3	2: PWM A3 – Output Signal/Input Capture
	PIN_13_SSI1_RXD	1: SSI 1 – Serial Receive Data
	PIN_13_GPIO	0: Port E-4 GPIO
14	PIN_14_UART2_RTS	3: UART 2 – Request to Send
	PIN_14_UART6_RXD	2: UART 6 – Receive
	PIN_14_SSI1_FS	1: SSI 1 – Serial Frame Sync
	PIN_14_GPIO	0: Port E-5 GPIO / Rapid GPIO 15
15	PIN_15_SDHC_DAT3	3: SDHC DAT3 Line / Card Detection
	PIN_15_PWM_A1	2: PWM A1 – Output Signal/Input Capture
	PIN_15_DSPI1_PCS0	1: DSPI 1 – Peripheral Chip Select 0
	PIN_15_GPIO	0: Port F-2 GPIO
16	PIN_16_UART2_TXD	3: UART 2 – Transmit
	PIN_16_PWM_B3	2: PWM B3 – Output Signal/Input Capture
	PIN_16_SSI1_TXD	1: SSI 1 – Serial Transmit Data
	PIN_16_GPIO	0: Port E-3 GPIO
19	PIN_19_T0IN	3: Timer Input 0
	PIN_19_T0OUT	2: Timer Output 0
	PIN_19_USBO_VBUS_OC	1: USB On-the-Go VBUS Over-Current

	PIN_19_GPIO	0: Port E-7 GPIO / Rapid GPIO 4
20	PIN_20_CAN1_RX	3: CAN 1 – Receive
	PIN_20_UART9_RXD	2: UART 9 – Receive
	PIN_20_I2C1_SDA	1: I2C 1 – Serial Data
	PIN_20_GPIO	0: Port C-7 GPIO
21	PIN_21_T1IN	3: Timer Input 1
	PIN_21_T1OUT	2: Timer Output 1
	PIN_21_SDHC_DAT1	1: SDHC DAT1 Line / Interrupt Detect
	PIN_21_GPIO	0: Port D-0 GPIO / Rapid GPIO 3
22	PIN_22_CAN1_TX	3: CAN 1 – Transmit
	PIN_22_UART9_TXD	2: UART 9 – Transmit
	PIN_22_I2C1_SCL	1: I2C 1 – Serial Clock
	PIN_22_GPIO	0: Port B-0 GPIO
23	PIN_23_T2IN	3: Timer Input 2
	PIN_23_T2OUT	2: Timer Output 2
	PIN_23_SDHC_DAT2	1: SDHC DAT2 Line / Read Wait
	PIN_23_GPIO	0: Port D-1 GPIO / Rapid GPIO 2
24	PIN_24_UART0_RXD	3: UART 0 – Receive
	PIN_24_I2C4_SDA	2: I2C 4 – Serial Data
	PIN_24_DSPI2_SIN	1: DSPI 2 – Serial Data In
	PIN_24_GPIO	0: Port F-4 GPIO
25	PIN_25_T3IN	3: Timer Input 3
	PIN_25_T3OUT	2: Timer Output 3
	PIN_25_USBO_VBUS_EN	1: USB On-the-Go VBUS Enable
	PIN_25_GPIO	0: Port D-2 GPIO / Rapid GPIO 1
26	PIN_26_UART0_TXD	3: UART 0 – Transmit
	PIN_26_I2C4_SCL	2: I2C 4 – Serial Clock
	PIN_26_DSPI2_SOUT	1: DSPI 2 – Serial Data Out
	PIN_26_GPIO	0: Port F-3 GPIO
27	PIN_27_I2C0_SCL	3: I2C 0 – Serial Clock
	PIN_27_UART8_TXD	2: UART 8 – Transmit
	PIN_27_CAN0_TX	1: CAN 0 – Transmit
	PIN_27_GPIO	0: Port B-2 GPIO
28	PIN_28_UART0_RTS	3: UART 0 – Request to Send
	PIN_28_UART4_RXD	2: UART 4 – Receive
	PIN_28_DSPI2_PCS0	1: DSPI 2 – Peripheral Chip Select 0
	PIN_28_GPIO	0: Port F-5 GPIO / Rapid GPIO 6
29	PIN_29_I2C0_SDA	3: I2C 0 – Serial Data
	PIN_29_UART8_RXD	2: UART 8 – Receive
	PIN_29_CAN0_RX	1: CAN 0 – Receive
	PIN_29_GPIO	0: Port B-1 GPIO
30	PIN_30_UART0_CTS	3: UART 0 – Clear to Send
	PIN_30_UART4_TXD	2: UART 4 – Transmit
	PIN_30_DSPI2_SCK	1: DSPI 2 – Serial Clock
	PIN_30_GPIO	0: Port F-6 GPIO / Rapid GPIO 5
31	PIN_31_SDHC_CLK	3: SDHC Clock
	PIN_31_PWM_A0	2: PWM A0 – Output Signal/Input Capture
	PIN_31_DSPI1_SCK	1: DSPI 1 – Serial Clock

