# Course Introduction

**Purpose**

- This course introduces the PowerQUICC III $I^2C$ bus and describes some uses and features of the boot sequencer for the PowerQUICC III.

**Objectives**

- Describe the PowerQUICC III $I^2C$ bus.
- Initialize the boot sequencer.
- Identify the EEPROM format to use with the boot sequencer.
- Compare and contrast standard $I^2C$ and extended $I^2C$.
- Identify the boot sequencer generator tool.
- Explain the importance of the boot sequencer tool to PowerQUICC III, using a burstable flash example.

**Content**

- 23 pages
- 4 questions

**Learning Time**

- 35 minutes

This course introduces the PowerQUICC III $I^2C$ bus and describes some uses and features of the boot sequencer for the PowerQUICC III. Next, this course shows how to initialize the boot sequencer, what kind of EEPROM format is required to use the boot sequencer, and some of the differences between standard $I^2C$ and extended $I^2C$. Also, this course introduces the boot sequencer generator, which helps simplify data for translation into the proper EEPROM format. Finally, the course contains a general burstable flash example to demonstrate why you might want to use the boot sequencer on the PowerQUICC III.

## I²C Controller Features

- Two-wire serial, bi-directional bus
- Multi-master operational
- Arbitration lost interrupt with automatic switch from master to slave
- Calling address identification interrupt
- START and STOP signal generation and detection
- Bus busy detection
- Software programmable clock divider from 160 to 32,768 Core Complex Bus (CCB) cycles
- Software selectable acknowledge bit
- On-chip filtering of SDA and SCL inputs
- Standard I²C support
- Boot sequencer mode support
  - Standard I²C
  - Extended I²C

Let's look at the features of the I²C controller. The I²C bus is a very simple, very direct, two-wire serial, bi-directional bus. It has a data signal and a clock signal. It can operate with multi-masters, and if you lose arbitration, you will receive a lost arbitration interrupt with an automatic switch from master to slave. It also provides a calling address identification interrupt. In addition, there is a standard START and STOP signal generation and detection, and a bus busy detection.

A software programmable clock divider allows you to divide down from the Core Complex Bus (CCB) clock. You can choose a divider between 160 and 32,768. Choose whichever divider you like, and you're ready to go.

In addition, some other standard features are a software selectable acknowledge bit and on-chip filtering of the data and clock inputs. SDA corresponds to the serial data signal and the SCL is the serial clock. Standard I²C protocol is supported on the I²C block. When it is in boot sequencer mode, however, it supports both standard I²C and extended I²C mode.

## Boot Sequencer

| What is the boot sequencer and why would you use it? |
| --- |
| • It is used before boot code is loaded to configure registers in any memory mapped address.<br>• It can configure the chip to perform non-standard operations:<br>– Initialize external memories<br>– Change default register values<br>– Boot from burstable flash, DDR, L2 |
| **What are some boot sequencer mode features?** |
| • Standard $I^2C$ support<br>• Extended $I^2C$ support<br>• Alternate Configuration Space (ACS) Mapping<br>– Can be used to write to external memories<br>• Boot sequencer executes before “time zero” if enabled |

What is the boot sequencer and why would you use it? This functionality is new in PowerQUICC III. It is used before the boot code is loaded into the chip. It allows you to configure the registers in any memory mapped address. This is a powerful feature because you can change any memory mapped register before you run boot code. For example, you can configure the chip to perform any non-standard operation, such as initializing external memories, changing default register values, or configuring it to boot from burstable flash, double data rate (DDR), or L2 cache. Previously, PowerQUICC did not support these boot methods.

The boot sequencer supports the extended I2C and the standard I2C protocols. The Alternate Configuration Space (ACS) Mapping feature is used to write to external memory. It allows you to write to a register that does not reside in the chip—it is still a memory mapped register, but it resides in some external memory. If the start of boot code execution is “time zero,” then the boot sequencer executes before time zero. With the boot sequencer, you can change register information before time zero.

## Power On Sequence

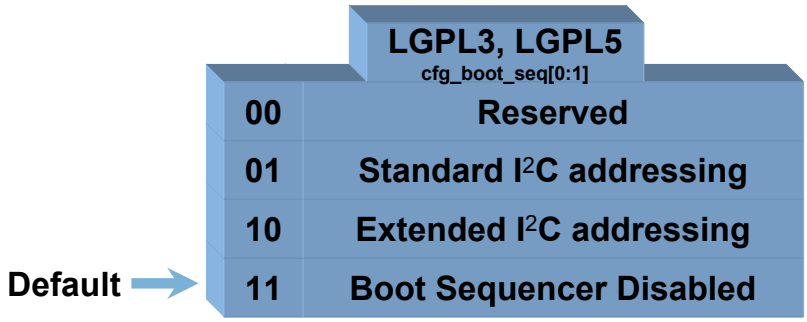
1. Power is applied to meet the voltage specifications.
2. ~HRESET is asserted and ~TRST is asserted.
3. Stable SYSCLK is applied.
4. HRESET is negated after required hold time.
5. PLLs and DLLs to e500 core and CPM are locked.
6. Boot sequencer is initialized, if enabled.
7. Wait for boot sequencer to complete, then simultaneously:
   - Enable external PCI accesses
   - Initiate RapidIO training
   - Enable e500 core to begin its boot process
   - Assert Ready Signal

