Interfacing DDR3/DDR2/LPDDR Memory to ADSP-SC58x/2158x Processors

Contributed by Balaji Kannan, Mitesh Moonat, Gaurav Singh and Sachin-V Kumar

Rev 3 – April 3, 2018

Introduction

The ADSP-SC5xx/215xx SHARC® family of processors (hereafter referred to collectively as “ADSP-SC5xx processors”) incorporate a Dynamic Memory Controller (DMC), which provides a glueless interface between off-chip DDR3/DDR2/LPDDR (hereafter referred to as “DDR”) memory devices and the rest of the processor infrastructure. The block diagram in Figure 1 shows at a high level how the DMC module connects various system bus masters (cores, peripheral DMAs, MDMAs, hardware accelerators, and so forth.) to the DDR memory device through the system fabric. For further technical details on the DMC module, refer to the ADSP-SC58x/ADSP-2158x SHARC+® Processor Hardware Reference Manual[2], ADSP-SC57x SHARC+® Processor Hardware Reference Manual[13], ADSP-SC582/SC583/SC584/SC587/SC589/ADSP-21583/21584/21587 SHARC+ Dual-Core DSP with ARM Cortex-A5 Data Sheet[11] and ADSP-SC570/SC571/SC572/SC573/ADSP-21571/21573 SHARC+ Dual-Core DSP with ARM Cortex-A5 Data Sheet[14].

Not all of the ADSP-SC5xx/ADSP-215xx processors support two DMC blocks. For more details on the exact number of DMC blocks supported on a particular part, refer to the corresponding data sheet.

Figure 1. DMC Interface Block Diagram
This EE-note discusses some of the important software and hardware guidelines that must be followed when interfacing ADSP-SC5xx processors with DDR memory devices. The associated zip file includes code examples that can be used for basic DMC initialization and for a few specific application scenarios. The code examples also include a subroutine which can be used to validate the DMC interface for different types of accesses (e.g., core, DMA, 8-/16-/32-/64-bit) and different data patterns (for example, all 0x0, all 0xF, all 0x5, all 0xA, incremental, random, and all bits toggling).

**DDR/LPDDR/DDR2/DDR3 Feature Comparison**

Since ADSP-SC5xx processors support three DDR standards (LPDDR, DDR2, and DDR3), it is important to understand the major differences among them (Table 1). Even though DDR itself is not supported on ADSP-SC5xx processors, it is worthwhile to include in the overall comparison. Along with these details, the features not supported on ADSP-SC5xx processors for DDR are shown in red.

<table>
<thead>
<tr>
<th>Item</th>
<th>DDR</th>
<th>LPDDR</th>
<th>DDR2</th>
<th>DDR3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data rate/pin</td>
<td>200/266/333/400 MT/s (100/133/166/200 MHz)</td>
<td>200/266/333/360/400 MT/s (100/133/166/180/200 MHz)</td>
<td>400/533/667/800 MT/s (200/266/333/400 MHz)</td>
<td>800/1066/1333/1600 MT/s (400/533/667/800 MHz)</td>
</tr>
<tr>
<td>Clock frequency</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power supply (VDD/VDDQ)</td>
<td>2.5 V</td>
<td>1.8 V</td>
<td>1.8 V</td>
<td>1.5 V</td>
</tr>
<tr>
<td>Interface</td>
<td>SSTL_2</td>
<td>LV-CMOS</td>
<td>SSTL_18</td>
<td>SSTL_15</td>
</tr>
<tr>
<td>Number of banks</td>
<td>4</td>
<td>4</td>
<td>4 or 8</td>
<td>8</td>
</tr>
<tr>
<td>Prefetch</td>
<td>2 bits</td>
<td>2 bits</td>
<td>4 bits</td>
<td>8 bits</td>
</tr>
<tr>
<td>Burst length</td>
<td>2/4/8</td>
<td>2, 4 or 8 (16 is optional)</td>
<td>4/8</td>
<td>4 (burst chop)/8</td>
</tr>
<tr>
<td>Posted CAS, Additive latency</td>
<td>No</td>
<td>No</td>
<td>Yes (AL = 0/1/2/3/4/5)</td>
<td>Yes (AL = 0/CL - 1/CL - 2)</td>
</tr>
<tr>
<td>RL, WL</td>
<td>RL = CL (no AL), WL = 1</td>
<td>RL = CL (no AL), WL = 1</td>
<td>RL = AL + CL, WL = RL - 1 = AL + CL - 1</td>
<td>RL = AL + CL, WL = AL + CWL</td>
</tr>
<tr>
<td>ZQ pin</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>Available. For ZQ calibration*2</td>
</tr>
<tr>
<td>/Reset pin</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>Available*4</td>
</tr>
<tr>
<td>Driver impedance (Ron)</td>
<td>Programmable</td>
<td>Programmable</td>
<td>Programmable</td>
<td>Programmable</td>
</tr>
<tr>
<td>Driver impedance calibration</td>
<td>NA</td>
<td>NA</td>
<td>For OCD calibration*3, 6</td>
<td>For ZQ calibration*2</td>
</tr>
<tr>
<td>ODT function</td>
<td>NA</td>
<td>NA</td>
<td>Available</td>
<td>Available</td>
</tr>
<tr>
<td>ODT calibration</td>
<td>NA</td>
<td>NA</td>
<td>NA*6</td>
<td>For ZQ calibration*2</td>
</tr>
<tr>
<td>CLK-DQS de-skew mechanism</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>Available (Write leveling, Read leveling) *5</td>
</tr>
<tr>
<td>Package</td>
<td>TSOP II</td>
<td>FBGA</td>
<td>FBGA</td>
<td>FBGA</td>
</tr>
</tbody>
</table>

Table 1. DDR/LPDDR/DDR2/DDR3 Feature Comparison

1. With clock frequency of 450 MHz, ADSP-SC5xx processors support data rate only up to 900 MT/s (Mega-Transfers/second).
2. ZQ Calibration (memory end): Calibrates DRAM ODT and Ron fluctuations with PVT (process, voltage, and temperature). An external resistor (240Ω±1%) is inserted between the DRAM ZQ pin and GND for reference. To perform ZQ calibration, the ZQCL or ZQCS command is used. (This is a self-calibration in which DDR3 performs all the measurements and adjustments automatically.)
3. OCD (Off Chip Driver Calibration): Calibrates DRAM Ron fluctuation with PVT. The external device connected to DRAM performs impedance measurement and adjustment (not self-calibration). OCD is an optional feature in DDR2. This feature is not supported on ADSP-SC5xx processors.

4. /RESET pin is introduced in DDR3 for system stability. /RESET is an active-low signal.

5. DDR3 DIMM uses fly-by topology for CMD/ADD/CLK signals to improve signal quality, causing flight time differences between DQ/DM/DQS and CMD/ADD/CLK. DDR3 de-skew mechanism to compensate for this is not supported on ADSP-SC5xx processors, which supports only a single DRAM chip.


Software Considerations – DMC Programming Model

Figure 2 shows the programming steps for initializing the DMC module for various DDR modes.