	PIN_31_GPIO	0: Port G-5 GPIO
32	PIN_32_UART1_RXD	3: UART 1 – Receive
	PIN_32_I2C5_SDA	2: I2C 5 – Serial Data
	PIN_32_DSPI3_SIN	1: DSPI 3 – Serial Data In
	PIN_32_GPIO	0: Port E-0 GPIO
33	PIN_33_SDHC_CMD	3: SDHC Command Line
	PIN_33_PWM_B0	2: PWM B0 – Output Signal/Input Capture
	PIN_33_DSPI1_SIN	1: DSPI 1 – Serial Data In
	PIN_33_GPIO	0: Port G-6 GPIO
34	PIN_34_UART1_TXD	3: UART 1 – Transmit
	PIN_34_I2C5_SCL	2: I2C 5 – Serial Clock
	PIN_34_DSPI3_SOUT	1: DSPI 3 – Serial Data Out
	PIN_34_GPIO	0: Port F-7 GPIO
35	PIN_35_SDHC_DAT0	3: SDHC DAT0 Line / Busy-State Detect
	PIN_35_PWM_B2	2: PWM B2 – Output Signal/Input Capture
	PIN_35_DSPI1_SOUT	1: DSPI 1 – Serial Data Out
	PIN_35_GPIO	0: Port G-7 GPIO
36	PIN_36_UART1_RTS	3: UART 1 – Request to Send
	PIN_36_UART5_RXD	2: UART 5 – Receive
	PIN_36_DSPI3_PCS0	1: DSPI 3 – Peripheral Chip Select 0
	PIN_36_GPIO	0: Port E-1 GPIO / Rapid GPIO 8
37	PIN_37_SDHC_DAT1	3: SDHC DAT1 Line / Interrupt Detect
	PIN_37_PWM_A2	2: PWM A2 – Output Signal/Input Capture
	PIN_37_DSPI1_PCS1	1: DSPI 1 – Peripheral Chip Select 1
	PIN_37_GPIO	0: Port F-0 GPIO
38	PIN_38_UART1_CTS	3: UART 1 – Clear to Send
	PIN_38_UART5_TXD	2: UART 5 – Transmit
	PIN_38_DSPI3_SCK	1: DSPI 3 – Serial Clock
	PIN_38_GPIO	0: Port E-2 GPIO / Rapid GPIO 7
39	PIN_39_SDHC_DAT2	3: SDHC DAT2 Line / Read Wait
	PIN_39_PWM_B1	2: PWM B1 – Output Signal/Input Capture
	PIN_39_DSPI1_PCS2	1: DSPI 1 – Peripheral Chip Select 2
	PIN_39_GPIO	0: Port F-1 GPIO
49	PIN_49_IRQ3	3: External Interrupt 3
	PIN_49_DSPI0_PCS3	2: DSPI 0 – Peripheral Chip Select 3
	PIN_49_USBH_VBUS_EN	1: USB Host VBUS Enable
	PIN_49_GPIO	0: Port C-3 GPIO
50	PIN_50_IRQ2	3: External Interrupt 2
	PIN_50_DSPI0_PCS2	2: DSPI 0 – Peripheral Chip Select 2
	PIN_50_USBH_VBUS_OC	1: USB Host VBUS Over-Current
	PIN_50_GPIO	0: Port C-2 GPIO

Pin Constants Table

The “Definition” column in the pin constants table above describes the values available for each pin when used with the PinIO class member function “function”. For example, if pin P1-30 was to be configured for GPIO, then it would be written as:

```
Pins[30].function( PIN_30_GPIO );
```

Or, if UART 0 clear-to-send signal functionality is needed on pin P1-30, then it would be written as:

```
Pins[30].function( PIN_30_UART0_CTS );
```

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 (some pins that have only one alternate function may use either ‘1’ or ‘2’, while others that have no alternate function will use ‘1’ or ‘3’ for the primary function):

3: Primary Function
2: Alternate Function 1
1: Alternate Function 2
0: GPIO

These values are used to set the bits in a pin’s respective pin assignment register to configure for a specific function.

