5405 Morehouse Drive, Ste 350
San Diego, CA 92121
858.558.0293 phone
858.558.8549 fax

# Databus IO Expansion

Date: November 2, 2012
Revision 1.0

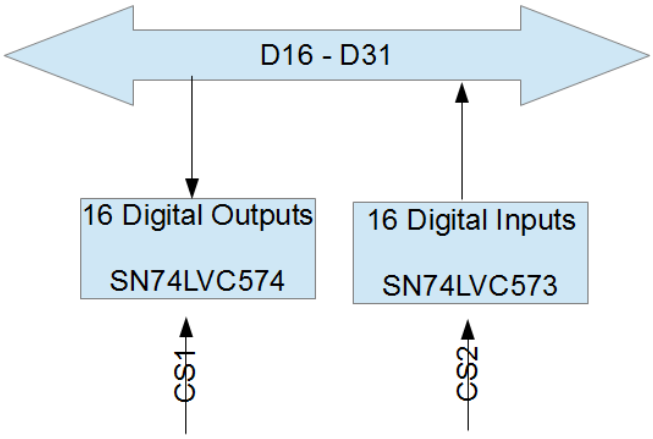
## 1. Hardware Compatibility

- MOD5234-100IR, 200IR
- MOD5270-100IR, 200IR
- MOD5272-100IR, 200IR
- MOD5282-100IR, 200IR
- MOD54415x-100IR, 200IR

## 2. Introduction

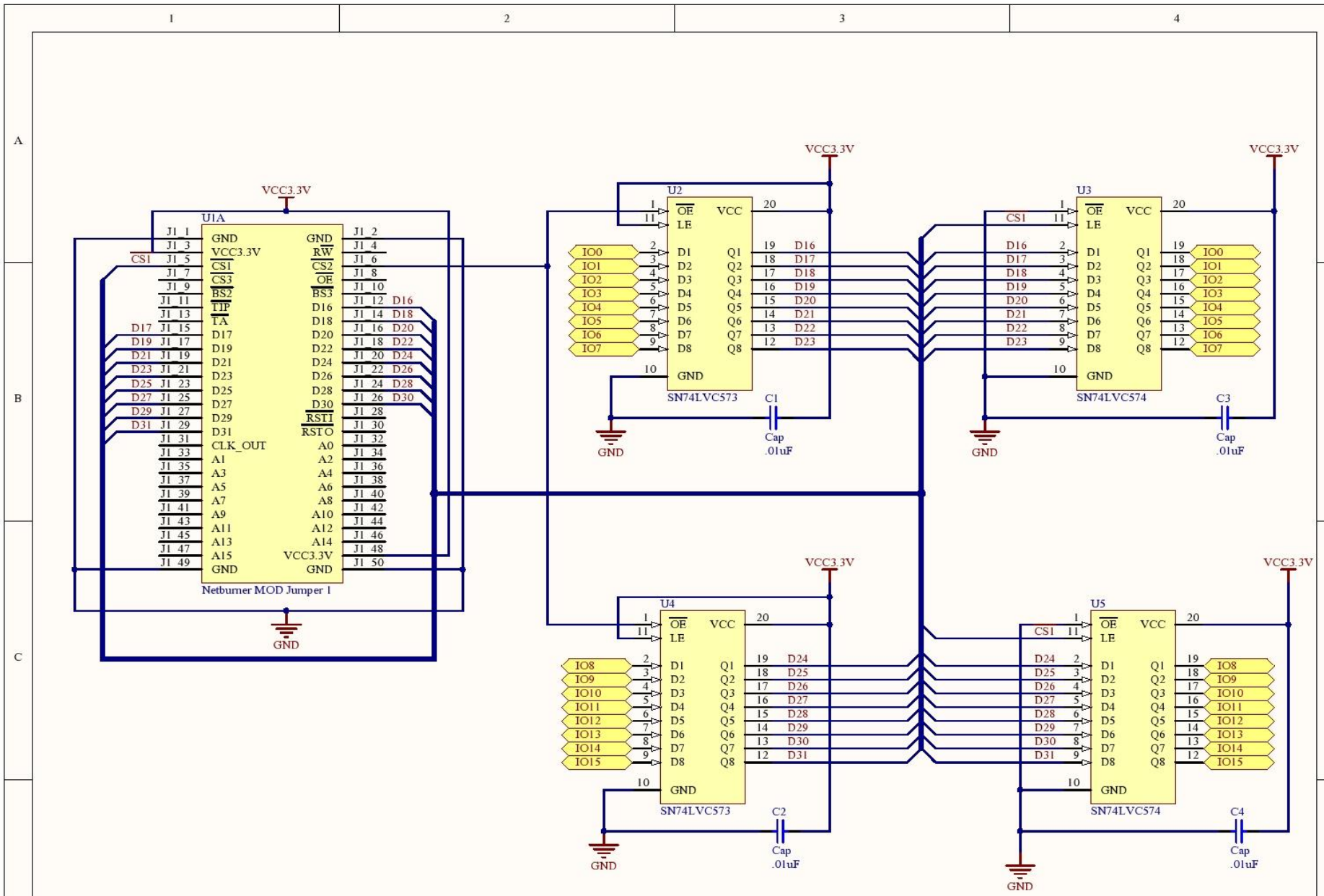
While many of the pins on the NetBurner modules can be configured to run as either their primary function or GPIO, from time to time applications are designed that require the use of more GPIO than is available on the platform. At this point, there are several options that a designer can take to increase the amount of IO pins available to them.