As can be seen, DMC initialization consists of three major steps:

1. Clock Generation Unit (CGU) Initialization
2. DMC PHY Initialization
3. DMC Controller Initialization

Figure 2. DMC Programming Model Flow Chart
**CGU Initialization**

The first step of DMC initialization is to verify that the DDR Clock (DCLK) is set to the required frequency. On ADSP-SC5xx processors, DCLK clocks both DMC0 and DMC1; however, DCLK can come from either CGU0 (default) or CGU1 by programming the Clock Distribution Unit (CDU). Routing DCLK from CGU1 is useful for cases where the required DCLK frequency is asynchronous to the CCLK and SYSCLK frequencies. For example, assume a case where the required CCLK frequency is 450 MHz, the SYSCLK frequency is 225 MHz, and the DCLK frequency is 400 MHz. Achieving this frequency combination may not be possible with a single CGU. One way to realize this configuration is to generate CCLK and SYSCLK using CGU0 and DCLK using CGU1. For more details on how to program the CGU and CDU, refer to the hardware reference manual\[2\].

Consider the following points:

- Ensure that the DCLK frequency does not violate the unique maximum and minimum JEDEC-specified frequencies for all three DDR modes (DDR3/DDR2/LPDDR). For example, the minimum DCLK frequency for DDR2 is 125 MHz.
- Additional latencies can be added by the fabric for an asynchronous SYSCLK::DCLK ratio.
- The SYSCLK to DCLK clock domain crossing may need to be reprogrammed based on the SYSCLK::DCLK ratio. Refer to the System Crossbars (SCB) chapter of the hardware reference manual\[2\] for more details.

**DMC Initialization after System Reset**

After reset, the DCLK generated from CGU0 could be set to the default frequency, which will require that the CGU be reinitialized to set the DCLK to the required new frequency. As shown in Figure 2, complete the following steps when initializing the CGU to generate the required DCLK for the first time after reset:

1. Set the `DMCx_PHY_CTL0.RESETDLL` bit.
2. Initialize the CGU to change the DCLK frequency.
3. Clear the `DMCx_PHY_CTL0.RESETDLL` bit.
4. Wait 9000 DCLK cycles for the DLL to lock. Polling status bits is not necessary, a simple delay loop suffices.

Typically, the CGU is first initialized in either “preload code” (when the application is loaded through the emulator) or by “initialization code” (when the application is loaded by the boot process, in the init block), which may need to be modified to meet system requirements. Refer to the Modifying Default Preload and Initialization Code for Customized CGU/DMC Settings section for more details.

**On-the-Fly DMC Re-Initialization**

If the DCLK frequency is not being changed as part of the re-initialization process, no CGU re-initialization is necessary.

If the DCLK frequency is being changed as part of the re-initialization process, but the DDR content does not need be preserved, use the same steps as described in DMC Initialization after System Reset to re-initialize the CGU.
However, if the DCLK frequency is being changed and code or data already resident in DDR memory must be preserved, follow these steps when re-initializing the CGU:

1. If anomaly 20-00-0067 applies, ensure that bit 22 of the $DMCx\_PHY\_CTL1$ register is set. See the anomaly lists\[5\][15] for details.
2. Ensure that the DMC is in the idle state by waiting for the $DMCx\_STAT.IDLE$ bit to be set.
3. Place the DMC into self-refresh mode by setting the $DMCx\_CTL.SRREQ$ bit.
4. Poll for the setting of the $DMCx\_STAT.SRREQ$ bit to wait for the self-refresh mode transition to complete.
5. Set the $DMCx\_PHY\_CTL0\.RESETDLL$ bit.
6. Initialize the CGU to change the DCLK frequency.
7. Clear the $DMCx\_PHY\_CTL0\.RESETDLL$ bit.
8. Wait 9000 DCLK cycles for the DLL to lock.
9. Bring the DMC out of self-refresh mode by clearing the $DMCx\_CTL.SRREQ$ bit.
10. Poll for the clearing of the $DMCx\_STAT.SRREQ$ bit to wait for the self-refresh exit to complete.

When re-initializing the DMC, make sure that the CGU/DMC initialization code is not executed from the DDR memory corresponding to the same DMC. For more details, refer to Question 2 in the Frequently Asked Questions section of this EE-note.

**DMC PHY Initialization**

The DMC PHY provides the interface between the DMC controller and the DMC pads. There are a few registers in the DMC PHY which need to be programmed based on the DDR mode. As shown in Figure 2, the PHY programming includes the following steps:

1. Set the appropriate DDR mode (DDR3/DDR2/LPDDR) in the $DMCx\_PHY\_CTL4$ register.
2. Make sure that bits 6, 7, 25, and 27 of the $DMC\_PHY\_CTL3$ register are set.
3. For DDR2/DDR3 mode, verify that the lower four bits of the $DMC\_PHY\_CTL0$ register ($DMCx\_PHY\_CTL0=0x0000000F$) and the upper six bits of the $DMC\_PHY\_CTL2$ register ($DMCx\_PHY\_CTL2=0xFC000000$) are set.
4. For DDR3 mode, write the $DMC\_CPHY\_CTL$ register with bit 1 set and the $DMC\_CPHY\_CTL[5:2]$ bits configured such that WL = CWL + AL (in DCLK cycles).
5. Configure the Driver Impedance and On-Die Termination (ODT), as discussed in the Driver Impedance and On-Die Termination (ODT) Configuration section of this EE-Note.

DMC PHY programming typically needs to be done only once after power-up or reset, unless driver impedance and/or ODT recalibration is required.

**Driver Impedance and On-Die Termination (ODT) Configuration**

Driver impedance (strength) and on-die termination (ODT) apply to both the memory side and the processor side of the interface, each with its own considerations.
**Memory Side Driver Impedance and ODT**

Driver impedance is programmable for all DDR modes:

- **LPDDR**: DMC_EMR
- **DDR2**: DMC_EMR1
- **DDR3**: DMC_MR1

However, drive impedance calibration is supported only for DDR3 mode.
ODT is programmable only for DDR2 and DDR3 modes:
- DDR2: DMC_EMRI
- DDR3: DMC_MR1

However, ODT calibration is supported only for DDR3 mode. For details, refer to the corresponding DDR memory device data sheet.

Both the driver impedance and ODT for the memory can be programmed as part of the DMC controller initialization by programming the corresponding mode registers accordingly.

In DDR3 mode, during DMC initialization, the DDR3 controller sends a ZQ Cal Long command (coarse calibration) to the memory device. After every self-refresh exit command, the controller also issues a ZQ Cal Long command to the device. A ZQ Cal Short command for fine calibration must be issued by the application (for example, a GP timer) in software. For more details, refer to the hardware reference manual [2].

**Processor Side Driver Impedance and ODT**

Driver impedance is programmable only for the DDR2 and DDR3 modes, and driver impedance calibration is supported for both. The DMC pads can be auto-calibrated to the required driver impedance using an external resistance DMC_RZQ by programming the DMCx_CAL_PADCTL2 register:

- The DMCx_CAL_PADCTL2.IMPWRAD bit field for the address (DMC_A[nn], DMC_BA[n]) and command (DMC_CKE, DMC_CS[n], DMC_ODT, DMC_RAS, DMC_RESET, DMC_WE) pads configure the driver impedance. The recommended value to be programmed is 60 Ω (0x3C) for both DDR2 and DDR3.

- The DMCx_CAL_PADCTL2.IMPWRDQ bit field for the data (DMC_DQ[nn]), DQS (DMC_LDQS, /DMC_LDQS, DMC_UDQS, /DMC_UDQS), clock (DMC_CK, /DMC_CK), and DM (DMC_U DM, DMC_LDM) pads configure the driver impedance. The recommended value to be programmed is 40 Ω (0x28) for both DDR2 and DDR3.