PinIO Class Member Functions

Using the PinIO 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, to configure pin P1-19 (Timer Input 0) as GPIO and set it high without the PinIO class, it would be written like this:

```
#include <sim5441x.h>

sim1.gpio.par_timer &= ~0x03;       // Configure pin P1-19 for GPIO
sim1.gpio.ppdsdr_e = 0x80;          // Set bit to be driven out on pin
sim1.gpio.pddr_e |= 0x80;           // 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[19].function(PIN_19_GPIO);     // Configure pin P1-19 for GPIO
Pins[19] = 1;                       // Drive 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[25].set(); Pins[25] = 1;
void clr()	Set output low	Pins[27].clr(); Pins[27] = 0;
BOOL read()	Read pin high/low state	BOOL bpinstate = Pins[12]; if (!Pins[9]) iprintf (“The pin is low”);
void hiz()	Set output to tristate (high impedance input)	Pins[23].hiz();
void drive	Turn output on (opposite of tristate)	Pins[38].drive();
void function()	Set pin to special function or GPIO	Pins[49].function(PIN_49_GPIO); Pins[34].function(PIN_34_UART1_TXD);

Program Example

```
/******************************************************************************
 * LED BINARY COUNTER
 *
 * This program configures multiple signal pins as GPIO outputs to utilize the
 * available general-purpose LEDs on the NANO-DEV-100CR development board as a
 * visual binary counter, depending on whether this program is compiled for the
 * NANO54415 platform. The LED with the lowest numerical designator ID is used
 * as the least significant bit. The binary counter increments once every
 * second.  Once the counter reaches the maximum value that the LEDs are capable
 * of displaying, the counter will automatically reset back to zero.
 */

#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 = "MCF5441X-LedBinaryCounter";


/*
 * Translate the value of the counter into binary form to be represented on the
 * LEDs of the development board being used
 */
void incrementLeds(BYTE nCount)
{
    /*
     * On the NANO-DEV-100CR, setting the pin turns on the LED, clearing the
     * pin turns off the LED
     */
    if (nCount & 0x01) Pins[19] = 1;    // LED2
    else Pins[19] = 0;

    if (nCount & 0x02) Pins[21] = 1;    // LED3
    else Pins[21] = 0;

    if (nCount & 0x04) Pins[23] = 1;    // LED4
    else Pins[23] = 0;

    if (nCount & 0x08) Pins[25] = 1;    // LED5
    else Pins[25] = 0;
}
```

```
/**
 * The main task
 */
void UserMain(void *pd)
{
    InitializeStack();
    if (EthernetIP == 0) GetDHCPAddress();
    OSChangePrio(MAIN_PRIO);
    EnableAutoUpdate();

    iprintf("Application started\r\n");

    /*
     * Configure NANO54415 pins P1-19, 21, 23, and 25 as GPIO
     */
    Pins[19].function(PIN_19_GPIO);     // NANO-DEV-100CR - LED2
    Pins[21].function(PIN_21_GPIO);     // NANO-DEV-100CR - LED3
    Pins[23].function(PIN_23_GPIO);     // NANO-DEV-100CR - LED4
    Pins[25].function(PIN_25_GPIO);     // NANO-DEV-100CR - LED5

    BYTE nCounter = 0;      // For tracking the binary counting of the LEDs

    while (1) {
        /*
         * Increment the counter once every second
         */
        OSTimeDly(TICKS_PER_SECOND);
        incrementLeds(nCounter++);

        /*
         * Since the NANO-DEV-100CR is limited to only four LEDs, the counter
         * must be reset back to zero once 0x0F is reached
         */
        if (nCounter == 0x10) nCounter = 0;
    }
}
```