# SB72 GPIO Configuration

# Application Note

Revision 1.0
April 21, 2005
Initial Release

## *Introduction*

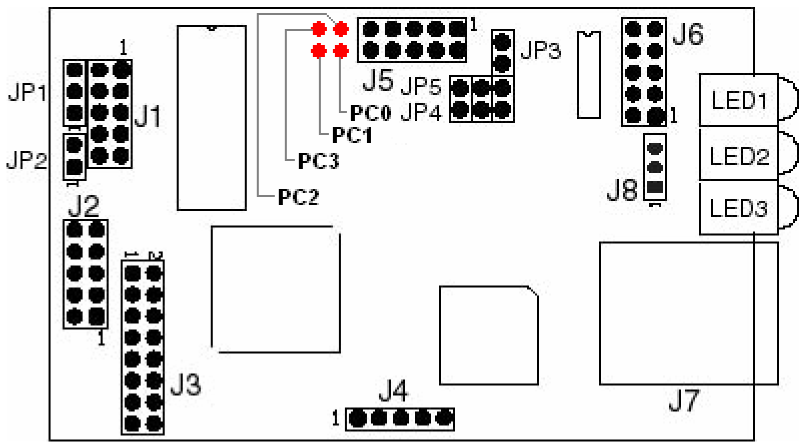
The NetBurner SB72 is capable of providing up to four general-purpose input/output signals from the Motorola ColdFire 5272 processor. These pins can be used either as their primary function or as GPIO, and the specific pins that make sense to use as GPIO will depend on which peripherals are needed for your specific application. A great reference for information contained in this document is "Chapter 17 – General Purpose I/O Module" of the MCF5272 User's Manual.

## *Port C (PC[3:0]) Pins*

Port C has a data direction and data register, but no control register. It is enabled only when the external data bus is 16 bits wide. This is done by holding QSPI_DOUT/WSEL high during reset. When QSPI_DOUT/WSEL is low during reset, the external data bus is 32 bits wide and port C is unavailable. On the SB72, this signal is set high on reset with a pull-up resistor, allowing port C to be enabled by default (see MCF5272 User's Manual, Section 17.3 and 17.4 on the usage of the port C data direction and data registers).

| SB72 32-Bit Function | SB72 16-Bit GPIO Function | GPIO Signal Description | Data Direction/Data Control Bits |
|---|---|---|---|
| D3 | PC3 | Port C Bit 3 | PCDDR3 / PCDAT3 |
| D2 | PC2 | Port C Bit 2 | PCDDR2 / PCDAT2 |
| D1 | PC1 | Port C Bit 1 | PCDDR1 / PCDAT1 |
| D0 | PC0 | Port C Bit 0 | PCDDR0 / PCDAT0 |

The four holes used for GPIO (PC0, PC1, PC2, and PC3) are located on the SB72 module diagram below in red.



## *Code Examples:*

```
// Port C Data Direction Register. Setting a bit configures a pin for
// output, and clearing a bit configures for input. The following sets
// the four GPIO holes as outputs.
sim.pcddr |= 0xF;

// Port C Data Register. The following writes to the register to set
// PC0 and PC2 low, and PC1 and PC3 high.
sim.pcdat = 0xA;

// Port C Data Register. The following register read returns the
// instantaneous values of the four GPIO holes.
WORD value_pc = sim.pcdat & 0xF;
```

The following example program alternates low and high settings of the odd and even-numbered GPIO signals every three seconds. At the end of each cycle, the data register is read, and its values are stored.

```
#include "predef.h"
#include <stdio.h>
#include <ctype.h>
#include <startnet.h>
#include <autoupdate.h>
#include <dhcpclient.h>

#include "..\sb72\system\sim5272.h"    // For access to processor
                                          registers
extern "C"
 {
   void UserMain( void *pd );
 }

const char *AppName = "SB72gpio";

void UserMain( void *pd )
 {
   InitializeStack();

   if ( EthernetIP == 0 )
    {
      if ( GetDHCPAddress() == 0 )
       {
         iprintf( "Get DHCP address: SUCCESS\r\n" );
       }
      else
       {
         iprintf( "Get DHCP address: FAILED\r\n" );
       }
    }
   else
    {
      iprintf( "Static IP address currently set\r\n" );
    }
   OSChangePrio( MAIN_PRIO );
   EnableAutoUpdate();
   StartHTTP();
   iprintf( "Application started\r\n\r\n" );

   //
   // Configure PC[3:0] as GPIO output signals. No control register is
   // needed since they are configured as GPIO by default on the SB72.
   //
   sim.pcddr |= 0xF;

   WORD value_pc;

   while ( 1 )
    {
      sim.pcdat = 0x5;               // Sets PC0 and PC2 high, PC1 and
                                        PC3 low

      OSTimeDly( 60 );               // 3-second delay

      sim.pcdat = 0xA;               // Sets PC1 and PC3 high, PC0 and
                                        PC2 low

      OSTimeDly( 60 );               // 3-second delay

      value_pc = sim.pcdat & 0xF;    // Reads GPIO pins and stores it in
                                        value_pc
    }
}
```