

PowerQUICC™ UPM Configuration

by *Networking and Multimedia Group*
Austin, TX
Freescale Semiconductor, Inc.

This application note describes how to effectively program and use the universal programmable machine (UPM) in the PowerQUICC line of communication processors through Freescale's UPM software tools.

Residing within the memory controller on the PowerQUICC¹ line of products, the UPM provides a designer with the flexibility to interface the microprocessor with a myriad of unique peripherals or devices. Though perhaps daunting at first, programming the UPM is a relatively simple task, made easier through the use of Freescale's UPM Programming tools that are available on corresponding product web pages.

Contents

1. Universal Programmable Machine (UPM)	1
2. Programming the PQ1 UPM	7
3. PQ1 UPM Programming Tool	8
4. PQ1 UPM Tool Import Script	9
5. Differences between PQ1 and PQ2 UPM	10
6. Programming the PQ2 UPM	11
7. Differences in the PQ2 and PQ3 UPM	11
8. Programming the PQ3 UPM	11
9. PQ2/PQ3 UPM Tool	12
10. PQ2/PQ3 UPM Tool Import Script	12
11. Revision History	13
12. Appendix A	14
13. Appendix B	16

1 Universal Programmable Machine (UPM)

The UPM is available in the memory controller of the PQ1, PQ2, and PQ3. In its most basic form, the UPM consists of a RAM array of 64 locations deep and 32 bits wide used to define the state of external pins corresponding to internal clock edges during a memory access. The array itself has evolved somewhat, but the use of, as well as methodology of programming, has remained much the same throughout PowerQUICC families.

1. The PowerQUICC family includes PowerQUICC I, II, and III, which are abbreviated as PQ1, PQ2, and PQ3 in this application note.

The RAM array is divided into six patterns, indexed into the array shown in [Table 1](#):

Table 1. UPM Array Indices

Pattern	Index
Read single-beat pattern (RSS)	0 (0x0)
Read burst cycle pattern (RBS)	8 (0x8)
Write single-beat pattern (WSS)	24 (0x18)
Write burst cycle pattern (WBS)	32 (0x20)
Periodic timer pattern (PTS)	48 (0x30)
Exception pattern (EXSA)	60 (0x3C)

The memory controller has a prior knowledge of the starting index of these six patterns. For example, when a single write access is requested, the UPM will automatically start executing, starting the RAM words at index 24. Execution from the UPM table will continue until a LAST bit is detected in a RAM word.

[Figure 1](#) is an example of the UPM table for a Read Single Beat access for a PQ1 class device with CLKOUT and Core clock at a 1:1 ratio. The upper portion of this figure shows the desired waveforms to be generated by the memory controller, while the lower half of the figure shows how the UPM array is programmed to generate such waveforms. Note that the table is incomplete. RSS starts at offset 0x0 within the UPM table and will execute until the LAST bit is encountered at offset 0x2. Since RBS starts at offset 0x8 in the UPM table and RSS takes only three entries, we are left with five unused entries in the table.

In this example, GPL1 is asserted starting at the falling edge of GCLK2_50 as an example of how GPL signals are programmable via the UPM. This, as well as other UPM GPLx signals, could conceivably be used as an additional chip enable, a write-enable, RAS, CAS, or other miscellaneous control signal.

Unused entries in the table may either be left unprogrammed or used for additional functionality under the control of software via the Machine ‘x’ Mode Register (MxMR) with the “run” opcode.

