ADSP-2159x/SC59x Programming Guidelines for Dynamic Memory Controller

Contributed by Deepak SH *1 – September 19, 2022*

Introduction

The ADSP-2159x/SC59x SHARC+® processor incorporates a Dynamic Memory Controller (DMC), which provides a glueless interface between off-chip DDR3 memory devices and the rest of the processor infrastructure. For further technical details on the DMC module, refer to the *ADSP-21594/ADSP-SC591/SC592/SC594: SHARC+ Dual-Core DSP with Arm Cortex-A5 Data Sheet*[1], *ADSP-21591/21593/21594/ADSP-SC591/SC592/SC594: SHARC+ Dual-Core DSP with Arm Cortex-A5 Data Sheet*[2], and the *ADSP-2159x/ADSP-SC591/592/594 SHARC+ Processor Hardware Reference*[3]. This EE-note discusses some of the important programming guidelines that must be followed when interfacing ADSP-2159x/SC59x SHARC+® processor with a DDR memory device. The associated *zip file*[4] includes code examples that can be used for basic DMC initialization, DMC initialization using `DMC_Registers_List_2159x_SC59x.xlsx` spreadsheet and DMC re-initialization. The code examples include a subroutine which can be used to validate the DMC interface for different types of accesses (for example, core, DMA, 8-/16-/32-/64-bit) and data patterns (for example, all 0x0, all 0xF, all 0x5, all 0xA, incremental, random, and all bits toggling).

Software Considerations – DMC Programming Model

Figure 1 shows the DMC programming flow. DMC initialization consists of:

- Clock Generation Unit (CGU) Initialization
- DMC PHY Initialization
- DMC Controller Initialization