Here you can see an abridged version of the power on sequence. The complete power on sequence can be found in the device-specific reference manual, which is available on the Web. First, power is applied, then you go through the regular hard reset sequence where you assert ~HRESET and ~TRST. A stable SYSCLK is applied, and then ~HRESET is negated after the required hold time.

The next step after that is to allow the phase-locked loops (PLLs) and the delay lock loops (DLLs) to the core and the CPM to lock. At this point, the boot sequencer, if you enable it, will run. Note that the boot sequencer runs before you enable external PCI accesses, before RapidIO starts training, and before the e500 core begins its boot process.

## Boot Sequencer Initialization

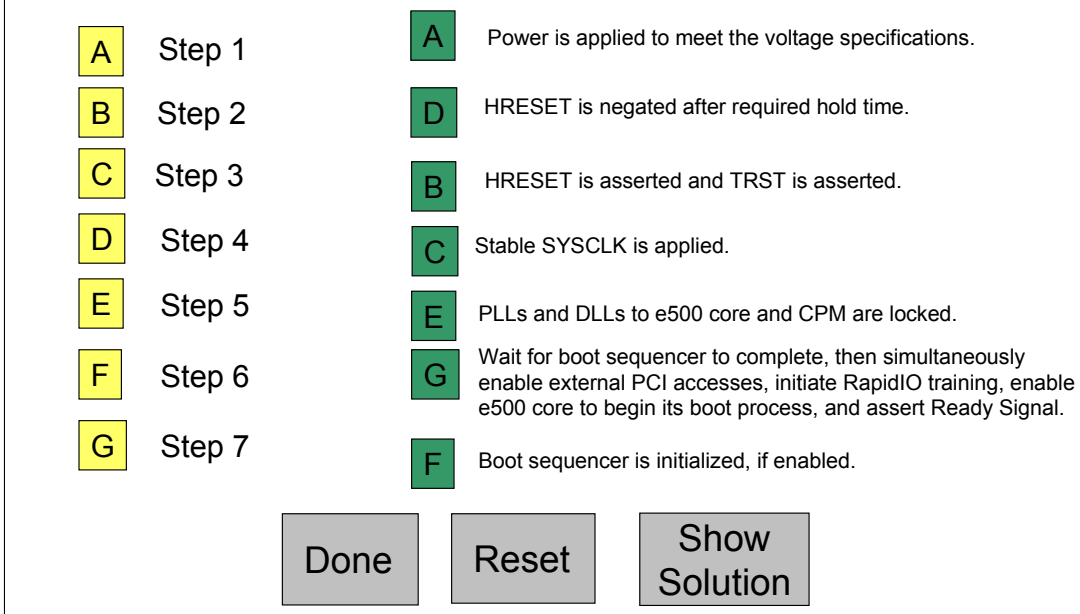
Mode determined at power-up:




You need to do some configuration on the chip before you power on. This is called power on reset configuration. The two power on reset configuration pins that control the boot sequencer are called LGPL3 and LGPL5, and together they tell PowerQUICC III whether to enable the boot sequencer in standard $I^2C$ addressing mode, enable the boot sequencer in extended $I^2C$ mode, or disable the boot sequencer. The default setting for these pins is 11, which indicates it will not use the boot sequencer at all or it will be disabled.