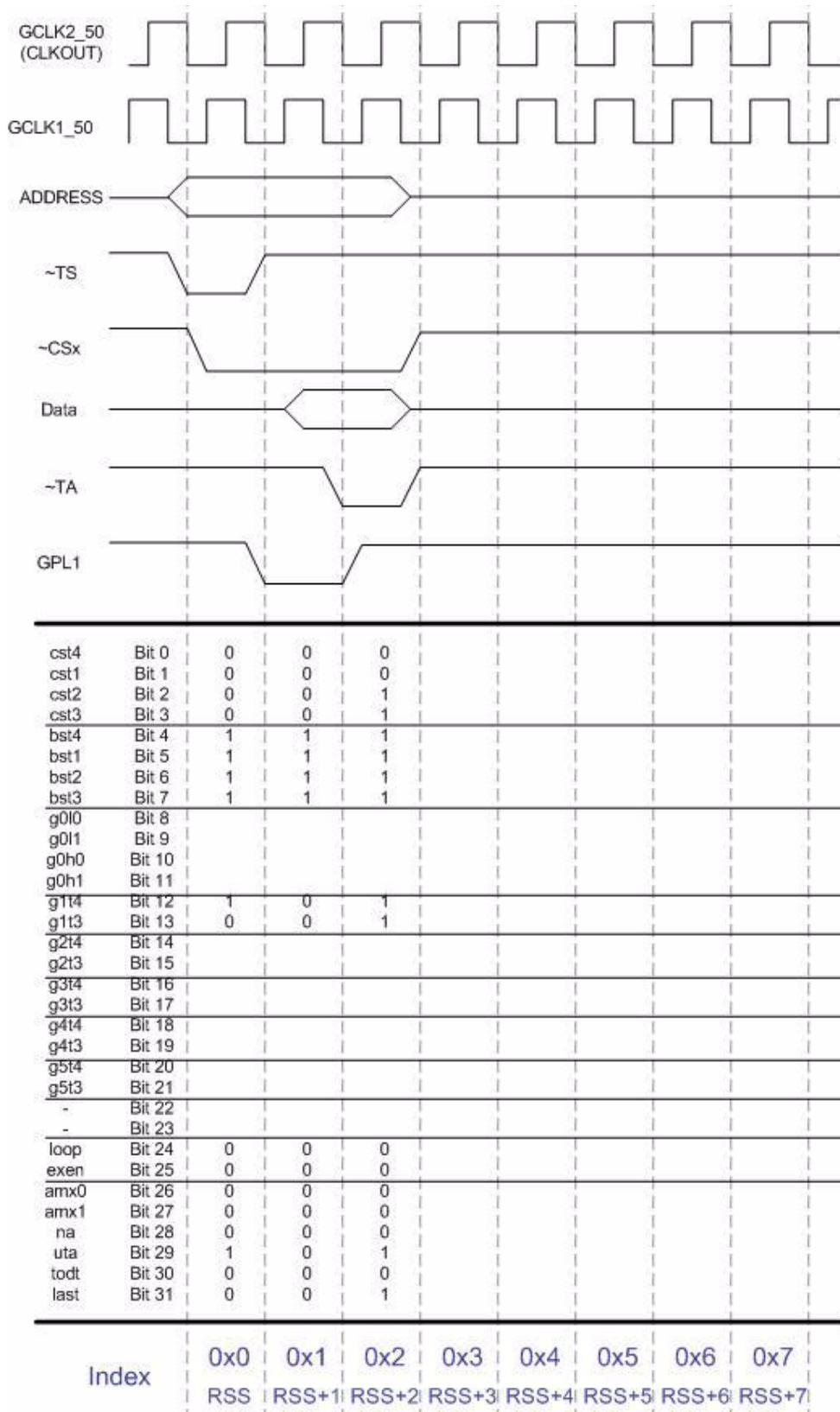


Figure 1. Example Read Single Beat UPM Pattern

Figure 2 shows the same example as in Figure 1, but with some of the extra words after the Read Single Beat access used for generating a precharge command. Specifically, offset 0x3 and 0x4 are used to issue a precharge command to SDRAM. GPL1 is used as \sim WE, GPL3 is RAS, and GPL5 is used as CAS. A precharge command is typically issued during SDRAM initialization. During initialization, to run the UPM pattern starting at offset 0x3 within the array, software would program the MCR register to MCR[OP]=10, MCR[MAD]=0011. The memory controller would then execute the pattern found at offset 0x3 and continue running patterns out of the UPM until a LAST bit is encountered, in this case, offset 0x4.

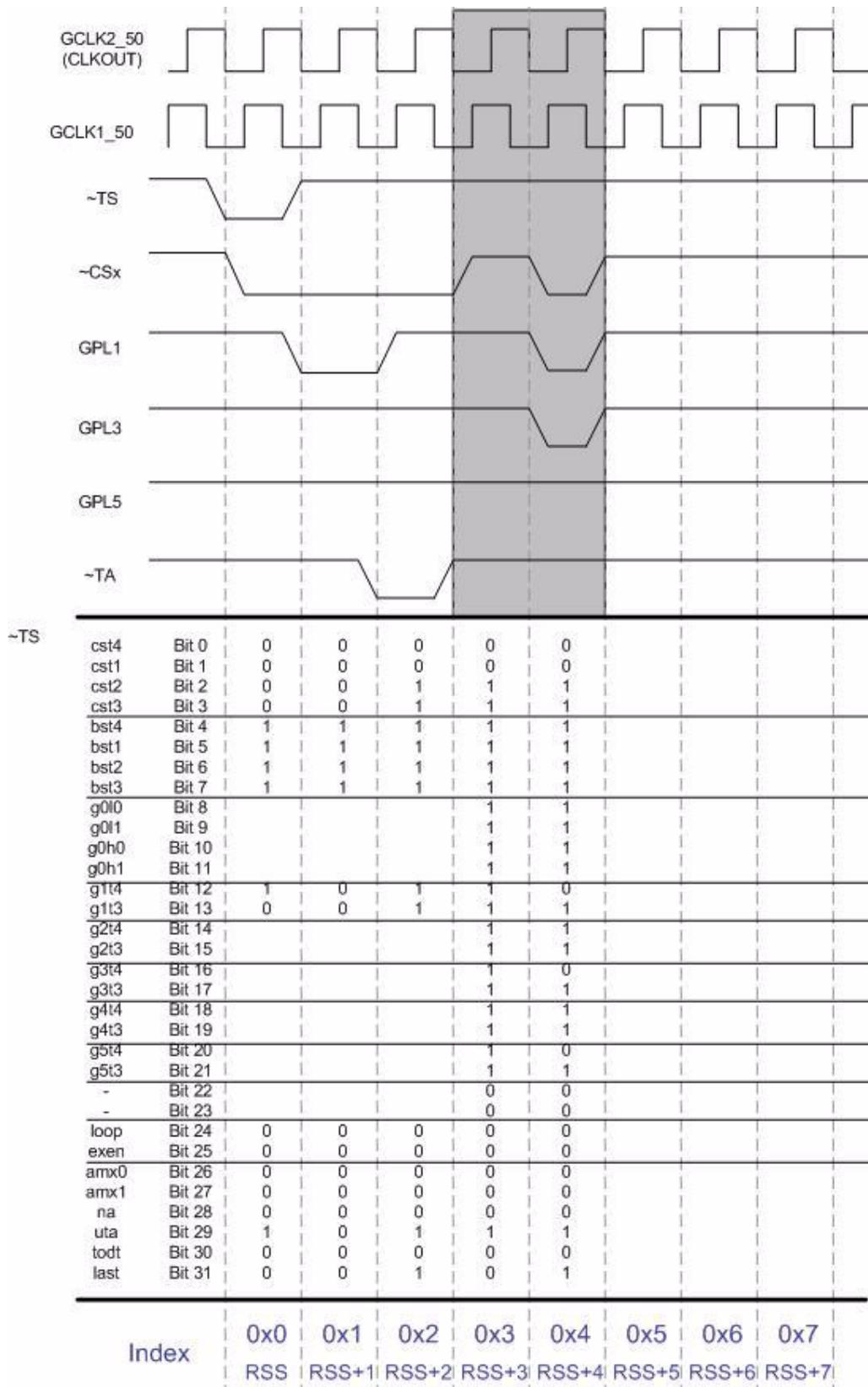


Figure 2. Use of Extra UPM space after RSS

1.1 Using Delays in the UPM Table

The **LOOP** bit field is available in the PQ1, PQ2, and PQ3. These bits indicate the beginning and end of a loop within the UPM table. The first time this bit is set while running a pattern indicates the beginning of a loop, the second time indicates the end of the loop. The corresponding loop will execute from 1 to 16, times depending on MxMR[RLFx] and MxMR[WLFx] bit settings.