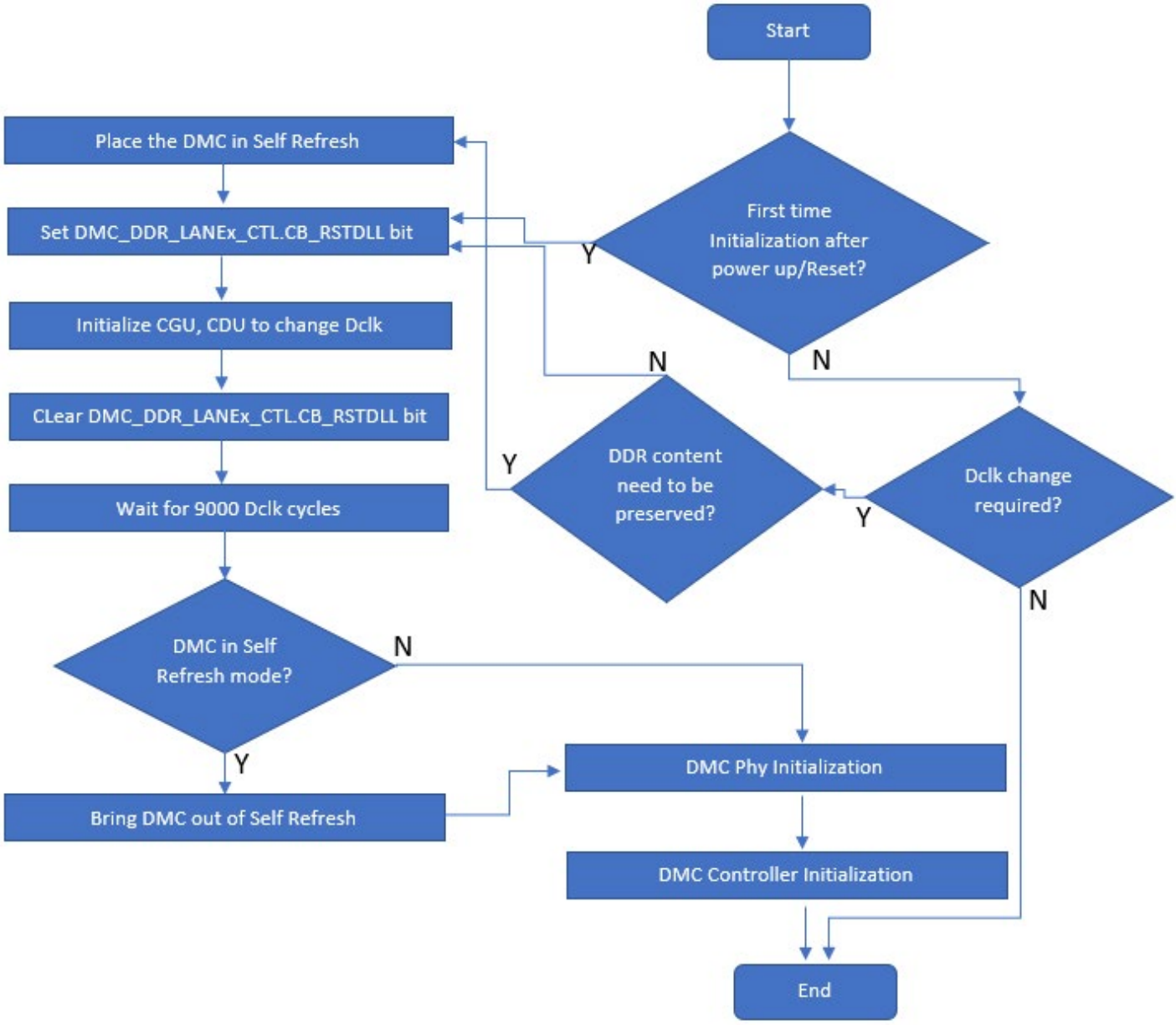

Figure 1: DMC Programming Model Flow Chart

CGU Initialization

Verify that the DDR clock (DCLK) is configured to the required frequency. On ADSP-2159x/SC59x SHARC+® Processor, DCLK clocks DMC0 however, DCLK can come from either CGU0 (default) or CGU1 by programming the CDU. Route DCLK from CGU1 when the required DCLK frequency is asynchronous to the CCLK and SYSCLK frequencies.

For example, assume a case where the required CCLK frequency is 1000 MHz, the SYSCLK frequency is 500 MHz, and the DCLK frequency is 800 MHz. Achieving this frequency combination may not be possible with a single CGU. To realize this configuration, generate CCLK and SYSCLK using CGU0 and

DCLK using CGU1. For details on how to program the CGU and Clock Distribution Unit (CDU), refer to the *Hardware Reference*[3]

Once the DMC is initialized, ensure that the DCLK frequency has not changed.

DMC Initialization

After reset, configure the DCLK generated from CGU0 could be set to the default frequency. The CGU must be re-initialized to configure the DCLK to the required new frequency. As shown in Figure 1, complete the following steps to initializing the CGU for the first time after reset:

1. Set (=1) the DMC_DDR_LANE0_CTL0.CB_RSTDLL and DMC_DDR_LANE1_CTL0.CB_RSTDLL bits.
2. Change the DMC clock frequency.
3. Clear (=0) the DMC_DDR_LANE0_CTL0.CB_RSTDLL and DMC_DDR_LANE1_CTL0.CB_RSTDLL bits.
4. Wait 9000 DCLK cycles for the DLL to lock.

Typically, the CGU is first initialized in either *preload code* (when the application is loaded through the emulator) or by *init code* (when the application is loaded by the boot process, in the init block) The code 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 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 to be preserved, use the same steps as described in DMC Initialization 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. Ensure that the DMC is in the idle state by waiting for the DMC_STAT.IDLE bit to be set (=1).
2. Place the DMC into self-refresh mode by setting (=1) the DMC_CTL.SRREQ bit.
3. Poll the DMC_STAT.SRREQ bit to set (=1); wait for the self-refresh mode transition to complete.
4. Set (=1) the DMC_DDR_LANE0_CTL0.CB_RSTDL and DMC_DDR_LANE1_CTL0.CB_RSTDLL bits.
5. Initialize the CGU and CDU to change the DCLK frequency.
6. Clear (=0) the DMC_DDR_LANE0_CTL0.CB_RSTDLL and DMC_DDR_LANE1_CTL0.CB_RSTDLL bits.
7. Wait 9000 DCLK cycles for the DLL to lock.
8. Bring the DMC out of self-refresh mode by clearing (=0) the DMC_CTL.SRREQ bit.