# Question

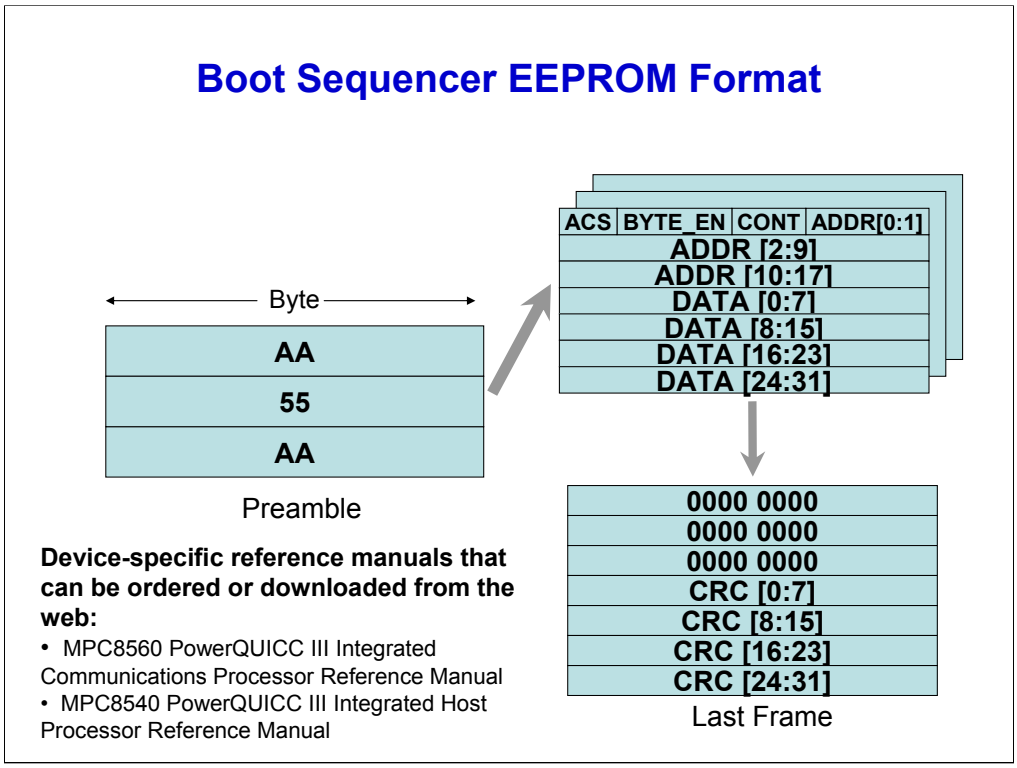
**Order the steps of the power on sequence by dragging the items on the left to their corresponding boxes on the right and then click Done.**




Let's review the power on sequence.

Correct.

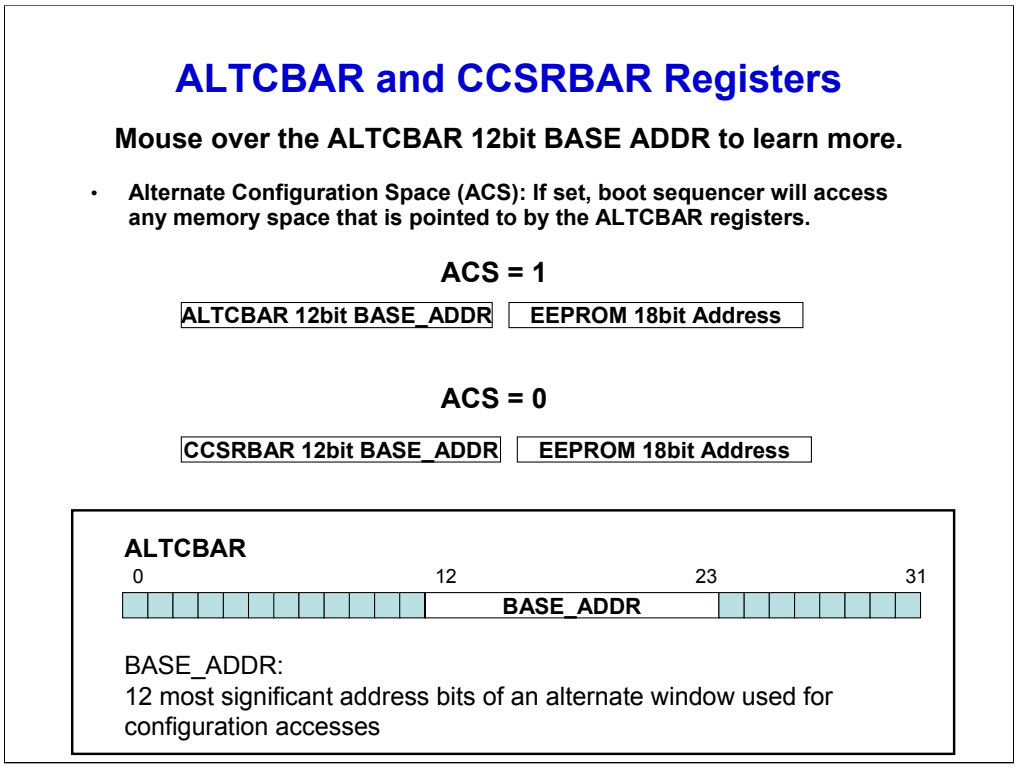
First, power is applied to meet the voltage specifications; second, HRESET and TRST are asserted. Third, stable SYSCLK is applied, and fourth, HRESET is negated after required hold time. Fifth, PLLs and DLLs to e500 core and CPM are locked, and sixth, the boot sequencer is initialized, if enabled. Finally, wait for the boot sequencer to complete, then simultaneously perform the remaining four activities.




Here you can see the format in which you need to program the EEPROM to run the boot sequencer. Three bytes make up the preamble. The first data that the boot sequencer must see are these three bytes, AA, 55, and then AA.