In addition to LOOP, the PQ2 and PQ3 also have a **REDO** bit field in the UPM table. A REDO bit set in a UPM word will indicate to the UPM that the current word is to be executed multiple times. Special considerations should be taken when using REDO:

- LAST and REDO should not be set together
- REDO should not be used within the exception routine
- When LOOP and REDO are set together, the loop mechanism works as usual and the LOOP bit will be repeated according to the REDO function.
- When UTA and REDO are set together, PSDVAL is asserted for each execution of the RAM word.
- REDO may be used within a LOOP to lengthen UPM routines.

The Disable Timer (**TODT**) associated with the UPM allows for a minimum time guaranteed between two successive accesses to the same memory bank. This could be useful when DRAM requires RAS precharge time, or compact flash requires time to write. The disable timer period is determined in MxMR[DSx] and can be between 1 and 4 bus cycles. The disable timer does not affect memory accesses to different banks of memory. TODT is enabled differently in the PQ1 and PQ2/PQ3. To enable the disable timer in the PQ1, it may be set in the RAM table in the same RAM word in which LAST=1, or the word previous to that in which LAST=1. In the PQ2/PQ3, the disable timer must be set in the same RAM word in which LAST=1, otherwise it is ignored.

The UPM also includes an internal wait mechanism, enabled via the **WAEN** bit field, in order to allow for external devices to freeze the UPM in execution. The method for using the internal wait mechanism differs from PQ1 to PQ2 and PQ3.

The PQ1 wait mechanism differs for synchronous and asynchronous external masters. For a synchronous master, the external UPWAIT signal is sampled and synchronized by the memory controller upon reading a RAM word with the WAEN bit set. For the case where UPWAIT is asserted (active high) and sampled high on the previous falling edge of GCLK2 to that of when WAEN is set, the external signals will be frozen to the values defined by the UPM table on the next falling GCLK2. For an asynchronous master, ~AS is the external signal that activates the wait mechanism. The UPM enters a wait state if ~AS is sampled, asserted on the previous falling edge of GCLK2, and WAEN = 1 in the current RAM word. External signals are frozen after the falling edge of GCLK2 as programmed in the RAM word in which WAEN is set. The wait state is then exited when ~AS is negated, at which point all external signals controlled by the UPM are driven high asynchronously from the ~AS deassertion. Signals are driven in this state until the LAST bit is detected in a RAM word.

For the PQ2, if the UPM reads a RAM word with the WAEN bit set, the external (active high) UPWAIT signal is sampled and synchronized by the memory controller, and the current request is frozen. UPWAIT is sampled at the rising edge of CLKIN and, if UPWAIT is asserted and WAEN = 1 in the previous and current UPM word (WAEN needs to be set in two consecutive UPM words), the UPM is frozen until UPWAIT is negated. The value of the external pins driven by the UPM remains what it was in the previous

word read by the UPM. When UPWAIT is negated, the UPM continues on its normal function, executing the current cycle, as well as any remaining cycles, until a LAST bit is encountered. Note that during the wait cycles, the UPM negates \sim PSDVAL.

The PQ3's internal UPM wait mechanism is very similar to that of the PQ2, with the exception of the timing of the WAEN bit field. In the PQ3, WAEN no longer needs to be set in two consecutive UPM words, and the external UPWAIT signal is sampled and synchronized by the memory controller as if it were an asynchronous signal (within 1 bus cycle of WAEN assertion).

For more information on any of the aforementioned delay mechanisms, please refer to the corresponding product's reference manual.

2 Programming the PQ1 UPM

When programming the UPM on the PQ1, it is important to remember that the words within the RAM array specify the values of external signals at each edge of internal clocks GCLK1 and GCLK2. GCLK1 and GCLK2 may be different, depending on the ratio of core frequency to external bus clock.

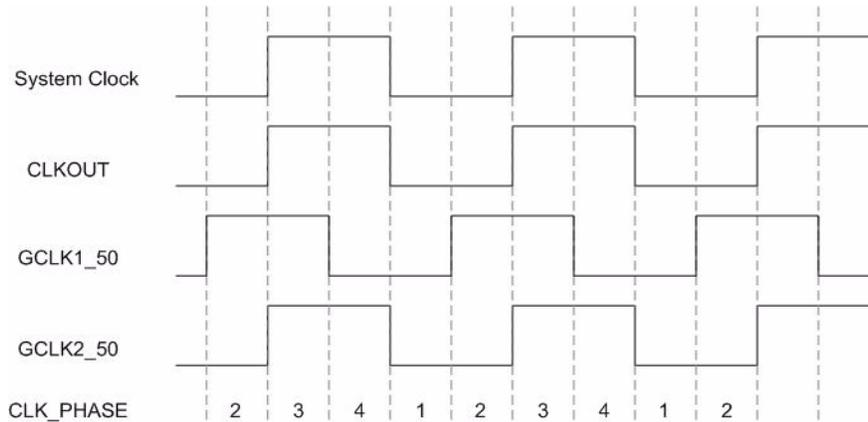


Figure 3. PQ1 UPM Clcking Scheme 1:1 Bus Clock

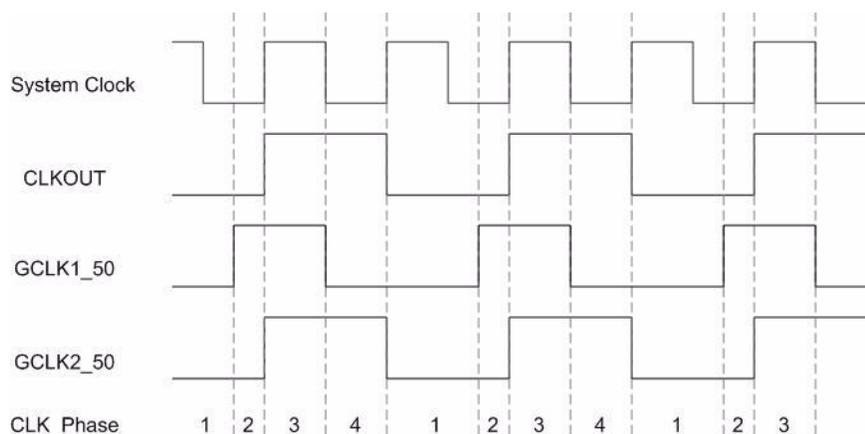


Figure 4. PQ1 UPM Clcking Scheme 1:2 Bus Clock

With a CLKOUT to core ratio of 1:1, as shown in [Figure 3](#), GCLK1 and GCLK2 each have a 50/50 duty cycle, enabling the user to program the UPM with true quarter system clock accuracy. As shown in [Figure 4](#), a CLKOUT to core ratio of 1:2 defines GCLK2 as a 50/50 duty cycle clock and GCLK1 as a 3/8 duty cycle clock. This configuration does not allow for true quarter clock programmability since UPM RAM words are output on clock edges of GCLK1 and GCLK2, which no longer line up at quarter system clock points.