ODT is programmable only for the DDR2 and DDR3 modes, and ODT calibration is supported for both. The DMC PHY can be programmed to auto-calibrate the ODT value (parallel termination) on the data and DQS pads by programming the DMCx_CAL_PADCTL2.IMPR TT bit field to 1.6 times the required ODT value in Ohms. For example, for 75 Ω termination, program the DMCx_CAL_PADCTL2.IMPR TT bit field to 1.6 * 75 = 120 (0x78). Set the DMCx_PHY_CTL1.BYPODTEN bit to bypass ODT programming.

Complete the following steps to configure ODT and calibrate driver impedance on the ADSP-SC5xx processor:

1. Configure ODT and drive impedance values in the DMCx_CAL_PADCTL2 register, as discussed above. Ensure that the DMCx_CAL_PADCTL0 register bits RTTCALEN, PDCALEN, and PUCALEN are set (they are set at reset).

2. Set the DMCx_CAL_PADCTL0.CALSTART bit to start the ODT and drive impedance calibration.

3. Wait for 300 DCLK cycles for the pad driver impedance and ODT calibration to complete. Polling status bits is unnecessary, as a simple delay loop suffices.
**DMC Controller Initialization**

The final step for the DMC initialization is the DMC controller initialization. The controller has a set of registers which has various bit fields that are categorized as follows:

- Hard-Wired Settings
- Mandatory Settings
- Optional Settings

**Hard-Wired Settings**

There are a few bits which are hard-coded in the DDR controller that software cannot adjust. Table 2 through Table 7 shade these bit fields in **ORANGE**.

**Mandatory Settings**

Many bits in the configuration, timing, and mode registers must be programmed based on the system to ensure proper DMC operation in the application. Table 2 through Table 7 shade these bit fields in **GREEN**. For more details on how to program these bit fields, refer to the hardware reference manual [2].

**Optional Settings**

There are a few bit fields which are not required to be modified for standard DMC operation; however, deeper knowledge of these bits saves power and improves throughput in certain application configurations. For example, the DMC_CTL.SRREQ bit can be used to operate the DMC in a low-power (self-refresh) mode, and the DMC_EMR2.PASR bit (in LPDDR only) may help for intelligent power savings. The DMC_CTL.PREC bit enables automatic precharge after each access, and the DMC_CTL.ADDRMODE bit improves throughput by switching between page and bank interleaving addressing modes. Users are expected to understand the functionality of these bits clearly by going through the hardware reference manual[2] and the corresponding memory device data sheet (especially for mode registers). These bits are shaded in **RED** in Table 2 through Table 7.

Referring back to Figure 2, complete the following steps to initialize the DMC controller:

1. Program the DMC configuration register (DMCx_CFG), as described in Table 2.

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit Field</th>
<th>Bit field (C=Controller, M=JEDEC)</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMC_CFG</td>
<td>IFWID</td>
<td>Interface Width 3:0-C</td>
<td></td>
<td>Always program to 2 (16-bit). All other values are reserved.</td>
</tr>
<tr>
<td></td>
<td>SDRWID</td>
<td>SDRAM Width 7:4-C</td>
<td></td>
<td>Always program to 2 (16-bit). All other values are reserved.</td>
</tr>
<tr>
<td></td>
<td>SDRSIZE</td>
<td>SDRAM Size 11:8-C</td>
<td></td>
<td>Obtain from memory device data sheet.</td>
</tr>
<tr>
<td></td>
<td>EXTBANK</td>
<td>External Banks 15:12-C</td>
<td></td>
<td>Always program to zero (16-bit). All other values are reserved.</td>
</tr>
<tr>
<td></td>
<td>Reserved</td>
<td>Reserved 31:16-C</td>
<td></td>
<td>Always write these bits with zero.</td>
</tr>
</tbody>
</table>

*Table 2. DMC Controller Configuration Register Bit Fields*
2. Program the DMC timing registers (DMCx_TR0, DMCx_TR1, and DMCx_TR2), as described in Table 3.

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit Field</th>
<th>Bit field (C=Controller, M=JEDEC)</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMC_TR0</td>
<td>TRCD</td>
<td>RAS# to CAS# delay time</td>
<td>3:0-C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TWTR</td>
<td>Write-to-Read delay</td>
<td>7:4-C</td>
<td>Obtain from memory device data sheet.</td>
</tr>
<tr>
<td></td>
<td>TRP</td>
<td>Precharge-to-Active time</td>
<td>11:8-C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TRAS</td>
<td>Active-to-Precharge time</td>
<td>16:12-C</td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved</td>
<td></td>
<td>19:17-C</td>
<td>Always write these bits with zero.</td>
</tr>
<tr>
<td>TRC</td>
<td>Active-to-Active time</td>
<td>25:20-C</td>
<td>Obtain from memory device data sheet.</td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved</td>
<td></td>
<td>27:26-C</td>
<td>Always write these bits with zero.</td>
</tr>
<tr>
<td>TMRD</td>
<td>Mode register set-to-active</td>
<td>31:28-C</td>
<td>Obtain from memory device data sheet.</td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved</td>
<td></td>
<td>15:14-C</td>
<td>Always write these bits with zero.</td>
</tr>
<tr>
<td>TRFC</td>
<td>Refresh-to-Active command delay</td>
<td>23:16-C</td>
<td>Obtain from memory device data sheet.</td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved</td>
<td></td>
<td>27:24-C</td>
<td>Always write these bits with zero.</td>
</tr>
<tr>
<td>TRRD</td>
<td>Active-to-Active time</td>
<td>30:28-C</td>
<td>Obtain from memory device data sheet.</td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved</td>
<td></td>
<td>31</td>
<td>Always write this bit with zero.</td>
</tr>
<tr>
<td>TFAW</td>
<td>Four Activate Window</td>
<td>4:0-C</td>
<td>Obtain from memory device data sheet. tFAW is not applicable for LPDDR mode and should be kept zero.</td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved</td>
<td></td>
<td>7:5-C</td>
<td>Always write these bits with zero.</td>
</tr>
<tr>
<td>TRTP</td>
<td>Internal Read to Precharge time</td>
<td>11:8-C</td>
<td>Obtain from memory device data sheet. tRTP is not applicable for LPDDR mode and should be kept zero.</td>
<td></td>
</tr>
<tr>
<td>TWR (LPDDR only)</td>
<td>Write recovery time</td>
<td>15:12-C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TXP</td>
<td>Exit power down to next valid command</td>
<td>19:16-C</td>
<td>Obtain from memory device data sheet.</td>
<td></td>
</tr>
<tr>
<td>tCKE</td>
<td>CKE min pulse width</td>
<td>23:20-C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved</td>
<td></td>
<td>31:24-C</td>
<td>Always write these bits with zero.</td>
</tr>
</tbody>
</table>

Table 3. DMC Controller Timing Register Bit Fields
3. Program the DMC controller mode registers, depending on the mode being used:

   a. DDR2: Program the DMCx_MR, DMCx_EMR1, and DMCx_EMR2 registers (Table 4).

