How to Interface the PowerQUICC II™ Pro and PowerQUICC III™ Local Bus Controller to NAND Flash

This application note describes how to interface the Local Bus Controller (LBC) of the MPC83*xx* PowerQUICC II™ Pro family and the 85*xx* PowerQUICC III™ family processors to NAND Flash. Table 1 shows the processors included in these families:

Table 1. MPC83xx/MPC85xx Devices

MPC83xx PowerQUICC II™ Pro Family	MPC85xx PowerQUICC III™ Family
MPC8349E	MPC8560
MPC8347E	MPC8555E
MPC8343E	MPC8548E
	MPC8547E
	MPC8545E
	MPC8543E
	MPC8541E
	MPC8540

Contents

1 Introduction

NAND Flash is increasingly used in embedded systems due to its low cost and high density. Since NAND Flash uses a different interface than the traditional NOR Flash, it requires some glue logic or dedicated logic to interface with a memory controller.

However, because of the flexibility of the UPM controller, MPC83*xx* and MPC85*xx* processors can interface with NAND Flash without glue logic.

2 Basics of the NAND Flash Interface

NAND Flash has the following signals:

Table 2. NAND Flash Signals

Name	Direction	Function
IO[0:7]	Input/Output	Used for command, data, address.
IO[8:15]	Input/Output	For x16 device only. Used for data.
$\overline{CS}$	Input	Chip select
ALE	Input	Address latch enable
CLE	Input	Command latch enable
$\overline{WE}$	Input	Write enable
$\overline{RE}$	Input	Read enable
$\overline{WP}$	Input	Write protect
R/$\overline{B}$	Open drain	Ready/Busy
$\overline{RESET}$	Input	Reset NAND Flash

There are several different types of transactions as shown in Figure 1 that have distinct timings: write_command, write_address, write_data, and read_data.