9. Poll the DMC_STAT.SRREQ bit to clear (=0); wait for the self-refresh exit to complete.

When re-initializing the DMC, the CGU/DMC initialization code should **not** be executed from the DDR memory. Refer to the `SC594_DMC_Re_Initialization_A5_Core0` project in the associated *zip file*[4]

1. Use `__attribute__((section(".l2_cached_code")))` to place the functions in internal memory for the A5 core. Use `_Pragma("section(\"seg_int_code\")")` to place the functions in internal memory for the SHARC core.
2. Use `__attribute__((section(".l2_cached_data")))` to place the data in internal memory for the A5 core. Use `_Pragma("section(\"seg_int_data\")")` to place the data in internal memory for SHARC core.
3. Change the PWR service files as follows:
   a. Place `adi_pwr_ClockInit`, `adi_pwr_Init` and `adi_pwr_SelectCduClockSource` functions in internal memory.
   b. Remove `adi_osal_ExitCriticalRegion` and `adi_osal_EnterCriticalRegion` function calls in the `adi_pwr_WriteDIVCTLLocal` function.
      a. Place `adi_pwr_ClockInit`, `adi_pwr_Init` and `adi_pwr_SelectCduClockSource` functions in internal memory.
      b. Remove `adi_osal_ExitCriticalRegion` and `adi_osal_EnterCriticalRegion` function calls in the `adi_pwr_WriteDIVCTLLocal` function.

The first 16 bytes of DDR memory are overwritten by controller during initialization.

DMC PHY Initialization

Refer to the Performing ZQ Calibration and Programming Duty Cycles section in the *Hardware Reference* [2]. Ensure that the workaround to anomaly 20000117 from the *Silicon Anomaly List*[5] is applied.

DMC Controller Initialization

Table 1 through Table 4 show the bit fields used to program the DMC. Refer to the Programming the DMC Controller and Programming DQ Delay Trim sections in the *Hardware Reference*[3]. The controller has a set of registers with bit fields that control:

- 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. These are shaded in ORANGE in the `DMC_Registers_List_2159x_SC59x.xlsx` spreadsheet operation in the application. These are shaded in GREEN in the `DMC_Registers_List_2159x_SC59x.xlsx` spreadsheet. For more details on how to program these bit fields, refer to the *Hardware Reference*[3].

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. 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*[3] and the corresponding memory device data sheet (especially for mode registers). These bits are shaded in YELLOW in the `DMC_Registers_List_2159x_SC59x.xlsx` spreadsheet.

Register	Bit Field		Bit field (C=Controller, M=JEDEC)	Value	Comment
DMC_CTL	DDR3EN	DDR3 mode enable	0-C	Mandatory	Always program to **1** for standard DMC operation.
	INIT	Initialize DRAM Start	2-C		Always program to **1** for standard DMC operation.
	SRREQ	Self-Refresh Request	3-C	Optional	Program **0** for standard DMC operation.
	PDREQ	Power Down Request	4-C		Program **0** for standard DMC operation.
	PREC	Precharge	6-C		Program **0** for standard DMC operation.
	RESET	Reset SDRAM	7-C		Program **0** for standard DMC operation.
	ADDRMODE	Addressing (Page/Bank) Mode	8-C		Program **0** for standard DMC operation.
	RDTOWR	Read-to-Write Cycle.	11:9-C	Mandatory	Always program to **5** for standard DMC operation.
	PPREF	Postpone Refresh	12-C	Optional	Program **0** for standard DMC operation.
	DLLCAL	DLL Calibration Start	13-C		Program **0** for standard DMC operation.
	Reserved	Reserved	23:14-C	Mandatory	Always write these bits with **zero**.
	ZQCS	ZQ Calibration Short	24-C	Optional	Program **0** for standard DMC operation.
	ZQCL	ZQ Calibration Long	25-C		Program **0** for standard DMC operation.
	RL_DQS	Read leveling during DQS	26-C		Program **0** for standard DMC operation.