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit Field</th>
<th>Bit field (C=Controller, M=JEDEC)</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BT</td>
<td>Burst Type</td>
<td>3-C, A3-M</td>
<td>Only sequential burst type is supported. Keep this bit zero.</td>
</tr>
<tr>
<td></td>
<td>CL</td>
<td>CAS Latency</td>
<td>6:4-C, A6:A4-M</td>
<td>Program these bits with the required CAS latency.</td>
</tr>
<tr>
<td></td>
<td>TM</td>
<td>Test Mode</td>
<td>7-C, A7-M</td>
<td>Always program this bit to zero, as test mode is not supported.</td>
</tr>
<tr>
<td></td>
<td>DLLRST</td>
<td>DLL Reset</td>
<td>8-C, A8-M</td>
<td>Clear this bit for DDR2 mode.</td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved</td>
<td>15:13-C, A15:A13-M</td>
<td>Hard Wired</td>
<td>These bits are hard-wired to zero.</td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved</td>
<td>31:16-C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DMC_EMR1</td>
<td>DLEN</td>
<td>DLL Enable</td>
<td>0-C, A0-M</td>
<td>Keep this bit set to zero.</td>
</tr>
<tr>
<td></td>
<td>DIC</td>
<td>Output Driver Impedance Control</td>
<td>1-C, A1-M</td>
<td>Select the driver impedance using these bits from the memory side.</td>
</tr>
<tr>
<td></td>
<td>RTT</td>
<td>On Die Termination (ODT)</td>
<td>6,2-C, A6,A2-M</td>
<td>Select ODT value using these bits from the memory side.</td>
</tr>
<tr>
<td></td>
<td>AL</td>
<td>Additive Latency</td>
<td>5:3-C, A5,A3-M</td>
<td>Mandatory Can be cleared for basic DMC initialization. Refer to the memory device data sheet for more details on this bit.</td>
</tr>
<tr>
<td></td>
<td>DQS#</td>
<td>DQS# Enable</td>
<td>10-C, A10-M</td>
<td>Keep this bit set to zero.</td>
</tr>
<tr>
<td></td>
<td>RDQS</td>
<td>RDQS Enable</td>
<td>11-C, A11-M</td>
<td>RDQS is not supported. Keep this bit set to zero.</td>
</tr>
<tr>
<td></td>
<td>QOFF</td>
<td>Output Buffer Enable</td>
<td>12-C, A12-M</td>
<td>Keep this bit set to zero.</td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved</td>
<td>15:13-C, A15:A13-M</td>
<td>Hard Wired</td>
<td>These bits are hard-wired to zero.</td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved</td>
<td>31:16-C</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DCC</td>
<td>DCC Enable</td>
<td>3-C, A3-M</td>
<td>Mandatory DCC is not supported. Keep this bit set to zero.</td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved</td>
<td>6:4-C, A6:A4-M</td>
<td>Mandatory</td>
<td>Program these bits to zero.</td>
</tr>
<tr>
<td></td>
<td>SRF</td>
<td>High Temperature Self Refresh Rate Enable</td>
<td>7-C, A7-M</td>
<td>Optional These bits are unchanged for standard DMC operation. Refer to the hardware reference manual[2] for more details.</td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved</td>
<td>15:8-C, A15:A8-M</td>
<td>Hard Wired</td>
<td>These bits are hard-wired to zero.</td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved</td>
<td>31:16-C</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4. DMC Controller DDR2 Mode Register Bit Fields
b. DDR3: Program the **DMCx_MR0**, **DMCx_MR1**, and **DMCx_MR2** registers (**Table 5**).

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit Field</th>
<th>Bit field (C=Controller, M=JEDEC)</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DMC_MR0</strong></td>
<td>BL</td>
<td>Burst Length</td>
<td>1:0-C, A1:A0-M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CL</td>
<td>CAS Latency</td>
<td>6:4-C, A6:A4, A2-M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BT</td>
<td>Burst Type</td>
<td>3-C, A3-M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TM</td>
<td>Test Mode</td>
<td>7-C, A7-M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DLRST</td>
<td>DLL Reset</td>
<td>8-C, A8-M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WRRECOV</td>
<td>Write recovery</td>
<td>11:9-C, A11:A9-M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PD</td>
<td>Active Power Down Mode</td>
<td>12-C, A12-M</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Reserved</td>
<td>Reserved</td>
<td>15:13-C, A15:A13-M</td>
<td>Hard Wired</td>
</tr>
<tr>
<td></td>
<td>Reserved</td>
<td>Reserved</td>
<td>31:16-C</td>
<td></td>
</tr>
<tr>
<td><strong>DMC_MR1</strong></td>
<td>DLLEN</td>
<td>DLL Enable</td>
<td>0-C, A0-M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DIC0, DIC1</td>
<td>Output Driver Impedance Control</td>
<td>5,1-C, A5,A1-M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RT T0, RT T1, RT T2</td>
<td>On Die Termination (ODT)</td>
<td>9,6,2-C, A9,A6,A2-M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AL</td>
<td>Additive Latency</td>
<td>4,3-C, A4,A3-M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WL</td>
<td>Write Levelization</td>
<td>7-C, A7-M</td>
<td>Mandatory</td>
</tr>
<tr>
<td></td>
<td>Reserved</td>
<td>Reserved</td>
<td>8, 10-C, A8, A10 –M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TDQS</td>
<td>Termination Data Strobe</td>
<td>11-C, A11-M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QOFF</td>
<td>Output Buffer Enable</td>
<td>12-C, A12-M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reserved</td>
<td>Reserved</td>
<td>15:13-C, A15:A13-M</td>
<td>Hard Wired</td>
</tr>
<tr>
<td></td>
<td>Reserved</td>
<td>Reserved</td>
<td>31:16-C</td>
<td></td>
</tr>
<tr>
<td><strong>DMC_MR2</strong></td>
<td>PASR</td>
<td>Partial Array Self Refresh</td>
<td>2:0-C, A2:A0-M</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>CWL</td>
<td>CAS Write Latency</td>
<td>5:3-C, A5:A3-M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ASR</td>
<td>Auto Self Refresh</td>
<td>6-C, A6-M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SRT</td>
<td>Self-Refresh Temperature Range</td>
<td>7-C, A7-M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reserved</td>
<td>Reserved</td>
<td>8-C, A8-M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RTTWR</td>
<td>Dynamic ODT Write</td>
<td>10:9-C, A10:A9-M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reserved</td>
<td>Reserved</td>
<td>15:11-C, A15:A11-M</td>
<td>Hard Wired</td>
</tr>
<tr>
<td></td>
<td>Reserved</td>
<td>Reserved</td>
<td>31:16-C</td>
<td></td>
</tr>
</tbody>
</table>

**Table 5. DMC Controller DDR3 Mode Register Bit Fields**
c. LPDDR: Program the DMCx_MR and DMCx_EMR registers (Table 6).

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit Field</th>
<th>Bit field (C=Controller, M=JEDEC)</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BT</td>
<td>Burst Type 3-C, A3-M</td>
<td></td>
<td>Only sequential burst type is supported. Keep this bit set to zero.</td>
</tr>
<tr>
<td></td>
<td>CL</td>
<td>CAS Latency 6:4-C, A6:A4-M</td>
<td></td>
<td>Only CL=3 is supported for LPDDR mode.</td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved</td>
<td>15:13-C, A15:A13-M</td>
<td></td>
<td>These bits are hard-wired to zero.</td>
</tr>
</tbody>
</table>

|                     | Reserved  | 15:13-C, A15:A13-M               |       | Hard Wired |

| DMC_EMR            | PASR      | Partial Array Self Refresh 2:0-C, A2:A0-M |       | Optional |
| Reserved           | Reserved  | 15:8-C, A15:A8-M                 |       | Mandatory |
| Reserved           | Reserved  | 15:8-C, A15:A8-M                 |       | Mandatory |
| Reserved           | Reserved  | 15:8-C, A15:A8-M                 |       | Mandatory |
| Reserved           | Reserved  | 15:8-C, A15:A8-M                 |       | Mandatory |

Table 6. DMC Controller LPDDR Mode Register Bit Fields

4. Make sure that the DMC_DT_CALIB_ADDR register (Table 7) is programmed to an unused DMC location (by default, it is the starting address of the DMC address range). Refer to the hardware reference manual[2] for more details.