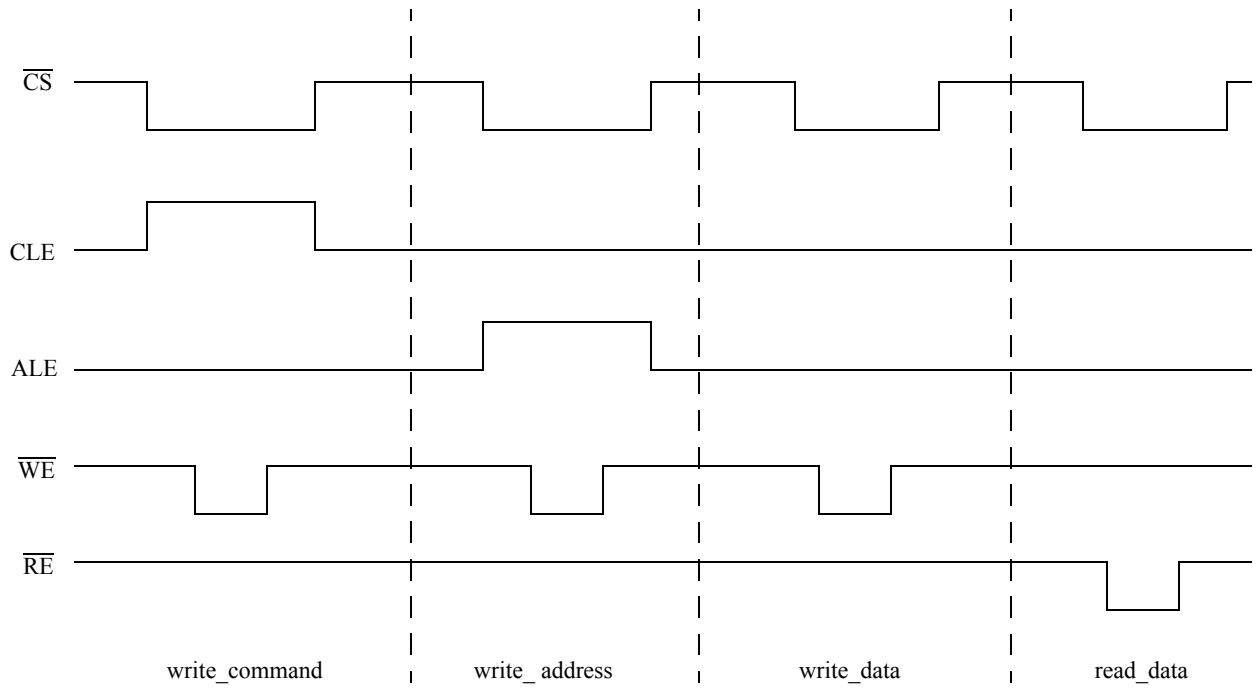

Figure 1. Waveform to NAND Transactions

For the basic operations, such as erase, write (program), or read, it is a sequence of the previous transactions.

For example, a write transaction follows this sequence:

write_command (0x80) => write_address(column address 1) => write_address(column address 2)=> write_address(row address 1) => write_address(row address 2) => write_data 1 => .... => write_data n => write_command (0x10)

The controller issues a write_command to tell the NAND Flash it intends to do a write. Then breaks the address into four pieces and transfers it to NAND in a predetermined order with write_address. Then the write data transfers to the NAND through write_data. The amount of write data can be up to one page. Finally, a command 0x10 issues to tell NAND that write has finished.

Read and erase transactions require a similar process, but have their own sequences.

3 How to Interface NAND with UPM

In order to interface with NAND, all the above waveforms must be producible. The desired waveforms are then invoked as needed and UPM is ideal for generating these waveforms. Figure 2 shows the connections used for an x16 devices. For an x8 device, LAD[0:7] should be connected to IO[7:0]. There are other ways to make the connections. However, one important requirement is that GPL2 must be connected to $\overline{RE}$ of NAND, so the GPCM is used after power-on-reset in order to boot from NAND Flash. $\overline{OE}$ of GPCM is GPL2 and it should be connected to $\overline{RE}$.

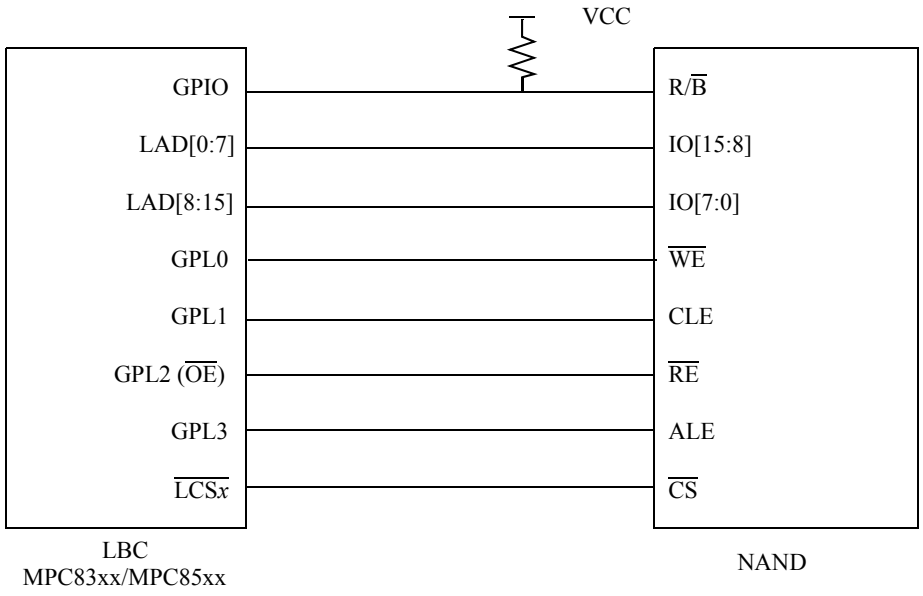

Figure 2. Connections between LBC and NAND

The Table 3 shows the RAM array allocation. For accessing the NAND Flash, all the required transactions including write_command, write_address, write_data, read_data, require only single beat accesses. This leaves the burst read/write area of UPM RAM array unused. Table 3 shows the allocation of transactions in the RAM array. The transaction write_address starts at 0x8 and the transaction write_data starts at 0x10. They occupy the normal burst read space.