		Gating Training.			
	AL_EN	Additive Latency Enable	27-C	Mandatory	Program 1 for Dclk frequency above 667 MHz
	Reserved	Reserved	31:28	Mandatory	Always write these bits with **zero**.

Table 1: DMC Control Register Bit Fields

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

Table 2: DMC Configuration Register Bit Fields

Register	Bit Field		Bit field (C=Controller, M=JEDEC)	Value	Comment
DMC_TR0	TRCD	RAS# to CAS# delay time	3:0-C		Obtain from memory device data sheet.
	TWTR	Write-to-Read delay	7:4-C		
	TRP	Precharge-to-Active time	11:8-C		
	TRAS	Active-to-Precharge time	16:12-C		
	Reserved	Reserved	19:17-C		Always write these bits with zero.
	TRC	Active-to-Active time	25:20-C		Obtain from memory device data sheet.
	Reserved	Reserved	27:26-C		Always write these bits with zero.
	TMRD	Mode register set-to-active	31:28-C		Obtain from memory device data sheet.
	TREF	Refresh	13:0-C		

DMC_TR1		Interval		Mandatory	
	Reserved	Reserved	15:14-C		Always write these bits with zero.
	TRFC	Refresh-to-Active command delay	24:16-C		Obtain from memory device data sheet.
	Reserved	Reserved	27-25-C		Always write these bits with zero.
	TRRD	Active-to-Activetime	30-28-C		Obtain from memory device data sheet.
	Reserved	Reserved	31		Always write this bit with zero.
DMC_TR2	TFAW	Four Activate Window	4:0-C		Obtain from memory device data sheet. tFAW is not applicable for LPDDR mode and should be kept zero.
	TFAW5	Extended Timing Four-Active Window bit 5	5-C		
	Reserved	Reserved	7:6-C		Always write these bits with zero.
	TRTP	Internal Read to Precharge time	11:8-C		Obtain from memory device data sheet. tRTP is not applicable for LPDDR mode and should be kept zero.
	TWR (LPDDR only)	Write recovery time	15:12-C		Obtain from memory device data sheet.
	TXP	Exit power down to next valid command	19:16-C		
	tCKE	CKE min pulse width	23:20-C		
	Reserved	Reserved	31:24-C		Always write these bits with zero.

Table 3: DMC Timing Register Bit Fields

Register	Bit Field		Bit field (C=Controller, M=JEDEC)	Value	Comment
DMC_MR0	BL	Burst Length	1:0-C, A1:A0-M	Mandatory	Only BL=8 is supported for DDR3. Always program these bits with zero.
	CL	CAS Latency	6:4,2-C, A6:A4, A2-M		Program these bits with the required CAS latency.
	Reserved	Reserved	3-C, A3-M		Always write this bit with zero.
	Reserved	Reserved	7-C, A7-M		Always write this bit with zero.
	DLLRST	DLL Reset	8-C, A8-M		Set this bit for DDR3 mode.
	WRRECOV	Write recovery	11:9-C, A11:A9-M		Program these bits with tWR value from the memory device datasheet. Refer to the hardware reference manual[3] for more details.
	PD	Active Power Down Mode	12-C, A12-M	Optional	Can be left unchanged for standard DMC operation. Refer to the hardware reference manual[3] for more details on these bits.
	Reserved	Reserved	15:13-C, A15:A13-M	Hard Wired	These bits are hard-wired to zero.
	Reserved	Reserved	31:16-C		
DMC_MR1	DLLEN	DLL Enable	0-C, A0-M	Mandatory	Keep this bit set to zero.
	DIC0, DIC1	Output Driver Impedance Control	5,1-C, A5,A1-M		Select the driver impedance using these bits from the memory side.
	RTT0, RTT1, RTT2	On Die Termination (ODT)	9,6,2-C, A9,A6,A2-M		Select ODT value using these bits from the memory side.
	AL	Additive Latency	4,3-C, A4,A3-M		Can be cleared for basic DMC initialization. Refer to the memory device data sheet for more details.
	WL	Write Leveling	7-C, A7-M		This bit shall be written with one
	Reserved	Reserved	8, 10-C, A8, A10 –M		These bits are reserved for future use (must be programmed to zero).
	TDQS	Termination Data Strobe	11-C, A11-M		Should be zero, as it is not applicable for 16-bit devices.
	QOFF	Output Buffer Enable	12-C, A12-M		Should be zero.
	Reserved	Reserved	15:13-C, A15:A13-M	Hard Wired	These bits are hard-wired to zero.
	Reserved	Reserved	31:16-C		
	PASR	Partial Array Self Refresh	2:0-C, A2:A0-M	Optional	This bit is unchanged for standard DMC operation. Refer to the hardware reference