5. Program the DMCx_CTL register (Table 7) with DMCx_CTL.INIT set to start the DMC initialization sequence.

6. Wait for the DMC initialization to complete by polling for the DMCx_STAT.INITDONE bit to set.

7. Program the DMCx_CTL.DLLCTL register (Table 7) with the value 0x948 (DATACYC = 9, DLLCALRDCNT = 72).
<table>
<thead>
<tr>
<th>Register</th>
<th>Bit Field</th>
<th>Bit field (C=Controller, M=JEDEC)</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DMC_CTL</strong></td>
<td>DDR3EN</td>
<td>Enable DDR3 mode</td>
<td>0-C</td>
<td>Set this bit for DDR3 mode, clear for DDR2 and LPDDR modes.</td>
</tr>
<tr>
<td></td>
<td>LPDDR</td>
<td>Enable LPDDR mode</td>
<td>1-C</td>
<td>Set this bit for LPDDR mode, clear for DDR3 and DDR2 modes.</td>
</tr>
<tr>
<td></td>
<td>INIT</td>
<td>Start DMC initialization</td>
<td>2-C</td>
<td>Set this bit to start the DMC initialization sequence.</td>
</tr>
<tr>
<td></td>
<td>SRREQ</td>
<td>Self-Refresh Request</td>
<td>3-C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PDREQ</td>
<td>Power Down Request</td>
<td>4-C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DPDREQ</td>
<td>Deep Power Down Request</td>
<td>5-C</td>
<td>These bits are unchanged for standard DMC operation. Refer to the hardware reference manual[2] for more details.</td>
</tr>
<tr>
<td></td>
<td>PREC</td>
<td>Precharge</td>
<td>6-C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RESET</td>
<td>Reset SDRAM</td>
<td>7-C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADDRMODE</td>
<td>Addressing Mode</td>
<td>8-C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RDTOWR</td>
<td>Read-to-Write Cycle</td>
<td>11:9-C</td>
<td>Always program this bit field with 2.</td>
</tr>
<tr>
<td></td>
<td>PPREF</td>
<td>Postpone Refresh</td>
<td>12-C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DLLCAL</td>
<td>DLL Calibration Start</td>
<td>13-C</td>
<td>These bits are unchanged for standard DMC operation. Refer to the hardware reference manual[2] for more details.</td>
</tr>
<tr>
<td></td>
<td>Reserved</td>
<td>Reserved</td>
<td>23:14-C</td>
<td>Always write these bits with zero.</td>
</tr>
<tr>
<td></td>
<td>ZQCS</td>
<td>ZQ Calibration Short</td>
<td>24-C</td>
<td>These bits are unchanged for standard DMC operation. Refer to the hardware reference manual[2] for more details.</td>
</tr>
<tr>
<td></td>
<td>ZQCL</td>
<td>ZQ Calibration Long</td>
<td>25-C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reserved</td>
<td>Reserved</td>
<td>31:26-C</td>
<td>Always write these bits with zero.</td>
</tr>
</tbody>
</table>

| **DLLCTL** | DLLCALRDCNT | DLL Calibration RD Count | 7:0-C | Program this bit field to 72. |
| | DATACYC | Data Cycles | 11:8-C | Program this field to 9. |
| | Reserved | Reserved | 31:9-C | Always write these bits with zero. |
| | **DMC_DT_CALIB_ADDR** | Data Calibration Address Register | 31:0-C | Program this value with an unused DMC address. Refer to the hardware reference manual[2] for more details. |

*Table 7. DMC Controller Control/Calibration Bit Fields*

In addition to the above steps, also ensure that the workaround to anomaly 20-00-0037 (if applicable to the processor silicon revision being used) is also implemented after the DMC initialization is complete. See the processor silicon anomaly lists [5][15] for details.

Before the DDR controller enters the initialization procedure, it automatically provides the appropriate power-up/reset/first command timing according to the JEDEC standards.
DMC Initialization Code

The ZIP file\[^6\] associated with this EE-note provides the CGU and DMC initialization subroutines along with code examples which can be used to initialize the CGU and DMC controller for any custom settings.

**CGU Initialization**

The CGUInit.c and CGUInit.h files can be used to initialize the CGU to the required clock settings. For example, the main.c file in the ADSP-SC58x/DDR3_Init_Core1 folder illustrates how to initialize the CGU to generate the following clock combination on the ADSP-SC589 EZ-Board®, which features a 25 MHz CLKin, as shown in Listing 1:

- PLLCLK = 450 MHz
- CCLK = 450 MHz
- SYCLK = 225 MHz
- DCLK = 450 MHz
- SCLK0 = SCLK1 = 225 MHz

```c
void CGUInit(void)
{
    int i;

    // PLLCLK = 450 MHz, CCLK = 450 MHz, SYCLK = 225 MHz, DCLK = 450 MHz, SCLK0 = SCLK1 = 225 MHz
    CGU0_PARAM_LIST_DEFAULT.ctl_MSEL=18;
    CGU0_PARAM_LIST_DEFAULT.ctl_DF=0;
    CGU0_PARAM_LIST_DEFAULT.div_CSEL=1;
    CGU0_PARAM_LIST_DEFAULT.div_SYSSEL=2;
    CGU0_PARAM_LIST_DEFAULT.div_S0SEL=2;
    CGU0_PARAM_LIST_DEFAULT.div_S1SEL=2;
    CGU0_PARAM_LIST_DEFAULT.div_DSEL=1;

    // Set DM CX PHY_CTL0.RESETDLL bit.
    *pREG_DMC0_PHY_CTL0=0x800;
    *pREG_DMC1_PHY_CTL0=0x800;

    // Initialize CGU to change the DCLK frequency
    CGU_Init(0, 0, &CGU0_PARAM_LIST_DEFAULT, 0);

    // Clear DM CX PHY_CTL0.RESETDLL bit
    *pREG_DMC0_PHY_CTL0=0x800;
    *pREG_DMC1_PHY_CTL0=0x800;

    // Wait for 9000 DCLK cycles for the DLL to lock
    for(i=0;i<9000*pMC_Param_List.cclk_dclk_ratio;i++);
}
```

**Listing 1. CGU Initialization**

For custom clock settings, the values of MSEL, DF, CSEL, SYSSEL, S0SEL, S1SEL, and DSEL can be changed accordingly.
**DMC Initialization**

The DMCInit.c and DMCInit.h files can be used to initialize the DMC to the required settings. For example, the main.c file in the ADSP-SC58x_DDR3_Init_Corel folder illustrates how to initialize the DMC for the Micron MT41K128M16-125 DDR3 memory device[7] with a DCLK frequency of 450 MHz, as shown in Listing 2.

```c
//Initialize the DMC parameters list
DMC_Param_list.dmc_no=0;
DMC_Param_list.ddr_mode=DDR3_MODE;
DMC_Param_list.phy_init_required=true;
DMC_Param_list.cclk_dclk_ratio=1;
DMC_Param_list.anomaly_200000037_applicable=true;
DMC_Param_list.dmc_ctl_value=DMC_CTL_VALUE;
DMC_Param_list.dmc_cfg_value=DMC_CFG_VALUE;
DMC_Param_list.dmc_tr0_value=DMC_TR0_VALUE;
DMC_Param_list.dmc_tr1_value=DMC_TR1_VALUE;
DMC_Param_list.dmc_tr2_value=DMC_TR2_VALUE;
DMC_Param_list.dmc_mr0_value=DMC_MR0_VALUE;
DMC_Param_list.dmc_mr1_value=DMC_MR1_VALUE;
DMC_Param_list.dmc_mr2_value=DMC_MR2_VALUE;
DMC_Param_list.padctl2_value=DMC_PADCTL2_VALUE;
DMC_Param_list.dmc_cphyctl1_value=DMC_CPHYCTL1_VALUE;
DMC_Param_list.dmc_data_calib_add_value=DMC0_DATA_CALIB_ADD;

//Initialize CGU - PLLCLK = 450 MHz, CCLK = 450 MHz, SYCLK = 225 MHz, DCLK = 450 MHz, SCLK0 = SCLK1 = 112.5 MHz
CGUInit();

//Initialize DMC0 now
DMCInit(DMC_Param_list);

DMC_Param_list.dmc_no=1;
DMC_Param_list.dmc_data_calib_add_value=DMC1_DATA_CALIB_ADD;

//Initialize DMC1 now
DMCInit(DMC_Param_list);
```