Next are the register inputs, and each register input has 7 bytes. So, you can think of it as a set of 7 bytes for each register. For example, if you have seven registers, then you need 49 bytes for these register “preloads.”

After you finish the last register, you're going to end the transmission and the EEPROM data with this last frame format. This last frame format is very specific, and the first three bytes must be zero. The last four bytes will be a 32-bit CRC value, which can be calculated using the polynomial found in the device-specific reference manual.




You'll notice that one of the fields in the first byte of the register format is called the ACS bit. This bit allows the boot sequencer to access a register that is not in the local memory. When enabled, it tells the boot sequencer it wants to access this register in an alternate memory space. So instead of the normal base address, which is set by a register called Configuration, Control**,** and Status Base Address Register (CCSRBAR), it uses the base address set by the Alternate Configuration Base Address Register (ALTCBAR). Roll your mouse pointer over the ALTCBAR 12bit BASE ADDR to learn more.

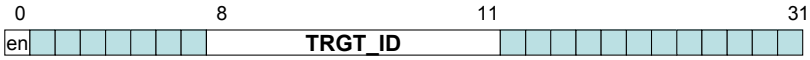
These are the first 12 bits of the base address.

The next bits are the 18-bit address. This is in the EEPROM format.

## ALTCAR Register

**Click TRGT ID to see a table showing how you can set the bits.**

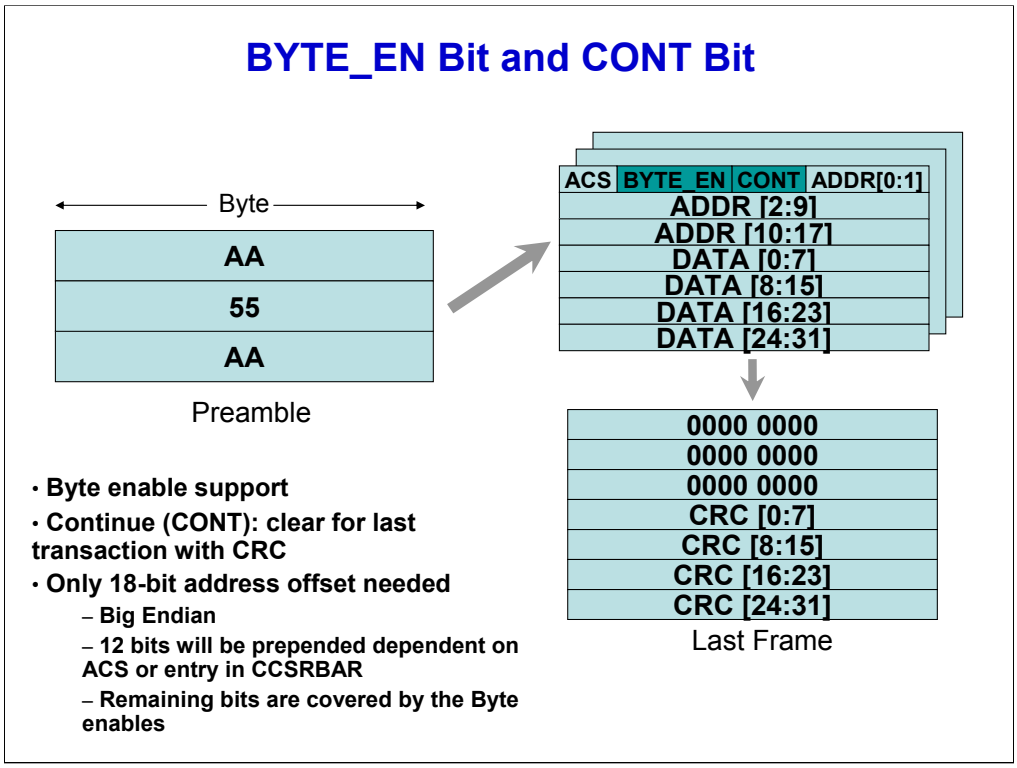
### ALTCAR




TRGT_ID: Identifies device ID to target when a transaction hits in the 1 MB address range defined

| | |
|---|---|
| **0000** | **PCI/PCI-X** |
| **0100** | **Local bus controller** |
| **1100** | **RapidIO** |
| **1111** | **Local memory – DDR SDRAM, SRAM** |

This is the Alternate Configuration Attribute Register (ALTCAR), which declares which device will be targeted when the transaction hits, whether it's a register in the PCI space, in the RapidIO, or going on the local bus. Click TRGT ID to see a table showing how you can set the bits.




Let's look at the byte enable (BYTE_EN) bit and the continue (CONT) bit. The BYTE_EN is four bits and it lets you choose the byte you want to program. If you have extra registers to program, you need to set the CONT bit to one. In the last frame, the first three bytes including the "continue" bit is zero. This will tell the boot sequencer that this is the last frame. A 32-bit CRC follows after the three bytes of zeros. A standard address is 32 bits. Only 18 bits of address are needed here because 12 bits are either coming from the CCSRBAR register or the ALTCBAR register. The remaining two bits are covered by the four bits of byte enable earlier in the packet.