The UPM RAM array is indirectly accessed via UPM specific registers in the PQ1. The MxMR defines most of the features of the specific UPM. The memory command and memory data registers (MCR and MDR) are used to initialize the UPM's RAM array. The memory address register (MAR) allows for software to initiate a pattern within the array to run.

The array must be initially programmed through software using a combination of these registers. A 32-bit word to be written to the array must be placed in the MDR register. Then the MCR register is written with the index into the array (0 to 63) of the corresponding data, as well as the write opcode.

The following is pseudocode for programming the array:

```

For i=0 to 63
    *mdr = upm_word++
    mcr = OP=WRITE | UPMx | MAD++;
next i;

```

UPM RAM array reads can be performed in the same fashion. In this case, the MCR register would be written with an index of the location within the table to read as well as the 'read' opcode. Upon execution, the corresponding UPM word would reside in the MDR register. This feature is especially useful for debugger scripts, as not all debuggers support direct access into the UPM RAM array.

It is also possible for software to initiate execution of a pattern, or portion of a pattern within the UPM array, as shown previously in [Figure 2](#). When executing an internal pattern, the MAR register will be utilized in order to drive an address on the external bus. MCR will be used to drive the pattern starting at the address provided in MCR[MAD]. The pattern will execute until a LAST bit is detected. For the example shown above in [Figure 2](#) we could leave MAR uninitialized, since the external address is unnecessary for a precharge command. MCR would be programmed with:

```

MCR[OP] = 10
MCR[MAD] = 0011

```

This would execute the pattern starting at offset 0x3, continuing until the LAST bit is encountered, in offset 0x4.

3 PQ1 UPM Programming Tool

The PQ1 UPM programming tool provides an easy-to-use GUI interface for programming the PowerQUICC I's UPM. It allows for either direct entry or entry via a waveform editor. The UPM Programming tool will take user input and will output the corresponding UPM table in ".s" assembly format for use by software.

The UPM Programming Tool can be found on the corresponding PQ1 product web page.

4 PQ1 UPM Tool Import Script

In order to import UPM tables back into the PQ1 UPM tool so that the data can be viewed as waveforms, the UPM table has to be converted to the proprietary “.mgp” format used by the UPM tool. “Appendix A” provides an example of a Perl script that can convert a UPM table from user software into the proprietary “.mgp” format required for use by the UPM tool.

A Perl interpreter is necessary in order to run the script. Unix/Linux distributions typically install Perl by default. For Windows, it is necessary to download and install a third party Perl interpreter, such as ActivePerl, from:

<http://www.activestate.com/Products/ActivePerl/>

Correct syntax for the script is:

```
./UPM_MGP.pl <inputfile> <outputfile>
```

Upon successfully converting user UPM tables, the user may open the “.mgp” outputfile from within the PQ1 UPM Tool. It is then possible to view the data as a waveform, make modifications to the table, and then re-export the table into an assembly file for use software.

The following is an example of the input format required for the script to correctly parse data:

```
# /* UPMA contents: */
UPMATable:
# /* single read. (offset 0 in upm RAM) */
    .long    0x0f0cfc04, 0x00af3c04, 0x0ffccc40, 0x1ffffc05
    .long    0xfffffc04, 0xefeeac34, 0x1fbd5c35, 0xfffffc04
# /* burst read. (offset 8 in upm RAM) */
    .long    0x0f0cfc04, 0x00ff3c04, 0x00fffc00, 0x00fffc00
    .long    0x00fffc00, 0x00fffc00, 0x1ffffc45, 0xfffffc04
    .long    0xfffffc05, 0xfffffc04, 0xfffffc04, 0xfffffc04
    .long    0xfffffc04, 0xfffffc04, 0xfffffc04, 0xfffffc04
# /* single write. (offset 18 in upm RAM) */
    .long    0x0f0cfc00, 0x00af0c04, 0x1ffddc45, 0xfffffc04
    .long    0xfffffc04, 0xfffffc04, 0xfffffc04, 0xfffffc04
# /* burst write. (offset 20 in upm RAM) */
    .long    0x0f0cfc00, 0x00ff0c00, 0x00fffc00, 0x00fffc00
    .long    0x01ffffc04, 0x1ffffc05, 0xfffffc44, 0xfffffc04
    .long    0xfffffc04, 0xfffffc04, 0xfffffc04, 0xfffffc04
    .long    0xfffffc04, 0xfffffc04, 0xfffffc04, 0xfffffc04
# /* refresh. (offset 30 in upm RAM) */
    .long    0x0ffc3c04, 0x0ffffc04, 0x1ffffc45, 0xfffffc04
    .long    0xfffffc04, 0xfffffc04, 0xfffffc04, 0xfffffc04
```

Differences between PQ1 and PQ2 UPM

```
.long    0xffffffffc04, 0xffffffffc04, 0xffffffffc04, 0xffffffffc04
# /* exception. (offset 3c in upm RAM) */
.long    0x0ffccc04, 0xffffffffc05, 0xffffffffc04, 0xffffffffc04
#
UPMTableEnd:
```

The script will parse the code segment, looking for the ‘0x’ signifying a RAM word entry. Upon completion, the script will inform the user of the number of RAM words encountered, which, for correct operation, should always be 64.