**Listing 2. DMC Initialization**

Though the ADSP-SC5xx processors do not support the DDR3L (1.35 V) standard, the MT41K128M16 is backward compatible with the 1.5 V standard, thus it can be used without issue. When interfacing a DDR3L device with ADSP-SC5xx processors, contact the memory vendor to make sure that it is compatible with the DDR3 standard.

For custom DMC settings, the DMC_Param_List structure must be updated according to the system requirements. The DMC register values can be derived using the DMC_Registers_List.xlsx spreadsheet in the associated ZIP file[6] by entering various DMC-specific processor and DDR memory-specific parameters (from the device data sheet), as shown in Figure 3.
Validating the DMC Interface

Once the DMC is initialized, it is important to validate it. It is recommended to check whether or not all the DMC registers have been initialized to the correct values, if there are any basic issues with the DMC hardware interface, and whether or not the DMC has indeed been correctly initialized by the software.

The DMCInit_Debug.c and DMCInit_Debug.h files in the associated ZIP file provide some assistance for performing this validation. The Print_DMC_Register_Contents() function can be used to print the DMC controller and PHY register values, and the Memory_Sweep_Test() function can be used to check whether or not both core and DMA (MDMA0) accesses to the DMC are working for different data word sizes (8-/16-/32-/64-bit and 32-byte DMA) and for different data patterns (0x0, 0xF, 0x5, 0xA, incremental, random, and all bits toggling). Figure 4 shows how the main.c file in the DDR3_Init_Core1 folder uses these functions to validate the DMC interface. The memory sweep size used in this code is 0x100000 (1 MB), which can be changed to validate the full DMC memory range (for example, 2 Gb = 256 MB).
Creating Preload and Initialization Code with Customized CGU and DMC Settings

Preload and initialization code are two concepts that are related to configuring the CGU and DMC prior to the application code running, depending on whether performing active debug via the emulator or controlling the boot stream for a stand-alone application.

**Preload Code**

When performing active debug on a target platform, an emulator is used. To make working with the board as transparent as possible for the user, the CrossCore® Embedded Studio (CCES) tools automate initialization of the CGU/DMC hardware such that applications can be built and loaded to off-chip memory for use in a debug session on the targeted board. This is done via *Preload Code*, and there are examples in the `Custom Preload Codes` folder for each of DDR2, DDR3, and LPDDR in the associated ZIP file. When the project is built, the associated executable file (DXE) is used by CCES (as shown in *Figure 5*) to initialize the CGU and DMC before loading the actual application using the emulator.

**Figure 5. Preload Code Being Used for the ADSP-SC589 EZ-Board**
Initialization Code

Unlike preload code, initialization code is actually a part of the application. It is separate from the application itself, and its DXE output is pre-pended to the application’s DXE file when CCES assembles the loader stream (LDR) that the processor parses during the boot process. This separate DXE is called the Initialization Block in the LDR file. The DXE is booted first into on-chip memory and is then executed before any attempts are made to resolve anything to the external DDR space. It is the ideal place for configuring the CGU and DMC in advance of trying to boot to DDR memory. In the Custom Init Codes folder in the associated ZIP file[^6], there are projects for DDR2, DDR3, and LPDDR. The DXE output corresponding to the project can be used as the default initialization code when generating an LDR file by pointing to the DXE in the Loader Options page of the Project Properties. See Figure 6.

![Figure 6. Initialization Code Selection in the Loader Options](image)

Typically, for applications requiring a one-time CGU and DMC initialization after reset, the preload (when loading the application via emulator) or initialization code (when booting the application standalone) should be sufficient. Thus, it is important to understand how to use and modify the default preload and initialization code for customized CGU/DMC settings.

Default Preload and Initialization Code

The CCES installation provides the DXE files produced by the default preload and initialization code for both the ADSP-SC589, ADSP-SC584 and ADSP-SC573 evaluation platforms in the directory:

<CCES Root>\Analog Devices\CrossCore Embedded Studio 2.2.0\SHARC\ldr

The corresponding source code and project files can be found in the \init_code\ subdirectory.

The default CGU configuration which the DMC initialization code is designed for is as follows:

1. ADSP-SC589 EZ-Board – Micron MT41K128M16-125 DDR3 device[^7] with:
   - PLLCLK = CCLK = DCLK = 450 MHz
   - SYSCLK = 225 MHz
   - SCLK0 = SCLK1 = 112.5 MHz

2. ADSP-SC584 EZ-Board – Micron MT47H128M16RT-25E XIT:C DDR2 device[^8] with:
   - PLLCLK = CCLK = DCLK = 400 MHz
   - SYSCLK = 200 MHz
   - SCLK0 = SCLK1 = 100 MHz

3. ADSP-SC573 EZ-Board – Micron MT41K128M16-125 DDR3 device[^7] with:
   - PLLCLK = CCLK = DCLK = 450 MHz
   - SYSCLK = 225 MHz
   - SCLK0 = SCLK1 = 112.5 MHz
Modifying Default Preload and Initialization Code for Customized CGU/DMC Settings

The CGU and DMC settings in the default preload and initialization source code may need to be modified for the following conditions:

1. When using the EZ-Board (or a custom board with the same DDR memory device as that populated on the EZ-Board) with non-default CGU settings.
2. When using a custom board with a different memory device.

For example, the following steps can be followed to modify the default preload code in the \ldr\init_code\SC58x_Init directory in CCES to interface the ADSP-SC589 processor to a Micron MT46H128M16LFDD-48 LPDDR memory device with a DCLK of 200 MHz:

1. Copy the sc589_preload_Core0 and src folders from \ldr\init_code\SC58x_Init and paste them to a new folder called SC589_LPDDR.
2. Create a workspace in the new SC589_LPDDR folder and import the sc589_preload_Core0 project into the workspace.
3. Replace the DMCInit.c and DMCInit.h files in the init_src folder with the ones provided in the associated ZIP file.
4. In the project properties window, remove the DMCInit.c and DMCInit.h files from the filter, as shown in Figure 7:

![Figure 7. Modifying Project Options for Building Custom ADSP-SC589 Preload Code](image)

5. Modify the CGU settings in the sc58x_init.h file to generate CCLK = 400 MHz, SYSCLK = 200 MHz, SCLK = 100 MHz, and DCLK = 200 MHz. As shown in Figure 8, MSEL is modified from 18 to 16, and DSEL is modified from 1 to 2 to achieve these settings.
6. Add the customized DMC register settings in the sc58x_init.h file, and replace the DMC initialization portion of the code for both DMC0 and DMC1 under the conditional build __CONFIG_DMC0__ and __CONFIG_DMC1__ macros, respectively. Be sure to also verify that the DMC_PARAM DMC_Param_List variable is defined, as shown in Figure 8.
7. Now build the project, and the customized preload DXE file is available in the \Debug folder. Similarly, the preload code for the ADSP-SC584 evaluation platform, the ADSP-SC573 evaluation platform and the initialization code for the ADSP-SC589, ADSP-SC584 and ADSP-SC573 evaluation platforms can be modified for any customized CGU and DMC settings.