## CRC-32

- CRC-32: $1+x^1+x^2+x^4+x^5+x^7+x^8+x^{10}+x^{11}+x^{12}+x^{16}+x^{22}+x^{23}+x^{26}+x^{32}$
- CRC should cover all bits stored in EEPROM prior to the CRC including:
  - Preamble
  - Register preloads
  - First 3 bytes of last frame (all 0's)
- If preamble or CRC fails, $I^2C$ asserts error signal.
  - Error triggers external hard reset request (hreset_req pin).

In the last frame format, the last 32 bits are CRC. The boot sequencer's EEPROM format uses a 32-bit CRC, and this is the polynomial that describes it. The polynomial can also be found in the reference manual. The CRC calculation should start with a value of 0xFFFF FFFF. The final XOR value is 0x0000 0000.

It's important to note that when you calculate the CRC, the CRC should cover all the bits stored in the EEPROM, including the preamble, all the register preloads, and the first three bytes of the last frame.

If for any reason the CRC does not match up with what the PowerQUICC III expects, the $I^2C$ bus will assert an error signal, and this error signal will be on the hreset_req pin. If the preamble fails right away, then it will also assert the same error signal on the hreset_req pin. At this point, the boot sequencer will halt. A hard reset is required to restart the boot sequencer.

## Question

**Is the following statement true or false? Click Done when you are finished.**

“The ACS bit allows the boot sequencer to access any kind of register not in the local memory. The normal base address is set by a register called CCSRBAR, but with ACS, the base address set by the ALTCBAR is used.”

True

False

Consider this question regarding EEPROM formatting.

Correct.

The ACS bit allows the boot sequencer to access any kind of register not in the local memory. The normal base address is set by a register called CCSRBAR, but with ACS, the base address set by the ALTCBAR is used.

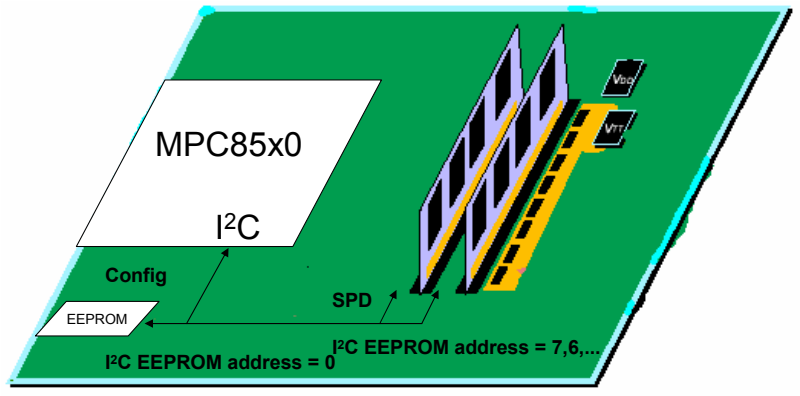
## Standard I²C

**How much space can it support?**

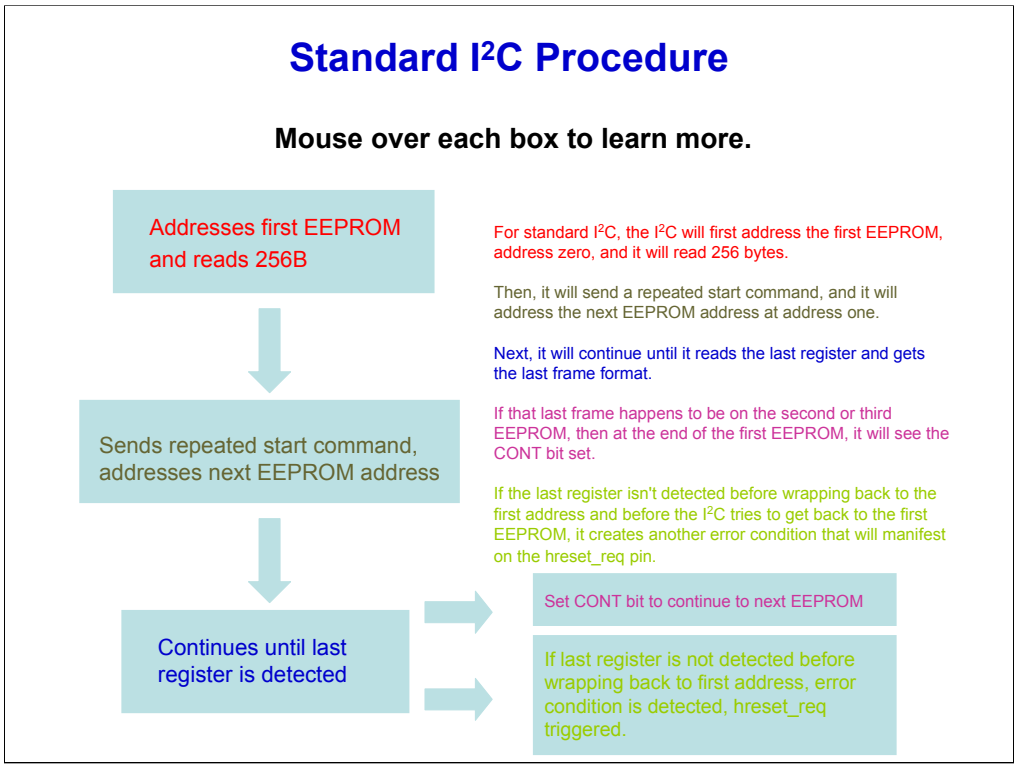
- Supports up to eight EEPROM blocks on bus
  - 3 Chip Enable bits
- Maximum EEPROM size: 2KB, 256B each block
- First EEPROM must be addressed: 0x1010 0000.
- Other EEPROMs must be addressed sequentially.