5 Differences between PQ1 and PQ2 UPM

As the PowerQUICC families evolved, changes were made in the implementation of the UPM. Below is a list of the major differences between the PQ1 and PQ2/PQ3 UPMs:

- First cycle timing transferred to the UPM array—In the PQ1 UPM, the first cycle value of some of the signals is determined from ORx[SAM,G5LA,G5LS]. This is eliminated in the PQ2. All signals are controlled only by the pattern written to the array.
- Timing of GPL[0:5]—In the PQ1 UPM, the GPL lines could change on the positive edge of clock phase 2 or 3 as shown in [Figure 3](#) and [Figure 4](#). In the PQ2 these signals can also change on the positive edge of clock phase 1 or 3.
- UPM controlled signals negated at end of an access—In the PQ1 UPM, if the user did not negate the UPM signals at end of an access, those signals kept their previous value. In the PQ2, all UPM signals are negated (CS, BS, GPL[0:4] driven to logic 1 and GPL5 driven to logic 0) at the end of that access, unless there is a back-to-back UPM access pending. This allows the UPM routine to finish one clock earlier because it is now possible, and desirable, to assert both UTA and LAST simultaneously.
- MCR is eliminated—In the PQ2, the function of RAM read/write and RUN is done through the MxMR.
- UTA polarity is reversed—In the PQ2, UTA is active high.
- The disable timer control (TODT) and LAST bit in the RAM array word must be set together, otherwise TODT is ignored.
- Refresh timer value is in a separate register—In the PQ2, the refresh timer value has moved to two registers, PURT and LURT, which can serve multiple UPMs.
- Refresh on the 60x bus must be done in UPMA.
- New feature—Repeated execution of the current RAM word (REDO bit) is available in PQ2/PQ3.
- Extended hold time on reads can be up to 8 clock cycles instead of 1 in the PQ1.
- Each UPM on the PQ1 has a wait signal. In the PQ2, the three UPMs share two wait signals (PUPMWAIT and LUPMWAIT).

6 Programming the PQ2 UPM

The PQ2 UPM is programmed in much the same way as the PQ1. The UPM RAM array is indirectly accessed via UPM specific registers. The MxMR registers define most of the features of the specific UPM. In the PQ2, the MCR register no longer exists and, instead, the memory data register (MDR) is used in conjunction with the MxMR register in order to initialize the UPM's RAM array. Finally, as in the PQ1, the memory address register (MAR) allows for software to initiate a pattern within the array to run.

The array must be initially programmed through software using a combination of these registers. A 32-bit word to be written to the array must be placed in the MDR register. Then, the MxMR register is written with the index into the array (0 to 63) of the corresponding data, as well as the write opcode.

The following is pseudocode for programming the array:

```

For i=0 to 63
    *mdr = upm_word++
    mxmr = OP =WRITE | UPMx | MAD++;
next i;

```

Running a pattern out of the UPM table, as shown previously in [Figure 2](#), also differs with the PQ2. MxMR[OP] would need to be set to 11 to indicate the RUN opcode, and MxMR[MAD] would be set with the patterns index into the array. Then, in order to run the pattern, a memory access is necessary anywhere within an address range that, via the ORx and BRx registers, accesses the corresponding UPM.

7 Differences in the PQ2 and PQ3 UPM

The only difference between UPM in the PQ2 and PQ3 is in the implementation of the UPM_WAIT signal. The PQ2 required that WAEN be set in two consecutive RAM words in order to enable sampling of UPWAIT. In the PQ3, WAEN no longer needs to be set in two consecutive words and the external UPWAIT signal is sampled and synchronized by the memory controller as if it were an asynchronous signal (within 1 bus cycle of WAEN assertion).

8 Programming the PQ3 UPM

With the exception of the differences mentioned above, the PQ3 UPM is functionally similar to the PQ2 UPM and patterns should be programmed as such. Due to platform differences on the two processor families, the actual programming of the array (that is, steps taken to insert a UPM program into the UPM RAM) differ slightly between the PQ2 and PQ3.

In order to program the PQ2 or PQ3 array one must follow the sequence below:

1. Set up BR_n and OR_n registers.
2. Write patterns into the RAM array.
3. Program MRTPR, LURT, and MAMR[RFEN] if refresh is required
4. Program MxMR

On PQ2, patterns are written to the RAM array by setting MxMR[OP]=01 and accessing the UPM with a single byte transaction.

On PQ3, patterns are similarly written to the RAM array by setting MxMR[OP]=01 and accessing the UPM, but with any write transaction to the relevant chip select. The entire PQ3 array is thus programmed by an alternating series of writes: to MDR each time followed by a read from MDR and then followed by a dummy write transaction to the relevant UPM assigned bank. Since writes/reads may occur out of order on the PQ3 platform, the read before the dummy write transaction is required to enforce proper ordering between updates to the MxMR register and the dummy accesses to the UPM memory region.

9 PQ2/PQ3 UPM Tool

The PQ2/PQ3 UPM programming tool provides an easy-to-use GUI interface for programming the PowerQUICC II/III's RAM table. It allows for either direct entry or entry via a waveform editor. The UPM Programming tool will take user input and will output the corresponding UPM table in ".s" assembly format for use by software.

The UPM Programming Tool can be found on the corresponding PowerQUICC II/III product web page.

10 PQ2/PQ3 UPM Tool Import Script

It may be desirable to import user data into the proprietary ".URF" format that the PQ2/PQ3 UPM Tool requires. "Appendix B" provides an example of a Perl script that will convert a UPM table from user code into the proprietary ".URF" format.

A Perl interpreter is necessary in order to run the script. Unix/Linux distributions typically install Perl by default. For Windows, it is necessary to download and install a third party Perl interpreter, such as ActivePerl, from:

<http://www.activestate.com/Products/ActivePerl/>

Correct syntax for the script is:

```
./UPM_URF.pl <inputfile> <outputfile>
```

Upon successfully converting user UPM tables, the user may open the ".URF" outputfile from within the PQ2/PQ3 UPM Tool. It is then possible to view the data as a waveform, make modifications to the table, and then re-export the table into an assembly file for use in software.