Although this EE-note provides guidance to customize the preload and initialization code for the ADSP-SC589 (DDR3 and LPDDR modes), ADSP-SC584 (DDR2 mode) and ADSP-SC573 (DDR3, DDR2 and LPDDR modes) processors, the examples should be used only as a reference to understand the procedure. It is the user’s responsibility to ensure that the latest preload and initialization code from the CCES installation is the starting point and careful modification is performed when making the required customization to support the desired CGU/DMC settings of the end application.

```
// Initialize the DMC parameters list
DMuParam_t DMC_Params[DMC_PARAM_COUNT];

// examples from DMC.getParamList();
DMC_Param_t DMC_Param_list;

// Initialize the DMC parameters
DMC_Init(DMC_Param_list);  

// DMC-SC589, ADSP-SC584 and ADSP-SC573 evaluation platforms

// DMC-SC589
DMC_Init(DMC_Param_list);  

// DMC-SC584
DMC_Init(DMC_Param_list);  

// DMC-SC573
DMC_Init(DMC_Param_list);  

```

**Figure 8. Modifying Source Code Files for Customized CGU/DMC Settings**

!!!important!!!

Although this EE-note provides guidance to customize the preload and initialization code for the ADSP-SC589 (DDR3 and LPDDR modes), ADSP-SC584 (DDR2 mode) and ADSP-SC573 (DDR3, DDR2 and LPDDR modes) processors, the examples should be used only as a reference to understand the procedure. It is the user’s responsibility to ensure that the latest preload and initialization code from the CCES installation is the starting point and careful modification is performed when making the required customization to support the desired CGU/DMC settings of the end application.
**Frequently Asked Questions**

**Question 1:** I am using the same DDR device as used on the EZ-Board, but I want to run the DMC at a different DCLK value than the EZ-Board. What do I need to take care of when initializing the DMC?

**Answer:** Just change the DCLK parameter value in the DMC_Registers_List.xlsx spreadsheet and copy the generated DMC register values into the main.h file of the corresponding example DMC initialization code provided in the associated ZIP file[6]. For example, Figure 9 shows how to generate and use the DMC register values for the Micron MT41K128M16-125 DDR3 device with a DCLK frequency of 400 MHz instead of 450 MHz.

![Figure 9. Generating DMC Register Values for the MT41K128M16-125 DDR3 Memory Device w/ DCLK = 400 MHz](image)

**Question 2:** I am using the ARM core, and all my data and code is in the external DDR memory. How can I re-initialize the DMC with a non-default DCLK value without affecting the memory contents?

**Answer:** To make sure that the code/data in the DMC is retained, initialize the CGU using the steps mentioned in the On-the-Fly DMC Re-Initialization section of this EE-note. By default, the linker file for the ARM core on the ADSP-SC5xx processors puts all the code and data in the L3 (DMC) space. Thus, a custom linker file is required to place the complete CGU and DMC re-initialization code into non-L3 (DDR) space (for example, L2). The DDR3_Re_Initialization_ARM_Core0 code example in the associated ZIP file[6] illustrates how to reinitialize the DMC through the ARM core to run the DCLK at 400 MHz instead of the default DCLK frequency of 450 MHz.

**Question 3:** Do I need to re-initialize PHY registers each time I reinitialize the DMC dynamically in my code?

**Answer:** PHY registers do not need to be initialized every time the DMC is reinitialized, unless the pads need to be calibrated to a different driver impedance and/or ODT.
**Question 4:** Can I run the DDR device at a DCLK of 400 MHz and still run the core at 450 MHz?

Yes, this is possible using two CGUs. One CGU can generate a CCLK of 450 MHz (for example, with SYSCLK = 225 MHz and SCLK = 112.5 MHz), and the other CGU can generate a DCLK at 400 MHz. The DDR2_Init_Two_CGU_Core1 example in the associated ZIP file[6] illustrates how to configure the DDR2 device on the ADSP-SC584 EZ-Board with a DCLK of 400 MHz while maintaining a CCLK of 450 MHz.

**Question 5:** Does the controller perform only a one-time DLL calibration during DMC initialization?

**Answer:** No, during initialization, the controller does a coarse DLL calibration. During runtime, the controller continues with fine calibration during auto-refresh commands and continues to update the corresponding calibration registers automatically to accommodate for temperature/PCB changes during runtime.

**Hardware Considerations**

Standard DDR interface board design guidelines are presented in this section to have optimal ADSP-SC5xx DMC interface signal and power integrity.

**Placement and General Routing Guidelines**

- Place the ADSP-SC5xx and the DDR3/DDR2/LPDDR memory as close as possible to each other, as this reduces routing length.
- The ADSP-SC5xx DMC interface supports only point-to-point design and does not support fly-by topology.
- Plan the stack-up such that all the DDR3/DDR2/LPDDR signals have continuous and immediate reference planes, and ensure that they do not cross splits.
- The DDR3/DDR2/LPDDR address, command, and control signals can use an even power plane as a reference. Ensure that there are no splits or discontinuities in the referenced power plane.
- The PCB trace characteristic impedance must be 50 Ω for single-ended signals and 100 Ω for differential signals.
- Route all the DDR signals as a group in each and every layer to avoid a mismatch in trace impedance and propagation delay.
- Ensure that the DDR signals are routed as groups and that signals within groups do not change layers.
  - For example, the DMC_DQ00-07, DMC_LDQS, and DMC_LDQM signals should be routed as a group, in the same layer and having the same reference. Changing the reference plane can change the trace impedance.
- To avoid crosstalk, ensure that all DDR signals have sufficient center-to-center spacing of at least 3W between DDR signals and 4W to other signals.
• Maintain perpendicularity between DMC signal routing if the DMC signals are routed in the adjacent layers. This configuration reduces crosstalk, as signals on adjacent layers will not be parallel to each other.

• Avoid test points on the DDR signals, as they create stubs which can act as an EMI source. Instead, use vias for probing or JEDEC-recommended methods that DDR memory vendors offer for diagnostics.

**Trace Length Matching Criteria**

The routing of all the DDR interface signals must be length-matched to avoid set-up and hold time violations due to propagation delay.

The length-matching criteria are as follows:

• Match the trace length of all address (DMC_A[nn], DMC_BA[n]) and command (DMC_CKE, DMC_CS[n], DMC_ODT, DMC_RAS, DMC_RESET, DMC_WE) signals within +/- 40 mils relative to the DMC_CK signal.

• Match the trace length of all data (DMC_DQ[nn]) and data mask (DMC_UDM, DMC_LDM) signals within +/- 40 mils relative to their corresponding DQS signal.
  
  o For example, the lower order data byte (DMC_DQ00 - DMC_DQ07) and the corresponding data mask (DMC_LDM) signals should be length-matched with the lower order strobe (DMC_LDQS) signal.

• The differential signals such as clock (DMC_CK and /DMC_CK) and DQS pairs (DMC_LDQS and /DMC_LDQS, DMC_UDQS and /DMC_UDQS) should be length-matched within +/- 10 mils.
  
  o For example, the trace length of the DMC_CK and /DMC_CK signals should be matched within +/- 10 mils relative to each other.