One of the simplest methods of greatly expanding the array of pins available is to make use of the data bus lines which consist of pins 12, 14-27, and 29 on jumper 1. With the use of the chip select modules, sections of memory can be monitored and used specifically for input and output. However, because these memory locations are not the only calls to memory being processed by your program, flip flops and latches can be used to store the input and output values until the processor is ready to sample or write to the IO. A block diagram for this process is given below.




## 3. Hardware Design

The schematic for the example GPIO expansion design is shown on the next page. This design supports any of the Netburner modules with two 50-pin jumpers. The schematic for jumper 1 of these modules is shown on the left side of the page, and SN74LVC573 and SN74LVC574 latches are shown on the right hand side of the page. Other than the latches and the jumper, there are power coupling capacitors attached between the power pins on the latches. As the latches expect between 3 and 5 volts of power, connecting them to the power lines from these modules should be sufficient.




| | | | |
|---|---|---|---|
| Title: *Data Bus GPIO Expansion* | | | NetBurner, Inc.<br>5405 Morehouse Drive<br>Suite 200<br>San Diego, CA 92121-4724<br>USA |
| Size: A | Model | Revision: 1.0 | |
| Date: 11/2/2012 | Time: 2:09:49 PM | Sheet 1 of 1 | |
| File: gpio_expansion.SchDoc | | | |

# 4. Bill of Material

| Used | Part Type | Designator | Vendor | Part Number | Description |
|---|---|---|---|---|---|
| 4 | 0.01uF Cap | C1 C2 C3 C4 | DIGIKEY | BC2662TB-ND | CAP 10000pF 50V |
| 2 | 74LVC573 Latch | U2 U4 | DIGIKEY | SN74LVC573ANE4-ND | IC D-TYP LATCH OCT TRI-ST 20-DIP |
| 2 | 74LVC574 Flip-Flop | U3 U5 | DIGIKEY | SN74LVC574ANE4-ND | IC OCT EDG-TRG D-TYP F-F 20-DIP |

# 5. Software Drivers

## 5.1. *Functionality*

Essentially, the additional hardware works with the processor by reading or writing the GPIO pins whenever certain locations in memory are read or written. The chip select modules when active set the chip select signal line low whenever an address matching the first 16 address bits (or less depending on the masking register) being read from memory match the base address assigned to the address register. For this reason, it is important to use an area of memory not used for other purposes or the chip select line will rewrite your gpio. Another important note is that while the first 16 bits of address must match, any address matching the last 16 address bits will rewrite your gpio. With this in mind it is important to note that while example given here allows for an increase of 16 gpio pins, using additional logic, the increase number of gpio pins available could be increased significantly by simple using the address lines to multiplex the inputs and outputs.

For the purposes of this application note, the following example code is broken into 3 parts: defining the addresses and prototypes for functions to read and write the gpio, an initialization function to activate the chip select modules, and a few functions to communicate with our inputs and outputs.

## 5.2. *Definitions*

As stated earlier, the chip select module operates with certain memory ranges. The following memory range is unused for the 5234, 5270, and 5282 modules and are used here for example. The Boolean in this example keeps track of whether the chip select modules are initialized for gpio or not.

```
//--- Chip select memory address definitions ---
#define BASEADDRESS                 0x10000000
#define IO_DIG_WR                   BASEADDRESS
#define IO_DIG_RD                   BASEADDRESS+0x1000000

//--- Variable definitions ---
BOOL bInit;

//--- Function prototypes ---
void InitChipBoardSelect12(void);
void WriteToOutputs(unsigned short);
unsigned short ReadFromInputs(void);
```

## 5.3. *Initialization Functions*

The following functions simply initialize or disable the chip select modules to be used. The initialize in this example works for the 5234, 5270, and 5282 modules, but the register information for the other modules can be found by reading through the user manuals for the other modules.

```
/*
 * InitChipBoardSelect12
 * Initializes the chip select modules used in reading and writing to the
 * data bus as gpio.
 */
void InitChipBoardSelect12( void )
{
        //Enable the Write Chip Select
        sim.cs[1].cscr = 0x2180;            // 0010 0001 1000 0000
        sim.cs[1].csmr = 0x00000001;
        sim.cs[1].csar = ( IO_DIG_WR >> 16 );

        //Enable the Read Chip Select
        sim.cs[2].cscr = 0x2180;            // 0010 0001 1000 0000
```

```
	sim.cs[2].csmr = 0x00000001;
	sim.cs[2].csar = ( IO_DIG_RD >> 16 );

	bInit = TRUE;
}

/*
 * Disable the Chip Select Modules
 */
void DisableChipSelect12( void )
{
	sim.cs[1].csmr = 0x0;
	sim.cs[2].csmr = 0x0;
	bInit = FALSE;
}
```

## 5.4. *Using GPIO*

The following functions will communicate with our new gpio pins. Essentially all that is required is to make a memory call to the address locations corresponding to each chip select module. Again, if your application requires that you further increase your IO capabilities, simply introduce more external logic and remember to write to the appropriate address by adding their definitions to the base address definitions given in the definition section.

```
/*
 * WriteToOutputs
 */
void WriteToOutputs( unsigned short output )
{
	if (!bInit) InitChipBoardSelect12();
	*(unsigned short *)(IO_DIG_WR) = output;
}

/*
 * ReadFromInputs
 */
unsigned short ReadFromInputs( void )
{
	if (!bInit)		InitChipBoardSelect12();
	return *(unsigned short *) (IO_DIG_RD);
}
```