					manual[3] for more details.
	CWL	CAS Write Latency	5:3-C, A5:A3-M	Mandatory	Obtain from memory device data sheet.
DMC_MR2	ASR	Auto Self Refresh	6-C, A6-M	Optional	These bits are unchanged for standard DMC operation. Refer to the hardware reference manual[3] for more details.
	SRT	Self-Refresh Temperature Range	7-C, A7-M		
	Reserved	Reserved	31-8-C	Hard Wired	These bits is hard-wired to zero.

Table 4: DMC DDR3 Mode Register Bit Fields

DMC Initialization Code

The associated *zip file*[4] provides code examples which can be used to initialize the CGU and DMC controller for any custom settings.

CGU Initialization

For custom clock settings, the structures `ADI_PWR_CGU_PARAM_LIST` and `ADI_PWR_CDU_PARAM_LIST` in the `adi_pwr_SC59x_config.c` file can be changed accordingly. Also, `cclkdclk_ratio` ratio shall be changed accordingly, for example when Cclk is 1000 MHz and Dclk is 800 MHz the ratio will be 1000/800 = 1.25.

DMC Initialization

The `adi_dmc.c` and `adi_dmc.h` files can be used to initialize the DMC to the required settings. For example, the `main.c` file in the `ADSP-SC594_DMCconfigGenerator_core1` project in the associated *zip file*[4], illustrates two ways to initialize the DMC for a DDR3 memory with DCLK frequency of 800 MHz as per JESD79_3F JEDEC specification[6].

- **Initialize_DMC_Basic** – Enable this macro in the `main.h` file to initialize the DMC by configuring the `ADI_DMC_PARAM_LIST` structure. Figure 2 shows a snapshot from the `main.c` file of the `ADI_DMC_PARAM_LIST` structure of code used to initialize the DMC with the `Initialize_DMC_Basic` macro. All of the DDR parameters required to initialize DMC/DDR memory are computed based on the JESD79-3F JEDEC specification[6]. The `Initialize_DMC_Basic` approach can be used to quickly test DDR across different frequencies, DriveStrength, and ODT settings. The complete list of DDR parameters used in this method can be printed on a console using the `adi_printDMCconfig()` API (enable the `Print_DMC_Config` macro). The DDR parameters printed are in the format used in the `adi_dmc_SC59x_family_x_config.h` file of the preload/Init code.

```
ADI_DMC_PARAM_LIST Dmc_config;

Dmc_config.DDRSpeedBin = ADI_DDR3_MEM_1600;      /* Choose the Speedbin based on the DDR clock frequency */
Dmc_config.DDRClockTimePeriod = 1.25f;           /* Time period or tck of DDR clock */
Dmc_config.CoreClockTimePeriod = 1.00f;          /* Time period of Core clock */
Dmc_config.ProAddCmdDrv = 100;                   /* Processor's DriveStrength of Address and command */
Dmc_config.ProClkDqsDrv = 90;                    /* Processor's DriveStrength of DQ, DQS, DM and clock */
Dmc_config.ProOdt = 75;                          /* Processor's ODT of DQ, DQS, DM */
Dmc_config.MemSize = ADI_DDR3_MEM_8Gb;           /* size of DDR memory */
Dmc_config.MemTemp = ADI_DDR3_MEM_NOMINAL_TEMP; /* Operating temperature range of DDR memory */
Dmc_config.MemDrv = ADI_DDR3_DRV_40;             /* DDR memory's DriveStength */
Dmc_config.MemOdt = ADI_DDR3_ODT_120;            /* DDR memory's ODT */

cclkdclk_ratio =  (Dmc_config.DDRClockTimePeriod/ Dmc_config.CoreClockTimePeriod);
```