• The maximum allowed trace length for DDR signals is 2 inches.

**Interface Termination**

All the DDR interface signals without ODT require external termination for improved signal integrity.

The termination guidelines are as follows:

• Install a 100 Ω termination resistor on the differential clock between DMC_CK and /DMC_CK, and place it close to the memory.

• Install external series termination resistors on all address and control signals, and place them closer to the processor.

• The value of the series termination resistors on the ADSP-SC5xx EZ-Board is 33 Ω. This value can vary based on the routing in the board. Appropriate series termination resistor values can be obtained based on simulation results of the board. The IBIS models for the ADSP-SC5xx processor can be downloaded from www.analog.com.

• The DDR2/3 data group signals do not require external termination, as they have ODT.
• The ODT value can be obtained based on simulation results of the board. IBIS models for the ADSP-SC5xx processor can be downloaded from www.analog.com.

• For the LPDDR interface, it is advisable to also install termination resistors for data signals, as ODT is not supported for LPDDR mode.

**DMC Power (VDD_DMC) Decoupling**

The DMC interface should have sufficient decoupling on the VDD_DMC rail and the memory power rail to avoid data corruption.

• Use the ADSP-SC589 EZ-Board[3]/ADSP-SC584 EZ-Board[4]/ADSP-SC573 EZ-Board[16] schematics as a reference to identify the decoupling capacitor quantity and values for the ADSP-SC5xx VDD_DMC rail.

• Refer to the DDR memory datasheet or consult the DDR memory vendor to identify the decoupling capacitor requirement for the DDR memory power rails. A critical parameter is $I_{dd7}$, which defines the peak current during multi-bank operation, depending on the speed grade and ambient temperature of the DDR memory.

• Place all the decoupling capacitors very near to the VDD_DMC power rail and use a solid power/ground plane.

• Ensure that the power and ground planes are adjacent to each other to provide the shortest return path, as well as higher capacitance.

• Have individual power and ground vias for every power and ground pin of the ADSP-SC5xx processor and the memory device to its associated plane.

• Place the decoupling capacitors for the VDD_DMC power rail as close as possible to the pin. This is very important, as DDR signal slew rates are really aggressive, so having these capacitors closer to the pin is a must (Figure 10).
Figure 10. Sample VDD_DMC Decoupling Capacitor Placement for the ADSP-SC584 Device

- Have dedicated power and ground vias for each decoupling capacitor pin. Sharing of vias is not allowed.

- Ideally, the trace length from the power via to the device pad should not exceed 30 mils. The maximum trace length from power via to bypass capacitor is 60 mils. The maximum trace length from power via to power ball pad is 35 mils.

- Placement of the mid-bulk bypass capacitors (10 µF) is not that critical, and they can be placed to accommodate other circuitry with more constrained placement and routing requirements.

Clean DMC_VREF Voltage Supply

DMC_VREF acts as a voltage reference for DDR2/3 data signals and compares the difference between a steady reference voltage (VREF) and the signal received for identifying the logic. Hence, it is recommended to have them clean and noise-free.

- Route the DMC_VREF trace away from high-speed signals and noisy power supplies with a distance of at least 40 mil.

- Guard traces can be provided around the DMC_VREF trace, if required. Ensure that the guard traces have sufficient ground vias stitched to the main ground plane.

- Provide adequate decoupling near the DMC_VREF pins of the ADSP-SC5xx processor, as well as the memory device.
• Keep the DMC_VREF trace as short as possible with a width of at least 20 mils.

**Recommended ADSP-SC5xx DMC_VREF Filtering Scheme**

• The RCR filtering scheme on DMC_VREF is recommended for ADSP-SC5xx products, as shown in Figure 11.

![Figure 11. Recommended DMC_VREF Filtering Network](image)

**PCB Placement and Routing Guidelines for the DMC_VREF Filter Network**

• Place the recommended RCR network between the DMC_VREF supply paths from the memory to the processor, as shown in Figure 12. For more details, refer to the ADSP-SC589 EZ-Board schematics[^3].

![Figure 12. Recommended Placement and Routing for DMC_VREF Filtering Network](image)

• Place the C(100 pF)-R(2 kΩ) portion of the RCR network as close as possible to the ADSP-SC5xx DMC_VREF pin.
• Use a small package, preferably 0402 size, for the 100pF capacitor to guarantee a high self-resonant frequency.

• Place the entire RCR network in one layer with no vias allowed in the trace.

**Memory-Side Recommendations for VREF Supply**

There are unique requirements for the VREF supply, depending on whether the design is for DDR2 or DDR3 memory.

**VREF Supply Recommendations for DDR3 Mode**

- DDR3 memory has two VREF pins: VREFCA and VREFDQ. VREFCA serves as the reference for clock, address, command, and control signals, and VREFDQ serves as the reference for strobe, data, and data mask signals.

- VREFCA and VREFDQ can have a common supply source but should be “STAR” routed and decoupled at the dedicated DRAM pins.

- Place two decoupling capacitors, 0.1 µF and 0.01 µF, for each VREF pin (VREFCA and VREFDQ).

- Place the 0.01 µF capacitor closer to the DRAM pin, followed by the 0.1 µF capacitor.

- Keep the length from the decoupling capacitor to the DRAM pin short with a wide trace.

**Figure 13** shows the recommended VREF supply circuit for a DDR3 memory device.

**Figure 13. Recommended VREF Supply Circuitry for DDR3 Memory Device**

**VREF Supply Recommendations for DDR2 Mode**

- DDR2 memory has only one VREF pin. Add 0.01 µF and 0.1 µF decoupling capacitors on VREF close to the DDR2 SDRAM.

- Place the 0.01 µF capacitor closer to the DRAM pin, followed by the 0.1 µF capacitor.

**Figure 14** shows the VREF supply circuitry used on the ADSP-SC589 EZ-Board for DDR3 mode, and **Figure 15** shows the same for the ADSP-SC584 EZ-Board for DDR2 mode.
**Recommendation for differential signals**

To meet JEDEC standard, the circuits shown in Figure 16 and Figure 17 are recommended for clock and DQS signals.

**Clock Recommendation**

For DDR2 and DDR3 mode, the DMC_CK signal requires a pull-up resistor and the DMC_CK# signal requires a pull-down resistor that is placed close to the memory.
**DQS Recommendation**

For DDR3 mode only, the `DMC_UDQS/DMC_LDQS` signal requires a pull-up resistor and the `DMC_UDQS#/DMC_LDQS#` signal requires a pull-down resistor closest to the memory.

---

**Figure 16. Clock Signal**

![Clock Signal Diagram]

**Figure 17: DQS Signal**

![DQS Signal Diagram]
The recommended values of resistor “x” and resistor “y” are 680Ω and 1200Ω, respectively, for ADSP-SC58x processors and 300Ω and 390Ω, respectively, for ADSP-SC57x processors.

References


## Document History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Rev 3 – April 1st, 2018**<br>by Sachin-V Kumar | Added a new note in “Trace length matching criteria” section  
Added a correction to a note in “Trace length matching criteria” section  
Changed “Listing.1 CGU initialization”  
Added new section “Recommendation for differential signals”  
Minor changes in example codes |
| **Rev 2 – June 25th, 2016**<br>by Sachin-V Kumar | Added descriptions to include ADSP-SC57x and ADSP-2157x series processors  
Added example codes for ADSP-SC57x series processors |
| **Rev 1 – December 17th, 2015**<br>by Balaji Kannan, Mitesh Moonat, and Gaurav Singh | Initial Release |