Table 3. UPM RAM Array Allocation

UPM Routine	Start Address	Transaction
Read single beat	0x0	read_data
Read burst	0x8	write_command
	0x10	write_address
Write single beat	0x18	write_data
—	0x20–0x3F	for others if needed

The read_data and write_data transactions are easy to generate. These two patterns are in the normal single read and single write area of the UPM RAM array. Set MAMR[OP] = 00 (normal operation), and a read and write to the UPM memory bank generates the corresponding waveforms. However write_command and write_address, are located in the read burst area. To generate the waveform, MAMR[OP] needs to be set to 11 (run pattern) and MAMR[MAD] = 0x8 or 0x10. Then a write to the UPM memory bank runs the corresponding pattern.

The run pattern only generates the control signals through GPL*n*. The command/address needs to be passed to NAND Flash IO pins for write_command and write_address as well. During the run pattern cycles, LAD pins are actually tri-stated. Following describes a method to pass the info during run pattern:

Program UPM_ARRAY[AMX] to 11 for write_command and write_address pattern

Program MAR to the desired address/command value.

If AMX is set to 11, UPM will output MAR to LAD during ALE phase instead of the real address. Its intended purpose is to drive mode value to the address pins during DRAM initialization. But since LAD*n* are address/data muxed pins, MAR and AMX can also be used to pass the command/data as well. During the run pattern, LAD is driven to the value of MAR first, then it is tri-stated. Since no one else is driving the LAD, LAD will maitain MAR value during the whole run pattern duration.

A write to NAND translates into the following sequences in the software (a dummy write means a single byte write to the memory bank controlled by UPM):

1. Setting: MAMR[OP] = 11, MAMR[MAD] = 0x8, MAR = 0x80000000

   dummy write

   => Write command 0x80
2. Setting: MAMR[OP] = 11, MAMR[MAD] = 0x10, MAR = Column address 1

   dummy write

   => Write column address 1

NOTE

write_address pattern starts at MAD=0x10

3. Similar to 2), setting MAR = Column address 2

   dummy write

   => Write column address 2
4. Setting MAR = Row address 2

   dummy write

   => Write row address 1
5. Setting MAR = Row Address 2

   dummy write

   => Write row address 2
6. Setting: MAMR[OP] = 00

   normal write data 1 to UPM bank

   => Write data 1
7. .....

   normal write data n to UPM bank => Write data n
8. Setting: MAMR[OP] = 11, MAMR[MAD] = 0x8, MAR = 0x10000000

   dummy write

   => Write command 0x10 to terminate the NAND write sequence

NAND Flash is a very slow device. The access time is usually on the order of tens of milliseconds. So during read/write/erase sequences, after the command sequence, NAND will not be ready for other access for a very long time. The R/$\overline{B}$ signal can be connected to a GPIO signal. Software polls until R/$\overline{B}$ is negated.

4 How to Boot from NAND

Note: this section only describes the idea on how to boot from NAND flash. It has not been tested.

Most NAND Flash implements a nice feature called power-on autoread. After turning on the power, a certain amount of data transfers from memory cells to a buffer. The data in the buffer can then be accessed like normal NOR Flash without complex command sequences. After power-on reset, LBC defaults to GPCM for $\overline{CS}$0. For every GPCM access, it sets $\overline{CS}$ and $\overline{OE}$. Setting these two signals is all that is needed to read the data from the NAND Flash buffer.

Although NAND can be accessed with GPCM in this way, there is a difference between NAND and normal ROM. That is, NAND has no address pins. It has an internal address pointer, and the pointer always increments. This means that the boot code cannot do jumps and branches. For the NAND Flash tested, which has 8 Kbytes of autoread memory, the size is large enough that a simple method can be used to overcome this no jump limitation.

The following is the layout of first 8 Kbyte NAND image:

- Code to set up the DDR (No branch)
- Code to write partial boot code to DDR

- Jump to DDR

Use a simple assembly code to write partial boot code to DDR as follows:

```
addis r2, r0, 0x(Instruction_H_16bit)
ori r2, r2, 0x(Instruction_L_16bit)
stw r2, DDR_offset(r1)
```

These binary instructions are inserted into the immediate data field of the assembly code. For every three instructions in NAND, one instruction can be written to DDR. So for 8 Kbytes NAND, around 600 instructions can be written, which is enough to set up the UPM for NAND access. After this is completed, read more boot code from NAND pages beyond the first 8 Kbytes to DDR, and boot from DDR.