Figure 2: ADI_DMC_PARAM_LIST Snapshot

Based on JESD79-3F JEDEC specifications[6] given for a particular speed bin, the DDR parameters printed or used for configuring the DMC with the `Initialize_DMC_Basic` method should work from a functional perspective. However, check the parameter values with the specific memory device data sheet using the `DMC_Registers_List_2159x_SC59x.xlsx` spreadsheet spreadsheet.

- **Initialize_DMC_Advanced -** Enable this macro in `main.h` file to initialize the DMC by configuring the `ADI_DMC_CONFIG` structure. This structure is the same as the one used in the `adi_dmc_SC59x_family_x_config.h` file of the preload/Init code. For custom DMC settings, the `ADI_DMC_CONFIG` structure must be updated according to the system requirements as shown in Figure 3 and Figure 4. The `ADI_DMC_CONFIG` structure can be configured using the `DMC_Registers_List_2159x_SC59x.xlsx` spreadsheet in the associated *zip file*[4] by entering various DMC-specific and DDR memory-specific parameters (from the device data sheet). See Figure 5. Use the generated hex values for the `ADI_DMC_CONFIG` structure from the `DMC_Registers_List_2159x_SC59x.xlsx` spreadsheet as shown in Figure 6 to configure the macros in `main.h` file as shown in Figure 3. This method of DMC initialization can be used to customize the DDR initialization routine in the preload/initcode as per the DDR memory data sheet; to gives the flexibility to fine tune any of the required DDR parameters.

```
#define CFG0_REG_DDR_DLLCTLCFG  0x0cf00622ul
#define CFG0_REG_DMC_MR2MR3     0x00180004ul
#define CFG0_REG_DMC_CTL_VALUE  0x08000a05ul
#define CFG0_REG_DMC_MRMR1      0x0d7000c0ul
#define CFG0_REG_DMC_TR0_VALUE  0x4271cb6bul
#define CFG0_REG_DMC_TR1_VALUE  0x61181860ul
#define CFG0_REG_DMC_TR2_VALUE  0x00450620ul
#define CFG0_REG_DMC_ZQCTL_0    0x00785A64ul
#define CFG0_REG_DMC_ZQCTL_1    0x00000000ul
#define CFG0_REG_DMC_ZQCTL_2    0x70000000ul
```

Figure 3: main.h file Snapshot

```
96      cclkdclk_ratio = 1.25;
97
98      /* Set DMC Lane Reset */
99      adi_dmc_lane_reset(true);
100
101     /* Initialize CGU and CDU */
102     if((uint32_t)adi_pwr_cfg0_init() != 0)
103     {
104         return ADI_DMC_FAILURE;
105     }
106
107     /* Clear DMC Lane Reset */
108     adi_dmc_lane_reset(false);
109
110     // add macros here in a header file and that can be used here
111     static ADI_DMC_CONFIG config =
112     {
113             CFG0_REG_DDR_DLLCTLCFG,             /* ulDDR_DLLCTLCFG */
114             CFG0_REG_DMC_MR2MR3,                /* ulDDR_EMR2EMR3 */
115             CFG0_REG_DMC_CTL_VALUE,             /* ulDDR_CTL */
116             CFG0_REG_DMC_MRMR1,                 /* ulDDR_MREMR1 */
117             CFG0_REG_DMC_TR0_VALUE,             /* ulDDR_TR0 */
118             CFG0_REG_DMC_TR1_VALUE,             /* ulDDR_TR1 */
119             CFG0_REG_DMC_TR2_VALUE,             /* ulDDR_TR2 */
120             CFG0_REG_DMC_ZQCTL_0,               /* ulDDR_ZQCTL0 */
121             CFG0_REG_DMC_ZQCTL_1,               /* ulDDR_ZQCTL1 */
122             CFG0_REG_DMC_ZQCTL_2                /* ulDDR_ZQCTL2 */
123     };
124
125     /* Initialize DMC PHY registers */
126     adi_dmc_phy_calibration(&config);
127
128     /* Initialize DMC Controller */
129     if(adi_dmc_ctrl_init(&config) != ADI_DMC_SUCCESS)
130     {
131         return ADI_DMC_FAILURE;
132     }
```