The following is an example of the input format required for the script to correctly parse data:

```
# /* UPMA contents: */
UPMTable:
# /* single read. (offset 0 in upm RAM) */
    .long    0x0f0cfc04, 0x00af3c04, 0x0ffccc40, 0x1ffffc05
    .long    0xfffffc04, 0xefeeac34, 0x1fbd5c35, 0xfffffc04
# /* burst read. (offset 8 in upm RAM) */
    .long    0x0f0cfc04, 0x00ff3c04, 0x00fffc00, 0x00fffc00
    .long    0x00fffc00, 0x00fffc00, 0x1ffffc45, 0xfffffc04
    .long    0xfffffc05, 0xfffffc04, 0xfffffc04, 0xfffffc04
```

```

        .long    0xffffffffc04, 0xffffffffc04, 0xffffffffc04, 0xffffffffc04
# /* single write. (offset 18 in upm RAM) */
        .long    0x0f0cfc00, 0x00af0c04, 0x1ffddc45, 0xffffffffc04
        .long    0xffffffffc04, 0xffffffffc04, 0xffffffffc04, 0xffffffffc04
# /* burst write. (offset 20 in upm RAM) */
        .long    0x0f0cfc00, 0x00ff0c00, 0x00fffc00, 0x00fffc00
        .long    0x01fffc04, 0x1fffc05, 0xfffc44, 0xffffffffc04
        .long    0xffffffffc04, 0xffffffffc04, 0xffffffffc04, 0xffffffffc04
        .long    0xffffffffc04, 0xffffffffc04, 0xffffffffc04, 0xffffffffc04
# /* refresh. (offset 30 in upm RAM) */
        .long    0x0ffc3c04, 0x0fffc04, 0x1fffc45, 0xffffffffc04
        .long    0xffffffffc04, 0xffffffffc04, 0xffffffffc04, 0xffffffffc04
        .long    0xffffffffc04, 0xffffffffc04, 0xffffffffc04, 0xffffffffc04
# /* exception. (offset 3c in upm RAM) */
        .long    0x0ffccc04, 0xfffc05, 0xfffc04, 0xffffffffc04
#
UPMTableEnd:

```

The script will parse the code segment, looking for the ‘0x’ signifying a RAM word entry. Upon completion, the script will inform the user of the number of RAM words encountered, which, for correct operation, should always be 64.

11 Revision History

Table 2 below provides a revision history of this document.

Table 2. Document Revision History

Rev. Number	Date	Substantive Change(s)
2	10/2009	Revised Section 8 , “Programming the PQ3 UPM.”
1	12/28/04	Replaced Figure 3 and made change to the second paragraph of Section 2 , “Programming the PQ1 UPM.”
0	11/18/04	Initial revision.

12 Appendix A

```
#!/usr/bin/perl
#    upm.pl          Rel 0.3   10/14/02
#    Copyright (c) 2004 FSL
#    All rights reserved.
#
#    Paul Genua
#    Freescale Semiconductor
#
# Usage: upm_MGP.pl INFILE OUTFILE
# Converts UPM data from an assembly file into "mgp" format
# to be read in by Freescale's UPM editor

$a=$ARGV[0];
$b=$ARGV[1];

print "upm_MGP.pl   Rel 0.3   10/14/02\n";
print "-- Convert UPM table to MPG format for PQ1 UPM tool\n\n";

open (OF,">$b") || die "cannot open $b for writing:  $!\n
        Correct usage: upm.pl INFILE OUTFILE\n";
print OF "set_device(\"MPC860\");;\n";
print OF "SystemClock = 16780.0;\n";
print OF "ReferenceClock = 32.768;;\n";

$count = 0;

open (EP,$a) || die "cannot open $a for reading:  $!\n
        Correct usage: upm.pl INFILE OUTFILE\n";
while ($line=<EP>)
{
    $line =~ tr/A-Z/a-z/;           #lowercase everything
    $linelen = length($line);
```

```

$delim=0;
while ($delim<$linelen)
    {
    $_=substr($line,$delim,2);
    if (/0x/)
        {
        $word=substr($line,$delim+2,8);
        $counthex = sprintf'%X',$count; #convert to hex
        print OF "set_upmram(0, \$$counthex, \$$word);;\n";
        $count = $count+1;
        } #if statement
    $delim=$delim+1;
    } #while linelen
} #while EP

close (EP) || die "can't close $a: $1";
close (OF) || die "can't close $b: $1";

print "Finished: found a total of $count words\n";
if($count=64)
    {
    print "UPM Table Complete\n";
    }
else
    {
    print "UPM Table Incomplete\n";
    }