Here are some of the basic features of standard I²C bus. How much data can be put into the boot sequencer? In standard I²C mode, it supports up to eight EEPROM blocks on the bus, using three chip enable bits. The maximum EEPROM size is 2 KB, with 256 bytes for each of the eight blocks. The addressing is sequential. The first block must be addressed with 0x1010 0000, and the other EEPROMs must be addressed sequentially from that address.

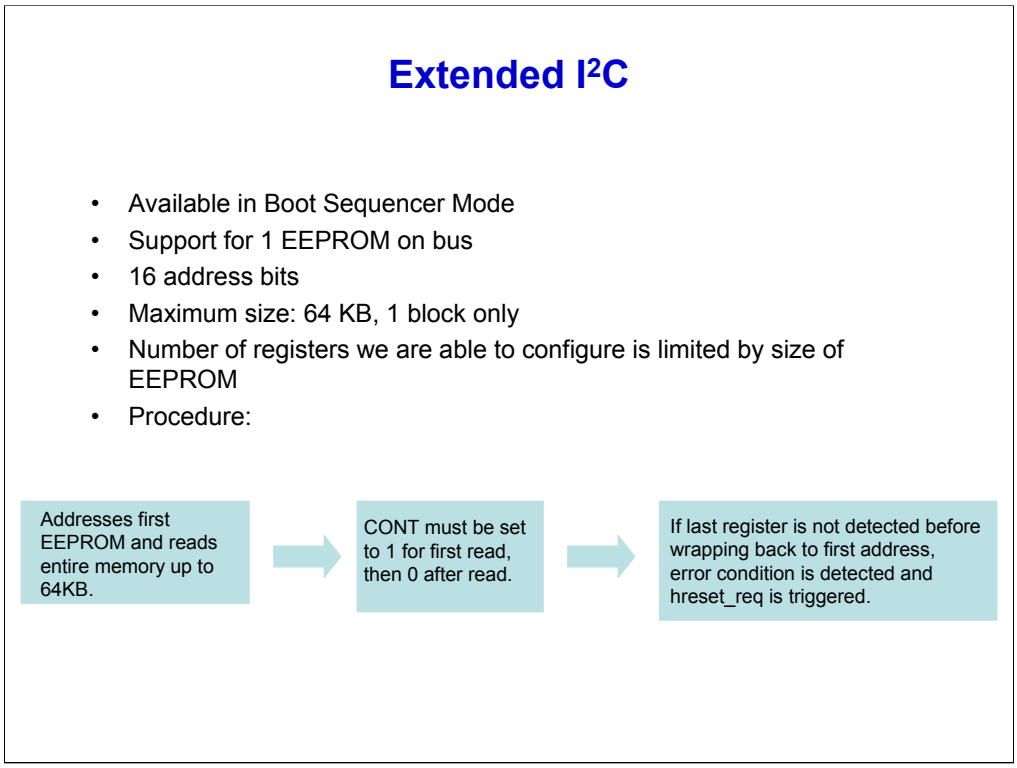
## I²C EEPROM Addressing




Here you can see an example of how the EEPROM would look on the I²C bus. EEPROM is on the left, and then on the right are the DDR DIMMs, which also have some SPD pins that hook into the I2C bus. The EEPROM address is going to start at zero, and some of the memories you have on the bus such as DDR DIMMs, may start from seven or six.




Here you can see the I$^2$C standard procedure. Roll your mouse pointer over each box in the diagram to learn more about each step in the procedure.




Here you can see the extended I$^2$C mode, which is only available on the boot sequencer; not on the regular I2C block. Extended I2C mode supports a maximum of one EEPROM and has 16 address bits. The maximum size is 64 KB in one block only. The number of registers you can configure in this mode is only limited by the EEPROM's size, up to 64 KB.