5 Sample Code for Setting UPM

```
////////////////////////////////////////////////////////////////
// NAND Flash programming example using UPM controller of MPC85XX //
////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
// # define ERASE; // for erasing NAND Flash
void BR_OR_init();
void UPMA_prog();
void run_pattern(int command, int pattern_offset);
void erase();
static unsigned long UPMATable[] =
{
0x0ff03c30,  0x0ff03c30,  0x0ff03c34,  0x0ff33c30, //Words 0 to 3
0xfff33c31,  0xffffffc30,  0xffffffc30,  0xffffffc30, //Words 4 to 7
0x0faf3c30,  0x0faf3c30,  0x0faf3c30,  0x0fff3c34, //Words 8 to 11
0xffff3c31,  0xffffffc30,  0xffffffc30,  0xffffffc30, //Words 12 to 15
0x0fa3fc30,  0x0fa3fc30,  0x0fa3fc30,  0x0ff3fc34, //Words 16 to 19
0xfff3fc31,  0xffffffc30,  0xffffffc30,  0xffffffc30, //Words 20 to 23
0x0ff33c30,  0x0fa33c30,  0x0fa33c34,  0x0ff33c30, //Words 24 to 27
0xfff33c31,  0xfff0fc30,  0xfff0fc30,  0xfff0fc30, //Words 28 to 31
0xfff3fc30,  0xfff3fc30,  0xfff6fc30,  0xfffcfc30, //Words 32 to 35
0xfffcfc30,  0xfffcfc30,  0xfffcfc30,  0xfffcfc30, //Words 36 to 39
0xfffcfc30,  0xfffcfc30,  0xfffcfc30,  0xfffcfc30, //Words 40 to 43
0xfffdfc30,  0xffffffc30,  0xffffffc30,  0xffffffc31, //Words 44 to 47
0xffffffc30,  0xffffffc00,  0xffffffc00,  0xffffffc00, //Words 48 to 51
0xffffffc00,  0xffffffc00,  0xffffffc00,  0xffffffc00, //Words 52 to 55
0xffffffc00,  0xffffffc00,  0xffffffc00,  0xffffffc01, //Words 56 to 59
0xffffffc00,  0xffffffc00,  0xffffffc00,  0xffffffc01  //Words 60 to 63
};

void main()
{
int i=0;
```