```

13 Appendix B

```
#!/usr/bin/perl

#   upm_URF.pl           Rel 0.3   10/14/02

#   Copyright (c) 2004 FSL

#   All rights reserved.

#

#   Paul Genua

#   Freescale Semiconductor

#

# Usage: upm_URF.pl INFILE OUTFILE

# Converts UPM data from an assembly file into "URF" format

# to be read in by Freescale's UPM editor for PQ2 / PQ3

$a=$ARGV[0];
$b=$ARGV[1];

print "upm_URF.pl   Rel 0.1   10/14/02\n";
print "-- Convert UPM table to \"URF\" format for PQ2/PQ3 UPM tool\n\n";
open (OF,">$b") || die "cannot open $b for writing:  $!\n
        Correct usage: upm_URF.pl INFILE OUTFILE\n";

$preface=pack("C*",0x55,0x50,0x4d,0x5f,0x52,0x61,0x6d,0x5f,0x45,0x64,0x69,0x74,0x6f,0x72,0x5f
,0x32,0x2e,0x30,0x2e,0x30,0x50,0x51,0x49,0x49,0x0d,0x0d,0x0a,0x30,0x0d,0x0d,0x0a,0x31,0x30,0x
0d,0x0d,0x0a,0x42,0x52,0x41,0x20,0x50,0x53,0x20,0x31,0x20,0x45,0x4f,0x46,0x49,0x45,0x4c,0x44,
0x20,0x0d,0x0d,0x0a,0x42,0x52,0x42,0x20,0x50,0x53,0x20,0x31,0x20,0x45,0x4f,0x46,0x49,0x45,0x4
c,0x44,0x20,0x0d,0x0d,0x0a,0x42,0x52,0x43,0x20,0x50,0x53,0x20,0x31,0x20,0x45,0x4f,0x46,0x49,0
x45,0x4c,0x44,0x20,0x0d,0x0d,0x0a,0x4c,0x43,0x52,0x52,0x20,0x43,0x4c,0x4b,0x44
,0x49,0x56,0x20,0x31,0x30,0x30,0x30,0x20,0x45,0x41,0x44,0x43,0x20,0x31,0x20,0x45
,0x4f,0x46,0x49,0x45,0x4c,0x44,0x20,0x0d,0x0d,0x0a,0x4d,0x41,0x4d,0x52,0x20,0x42
,0x53,0x45,0x4c,0x20,0x30,0x20,0x52,0x46,0x45,0x4e,0x20,0x30,0x20,0x4f,0x50,0x20
,0x30,0x20,0x55,0x57,0x50,0x4c,0x20,0x30,0x20,0x41,0x4d,0x20,0x30,0x20,0x44,0x53
,0x20,0x30,0x20,0x47,0x4f,0x43,0x4c,0x58,0x20,0x30,0x20,0x47,0x50,0x4c,0x34,0x20
,0x31,0x20,0x52,0x4c,0x46,0x58,0x20,0x30,0x20,0x57,0x4c,0x46,0x58,0x20,0x30,0x20
,0x54,0x4c,0x46,0x58,0x20,0x30,0x20,0x4d,0x41,0x44,0x20,0x30,0x20,0x45,0x4f,0x46
,0x49,0x45,0x4c,0x44,0x20,0x0d,0x0d,0x0a,0x4d,0x42,0x4d,0x52,0x20,0x42,0x53,0x45
,0x4c,0x20,0x30,0x20,0x52,0x46,0x45,0x4e,0x20,0x30,0x20,0x4f,0x50,0x20,0x30,0x20
,0x55,0x57,0x50,0x4c,0x20,0x30,0x20,0x41,0x4d,0x20,0x30,0x20,0x44,0x53,0x20,0x30
,0x20,0x47,0x4f,0x43,0x4c,0x58,0x20,0x30,0x20,0x47,0x50,0x4c,0x34,0x20,0x31,0x20
,0x52,0x4c,0x46,0x58,0x20,0x30,0x20,0x57,0x4c,0x46,0x58,0x20,0x30,0x20,0x54,0x4c
,0x46,0x58,0x20,0x30,0x20,0x4d,0x41,0x44,0x20,0x30,0x20,0x45,0x4f,0x46,0x49,0x45
,0x4c,0x44,0x20,0x0d,0x0d,0x0a,0x4d,0x43,0x4d,0x52,0x20,0x42,0x53,0x45,0x4c,0x20
,0x30,0x20,0x52,0x46,0x45,0x4e,0x20,0x30,0x20,0x4f,0x50,0x20,0x30,0x20,0x55,0x57
```

```
,0x50 ,0x4c ,0x20 ,0x30 ,0x20 ,0x41 ,0x4d ,0x20 ,0x30 ,0x20 ,0x44 ,0x53 ,0x20 ,0x30 ,0x20 ,0x47
,0x4f ,0x43 ,0x4c ,0x58 ,0x20 ,0x30 ,0x20 ,0x47 ,0x50 ,0x4c ,0x34 ,0x20 ,0x31 ,0x20 ,0x52 ,0x4c
,0x46 ,0x58 ,0x20 ,0x30 ,0x20 ,0x57 ,0x4c ,0x46 ,0x58 ,0x20 ,0x30 ,0x20 ,0x54 ,0x4c ,0x46 ,0x58
,0x20 ,0x30 ,0x20 ,0x4d ,0x41 ,0x44 ,0x20 ,0x30 ,0x20 ,0x45 ,0x4f ,0x46 ,0x49 ,0x45 ,0x4c ,0x44
,0x20 ,0x0d ,0x0d ,0x0a ,0x4f ,0x52 ,0x41 ,0x20 ,0x45 ,0x41 ,0x44 ,0x20 ,0x30 ,0x20 ,0x45 ,0x4f
,0x46 ,0x49 ,0x45 ,0x4c ,0x44 ,0x20 ,0x0d ,0x0d ,0x0a ,0x4f ,0x52 ,0x42 ,0x20 ,0x45 ,0x41 ,0x44
,0x20 ,0x30 ,0x20 ,0x45 ,0x4f ,0x46 ,0x49 ,0x45 ,0x4c ,0x44 ,0x20 ,0x0d ,0x0d ,0x0a ,0x4f ,0x52
,0x43 ,0x20 ,0x45 ,0x41 ,0x44 ,0x20 ,0x30 ,0x20 ,0x45 ,0x4f ,0x46 ,0x49 ,0x45 ,0x4c ,0x44 ,0x20
,0x0d ,0x0d ,0x0a );
```

```
print OF "$preface";
```

```
$count = 0;                                #UPMA
open (EP,$a) || die "cannot open $a for reading:  $!\n
    Correct usage: upm.pl INFILE OUTFILE\n";
while ($line=<EP>)
{
    $line =~ tr/A-Z/a-z/;                    #lowercase everything
    $linelen = length($line);

    $delim=0;
    while ($delim<$linelen)
    {
        $_=substr($line,$delim,2);
        if (/0x/)
        {
            $word=substr($line,$delim+2,8);
            $word_hex= hex $word;
            $word_dec = sprintf'%U',$word_hex; #convert to hex
            print OF "0 $count $word_dec\n";
            $count = $count+1;
        }
        $delim=$delim+1;                    #if statement
    }
} #while linelen
} #while EP
```

```
$count=0;                                #UPMB
```