The procedure is quite similar to the standard I$^2$C mode. It starts at the first EEPROM, and it reads the entire memory up to 64 KB. The difference is it will never go on to the other EEPROM since it only supports one EEPROM on the bus. What it will read is the CONT bit, and the CONT bit is always one if you have more registers to program. At the last frame, it will see that the CONT bit will be zero. Once again, if the last register is not detected before wrapping back to the first address, an error condition will be on the hreset_req pin.

# Question

**What is one of the important differences between standard I²C and extended I²C? Select the correct response and then click Done.**

a) Extended I²C can support several EEPROMs; standard I²C can only support one EEPROM.

b) Standard I²C can support several EEPROMs; extended I²C can only support one EEPROM.

c) Extended I²C can support one EEPROM; standard I²C can support no more than three EEPROMs.

Consider this question regarding the extended I²C and standard I²C.

Correct.

Standard I²C can support several EEPROMs; extended I²C can only support one EEPROM.

## Boot Sequencer Generator

- Required information for use
  - Register to be programmed
  - ACS, data byte enables, address, data in Big Endian
- Format of input file
  - Separated by spaces

| | | | |
|---|---|---|---|
| ACS | BYTE_EN[0:3] | ADDR[0:17] | DATA[0:31] |
| ACS | BYTE_EN[0:3] | ADDR[0:17] | DATA[0:31] |
| ACS | BYTE_EN[0:3] | ADDR[0:17] | DATA[0:31] |

- Output
  - Raw data file with correct EEPROM format preamble, and CRC
  - S-record format for customer's convenience

The boot sequencer generator tool tries to make it easier to program the EEPROM in the specified format declared in the user manual and in this course. If you tell the boot sequencer generator in a simple text file the ACS bit, the data byte enables, the address of the register you want to program, and the data you want to program it with, then it can calculate all the rest of the information for you.

Here you can see the format of a sample input file. It contains the four byte enables, a space, 18 bits of address, a space, and then 32 bits of data. You will proceed to the next line until you have your last register.

Next, you run this text file through the tool, and you get two output files. The first is a raw data file with the correct EEPROM format, preamble, and a CRC, just as you see it in the data book. The boot sequencer generator will tack on the preamble, and it will also calculate the CRC. The second file will be the same except in S-record format, in case you want to program the EEPROM with an S-record file.

## Burstable Flash Example

- Supported on PowerQUICC III as a boot device through boot sequencer
- PowerQUICC III talks to flash via UPM
- Procedure
  1. Use Boot Sequencer to program UPM.
  2. Configure PowerQUICC III to boot from local bus.

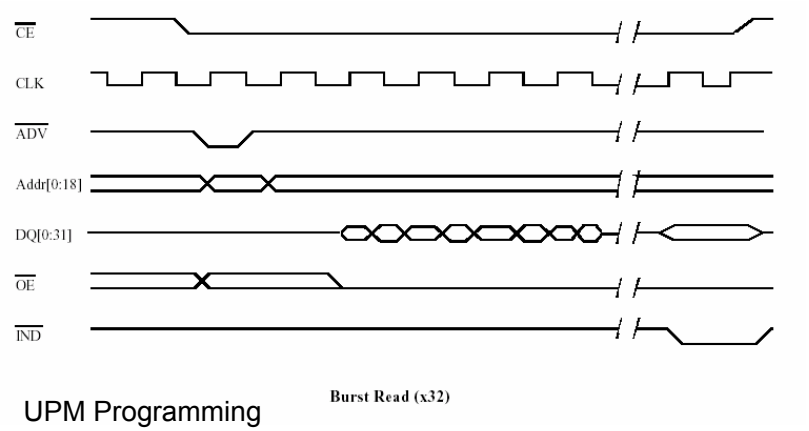
| | LCS[0:2] |
|---|---|
| 000 | PCI/PCI-X |
| 001 | DDR SDRAM |
| 010 | Reserved |
| 011 | Rapid IO |
| 000 | Reserved |
| 001 | Local bus GPCM – 8bit ROM |
| 010 | Local bus GPCM – 16bit ROM |
| 011 | Local bus GPCM – 32bit ROM |

Burstable flash is not supported on the PowerQUICC III as a boot device except through the boot sequencer. We can configure the PowerQUICC III to talk to this burstable flash through the user programmable machine (UPM). You can't use the general purpose chip-select machine (GPCM) to talk to the burstable flash because it does not support bursting capability.

Looking at the high-level procedure, you can see here that the first step is to use the boot sequencer to program the UPM. After this point, the UPM will know how to talk to the burstable flash.