Sample Code for Setting UPM

```
short int m[4];
//Initialize BR6 and OR6 registers for UPMA controller
BR_OR_init();

//BR7 register for GPCM CS7  to generate reset for NAND Flash
// CS7 is connected NAND reset pin
*(int *)(0x40005038) = 0xc0001801;
//OR7 register
*(int *)(0x4000503c) = 0xffe010f1;
// assert CS7 pin to reset NAND Flash
*(int *)(0xc0000000) = 0x12ab5678;

//Program the RAM arrays of UPMA
UPMA_prog();

// It is used to erase NAND Flash
#ifdef ERASE

//Issue Read Stop command 0xf0 to stop the auto read
run_pattern(0x00f00000, 8);

//Erase a block of 8K bytes from address 0x00000000
erase();
while(1);

#endif

// starts the auto read from NAND Flash
m[0]= *(short int *)(0xf2000000) ;
m[1]= *(short int *)(0xf2000000) ;
m[2]= *(short int *)(0xf2000000) ;
m[3]= *(short int *)(0xf2000000) ;

//Issue Read Stop command 0xf0 to finish the read mode
run_pattern(0x00f00000, 8);
```

```
/*******************************************/
/* The sequence of write data into NAND Flash */
/*******************************************/
//(1)Data Input command 0x80
run_pattern(0x00800000, 8);

//(2)Address input cycle,
//write address CA(1)=0x00,CA(2)=0x00,SA(1)=0x00,SA(2)=0x00 into NAND Flash
run_pattern(0x00000000, 16);
run_pattern(0x00000000, 16);
run_pattern(0x00000000, 16);
run_pattern(0x00000000, 16);

//(3)Input data cycle
//Write the following data pattern into NAND Flash,
//the starting address is 0x00000000
*(short int *)(0xF2000000) = 0x1255;
*(short int *)(0xF2000000) = 0x5666;
*(short int *)(0xF2000000) = 0xab77;
*(short int *)(0xF2000000) = 0xef88;

//(4)Issue Program command 0x10 to finish NAND Flash write sequence
run_pattern(0x00100000, 8);

/*********************************/
/* End of NAND Flash write sequence*/
/*********************************/



/*******************************************/
/* The sequence of read data from NAND Flash */
/*******************************************/

//(1) Read mode command
```

```
run_pattern(0x00000000, 8);

//(2)Address input cycle,
//write address CA(1)=0x00, CA(2)=0x00,SA(1)=0x00,SA(2)=0x00 into NAND Flash
run_pattern(0x00000000, 16);
run_pattern(0x00000000, 16);
run_pattern(0x00000000, 16);
run_pattern(0x00000000, 16);

//(3)Read data cycle
// read data from the starting address 0x00000000 of NAND Flash
i= *(short int *)(0xF2000000) ;
i= *(short int *)(0xF2000000) ;
i= *(short int *)(0xF2000000) ;
i= *(short int *)(0xF2000000) ;

//(4)Issue Read Stop command 0xf0 to finish the read mode
run_pattern(0x00f00000, 8);

/*********************************/
/* End of NAND Flash read sequence*/
/*********************************/

while (1) { i++; } // loop forever
}


void BR_OR_init()
{


//BR6 for UPM, base address 0xF2000000, 16bit, UPMA, valid
*(int *)(0x40005030) = 0xF2001081;
//OR6
*(int *)(0x40005034) = 0xffff8001;
```

```
}

void UPMA_prog()
{

int i=0;
// OP set to write to RAM array command
*(int *)(0x40005070) = 0x10000000;

// Write word to RAM arrays

for (i=0;i<64;i++)
{
*(int *)(0x40005088) = UPMATable[i];
// dummy write
*(char *)(0xf2000000) = 0x0;
}

//change MAMR back to normal mode
*(int *)(0x40005070) = 0x00000000

//wait until the write of MAMR finished
while (*(int *)(0x40005070)!= 0x00000000);
}

void   run_pattern(command, pattern_offset)
{
int i;
//MAMR
*(int *)(0x40005070) = 0x30000000|pattern_offset;
//MAR
*(int *)(0x40005068) = command;
// dummy write
*(char *)(0xf2000000) = 0x0;
```

```
//change MAMR back to normal mode
*(int *)(0x40005070) = 0x00000000;

//wait until the write of MAMR finished

while (*(int *)(0x40005070) != 0x00000000);

}
void erase()
{

//Enter Erase Mode command 0x60
run_pattern(0x00600000, 8);
//Write address command, SA(1)=0x00, SA(2)=0x00
run_pattern(0x00000000, 16);
run_pattern(0x00000000, 16);
//Command 0xd0 erases the sector data
run_pattern(0x00d00000, 8);

}
```

6 Revision History

Table 4 provides a revision history for this application note.

Table 4. Document Revision History

Revision Number	Date	Substantive Change(s)
0	02/21/2005	Initial release.
1	10/17/2005	Figure 2, GPL3 now connects to NAND ALE. There is a mismatch between the example code and Table 3. Table 3 has been updated. The text following the table 3 has been revised. A note has been added to the section “How to boot from NAND” to indicate that it has not been tested.

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

How to Reach Us:

Home Page:
www.freescale.com

email:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
(800) 521-6274
480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Technical Information Center
3-20-1, Minami-Azabu, Minato-ku
Tokyo 106-0047 Japan
0120 191014
+81 3 3440 3569
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
(800) 441-2447
303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@
hibbertgroup.com

NANDFLASHWP
Rev. 1
10/2005

freescale™
semiconductor