Appendix B

```

while ($count<64) {
    print OF "1 $count 4294966272\n";
    $count = $count+1;
}

$count=0;                #UPMC
while ($count<64) {
    print OF "2 $count 4294966272\n";
    $count = $count+1;
}

print OF "8\n0 0 0 0\n0 0 1 0\n0 0 2 0\n0 0 3 0\n0 0 4 0\n0 0 5 0\n0 0 6 0\n0 0 7 0\n16\n0 1 0
0\n0 1 1 0\n0 1 2 0\n0 1 3 0\n0 1 4 0\n0 1 5 0\n0 1 6 0\n0 1 7 0\n0 1 8 0\n0 1 9 0\n0 1 10 0\n0
1 11 0\n0 1 12 0\n0 1 13 0\n0 1 14 0\n0 1 15 0\n8\n0 2 0 0\n0 2 1 0\n0 2 2 0\n0 2 3 0\n0 2 4
0\n0 2 5 0\n0 2 6 0\n0 2 7 0\n16\n0 3 0 0\n0 3 1 0\n0 3 2 0\n0 3 3 0\n0 3 4 0\n0 3 5 0\n0 3 6
0\n0 3 7 0\n0 3 8 0\n0 3 9 0\n0 3 10 0\n0 3 11 0\n0 3 12 0\n0 3 13 0\n0 3 14 0\n0 3 15 0\n12\n0
4 0 0\n0 4 1 0\n0 4 2 0\n0 4 3 0\n0 4 4 0\n0 4 5 0\n0 4 6 0\n0 4 7 0\n0 4 8 0\n0 4 9 0\n0 4 10
0\n0 4 11 0\n4\n0 5 0 0\n0 5 1 0\n0 5 2 0\n0 5 3 0\n8\n0 0 0 0\n1 0 1 0\n1 0 2 0\n1 0 3 0\n1 0
4 0\n1 0 5 0\n1 0 6 0\n1 0 7 0\n16\n1 1 0 0\n1 1 1 0\n1 1 2 0\n1 1 3 0\n1 1 4 0\n1 1 5 0\n1 1
6 0\n1 1 7 0\n1 1 8 0\n1 1 9 0\n1 1 10 0\n1 1 11 0\n1 1 12 0\n1 1 13 0\n1 1 14 0\n1 1 15 0\n8\n1
2 0 0\n1 2 1 0\n1 2 2 0\n1 2 3 0\n1 2 4 0\n1 2 5 0\n1 2 6 0\n1 2 7 0\n16\n1 3 0 0\n1 3 1 0\n1
3 2 0\n1 3 3 0\n1 3 4 0\n1 3 5 0\n1 3 6 0\n1 3 7 0\n1 3 8 0\n1 3 9 0\n1 3 10 0\n1 3 11 0\n1 3
12 0\n1 3 13 0\n1 3 14 0\n1 3 15 0\n12\n1 4 0 0\n1 4 1 0\n1 4 2 0\n1 4 3 0\n1 4 4 0\n1 4 5 0\n1
4 6 0\n1 4 7 0\n1 4 8 0\n1 4 9 0\n1 4 10 0\n1 4 11 0\n4\n1 5 0 0\n1 5 1 0\n1 5 2 0\n1 5 3 0\n8\n2
0 0 0\n2 0 1 0\n2 0 2 0\n2 0 3 0\n2 0 4 0\n2 0 5 0\n2 0 6 0\n2 0 7 0\n16\n2 1 0 0\n2 1 1 0\n2
1 2 0\n2 1 3 0\n2 1 4 0\n2 1 5 0\n2 1 6 0\n2 1 7 0\n2 1 8 0\n2 1 9 0\n2 1 10 0\n2 1 11 0\n2 1
12 0\n2 1 13 0\n2 1 14 0\n2 1 15 0\n8\n2 2 0 0\n2 2 1 0\n2 2 2 0\n2 2 3 0\n2 2 4 0\n2 2 5 0\n2
2 6 0\n2 2 7 0\n16\n2 3 0 0\n2 3 1 0\n2 3 2 0\n2 3 3 0\n2 3 4 0\n2 3 5 0\n2 3 6 0\n2 3 7 0\n2
3 8 0\n2 3 9 0\n2 3 10 0\n2 3 11 0\n2 3 12 0\n2 3 13 0\n2 3 14 0\n2 3 15 0\n12\n2 4 0 0\n2 4 1
0\n2 4 2 0\n2 4 3 0\n2 4 4 0\n2 4 5 0\n2 4 6 0\n2 4 7 0\n2 4 8 0\n2 4 9 0\n2 4 10 0\n2 4 11
0\n4\n2 5 0 0\n2 5 1 0\n2 5 2 0\n2 5 3 0\n";

close (EP) || die "can't close $a: $!";
close (OF) || die "can't close $b: $!";

print "Finished: found a total of $count words\n";
if($count=64)
{
    print "UPM Table Complete\n";
}
else
{
    print "UPM Table Incomplete\n";
}

```

THIS PAGE INTENTIONALLY LEFT BLANK

How to Reach Us:

Home Page:

www.freescale.com

Web Support:

http://www.freescale.com/support

USA/Europe or Locations Not Listed:

Freescale Semiconductor, Inc.
 Technical Information Center, EL516
 2100 East Elliot Road
 Tempe, Arizona 85284
 1-800-521-6274 or
 +1-480-768-2130
 www.freescale.com/support

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)
 www.freescale.com/support

Japan:

Freescale Semiconductor Japan Ltd.
 Headquarters
 ARCO Tower 15F
 1-8-1, Shimo-Meguro, Meguro-ku
 Tokyo 153-0064
 Japan
 0120 191014 or
 +81 3 5437 9125
 support.japan@freescale.com

Asia/Pacific:

Freescale Semiconductor China Ltd.
 Exchange Building 23F
 No. 118 Jianguo Road
 Chaoyang District
 Beijing 100022
 China
 +86 10 5879 8000
 support.asia@freescale.com

For Literature Requests Only:

Freescale Semiconductor
 Literature Distribution Center
 1-800 441-2447 or
 +1-303-675-2140
 Fax: +1-303-675-2150
 LDCForFreescaleSemiconductor
 @hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale and the Freescale logo are trademarks or registered trademarks of Freescale Semiconductor, Inc. in the U.S. and other countries. All other product or service names are the property of their respective owners. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© Freescale Semiconductor, Inc., 2009. All rights reserved.

Document Number: AN2810
 Rev. 2
 10/2009