Next, you configure the PowerQUICC III to boot from the local bus through the LCS0 through LCS2 pins. These pins allow you to choose from PCI, PCI-X, DDR, local bus GPCM, 8-bit, 16-bit, and 32-bit.

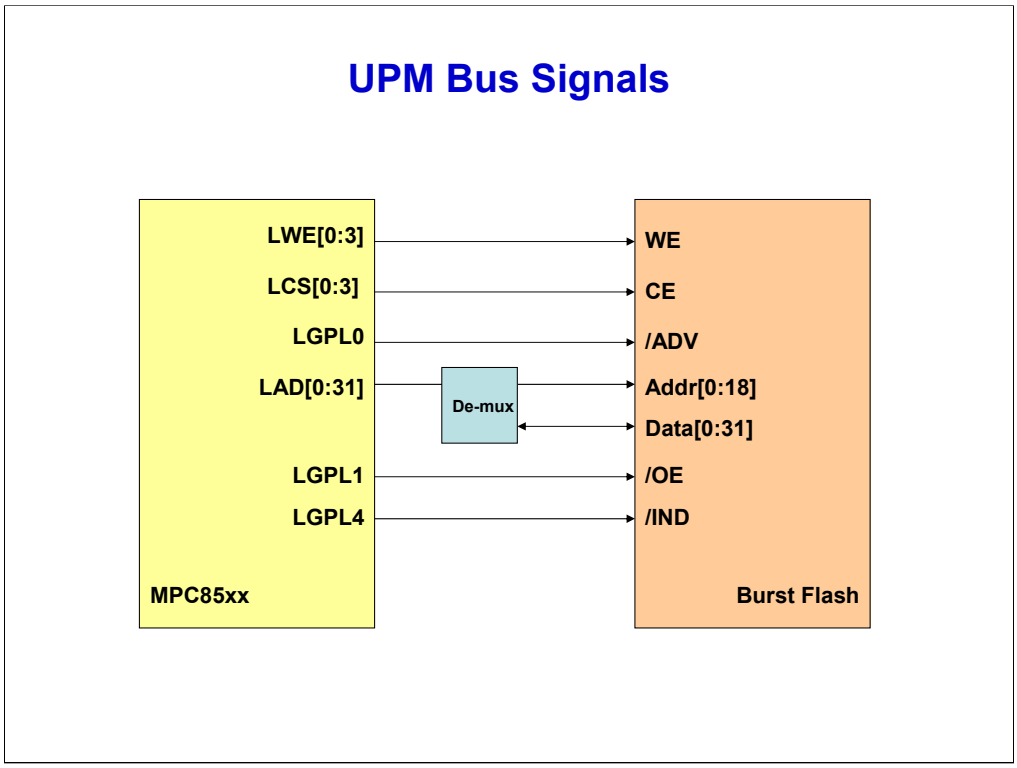
## UPM Programming




- UPM Programming
  - BR0 (addr FF70 5000)
  - OR0 (addr FF70 5004)
  - Program RAM array
    - MAMR (addr FF70 5070)
    - MDR (addr FF70 5088)

Here you can see a burstable flash example.

If we want to program the UPM to talk to the burstable flash, you need to set the BR0 register and the OR0 register. Then, you program the RAM array. Here you can see sample addresses shown. These are the addresses for the BR0 register, the OR0 register, and two others that you need to program the RAM array.




Here you can see one example of how to connect to the burstable flash. The UPM bus signals are on the left, and connected to the burstable flash on the right.

# Question

**Which of the following options describe the boot sequencer generator? Select all that apply and then click Done.**

It enables PowerQUICC II to use burstable flash.

It makes it easier to program the EEPROM in the specified format declared in the user manual and in this course.

It enables use of burstable flash on PowerQUICC III.

It generates only one file: a raw data file with the correct EEPROM format, preamble, and a CRC.

Consider this question regarding the boot sequencer generator.

Correct.

The boot sequencer generator makes it easier to program the EEPROM in the specified format declared in the reference manual and in this course. It also enables use of burstable flash on PowerQUICC III. Finally, it generates two files: a raw data file with the correct EEPROM format, preamble, and a CRC, and a second file, which is the same except in S-record format.

## Course Summary

Boot sequencer:

- Runs before boot code
- Used to change default register values
- Supports Standard or Extended $I^2C$

Boot sequencer generator:

- Produces S-record format and raw EEPROM format files
- Allows PowerQUICC III to support burstable flash

This course introduced the PowerQUICC III $I^2C$ bus and described some uses and features of the boot sequence for the PowerQUICC III. You can run the boot sequencer before boot code is run to change the values of default registers. Next, this course described the power on sequence and how to initialize the boot sequencer, as well as the EEPROM format and some of the differences between standard $I^2C$ and extended $I^2C$. Finally, this course demonstrated how the boot sequence generator, which helps simplify data for translation into the proper EEPROM format, can be used advantageously to facilitate PowerQUICC III's use of burstable flash.