Figure 4: main.c file Snapshot

Parameter	Value	Unit
DCLK	800	MHz
SDRAM Size	4096	MB
tRCD	13.75	ns
tWTR	6	tCK
tRP	13.75	ns
tRAS	35	ns
tRC	48.75	ns
tMRD	4	tCK
tREFI	7.8	us
tRFC	260	ns
tRRD	6	tCK
tFAW	40	ns
tRTP	6	tCK
tWR	15	ns
tXP	5	tCK
tCKE	4	tCK
CAS Read Latency (CL)	11	tCK
Burst Length	8	tCK
Driver Impedance (Memory)	RZQ/6(40)	Ohms
On Die Termination (Memory)	RZQ/2(120)	Ohms
Driver Impedance (Processor-	100	Ohms
Driver Impedance (Processor-	90	Ohms
On Die Termination (Processor)	75	Ohms
Additive Latency (AL)	AL Disabled	tCK
CAS Write Latency (CWL)	8	tCK

Figure 5: DMC_Registers_List_2159x_SC59x.xlsx Snapshot

ADI_DMC_CONFIG Structure	32 bit Hex value
uIDDR_DLLCTLCFG	CF00622
uIDDR_EMR2EMR3	180004
uIDDR_CTL	8004A05
uIDDR_MREMR1	D7000C0
uIDDR_TR0	4271CB6B
uIDDR_TR1	60D01860
uIDDR_TR2	450620
uIDDR_ZQCTL0	785A64
uIDDR_ZQCTL1	0
uIDDR_ZQCTL2	70000000

Figure 6: DMC_Registers_List_2159x_SC59x.xlsx Snapshot

Validating the DMC Interface

Once the DMC is initialized, it is important to validate it. It is recommended to check if all the DMC registers have been initialized to the correct values, if there are any basic issues with the DMC hardware interface, and if the DMC has indeed been correctly initialized by the software. The register values from the register browser can be compared with the register values in `DMC_Registers_List_2159x_SC59x.xlsx` spreadsheet.

The `Memory_Sweep_Test()` function can be used to check if all the cores 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). The `main.c` file in the `ADSP-21593/ADSP-SC594_DMCconfigGenerator_Core1` project uses these functions to validate the DMC interface. The memory sweep size used in this code is 0x800000 (8 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 tothe 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*, the preload code projects can be found at this path '*Analog Devices\CrossCore Embedded Studio 2.11.0\SHARC\ldr\init_code\SC59x_Init*' in the CCES installation directory
CCES uses the pre-built executable file (See Figure 7) to initialize the CGU and DMC before loading the actual application using the emulator.

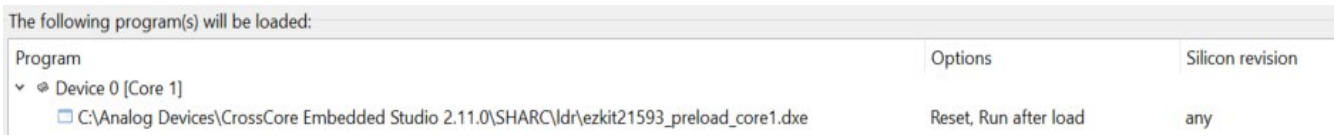

Figure 7: Preload Code for EV-21593-SOM

Initialization Code

Unlike preload code, initialization code is actually a part of the application. It is separate from the application. The DXE output is pre-pended to the DXE file of application 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 subsequently 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. The Init code projects can be found at *Analog Devices\CrossCore Embedded Studio 2.11.0\SHARC\ldr\init_code\SC59x_Init* in the CCES installation directory. The DXE output corresponding to the Init code 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 8.

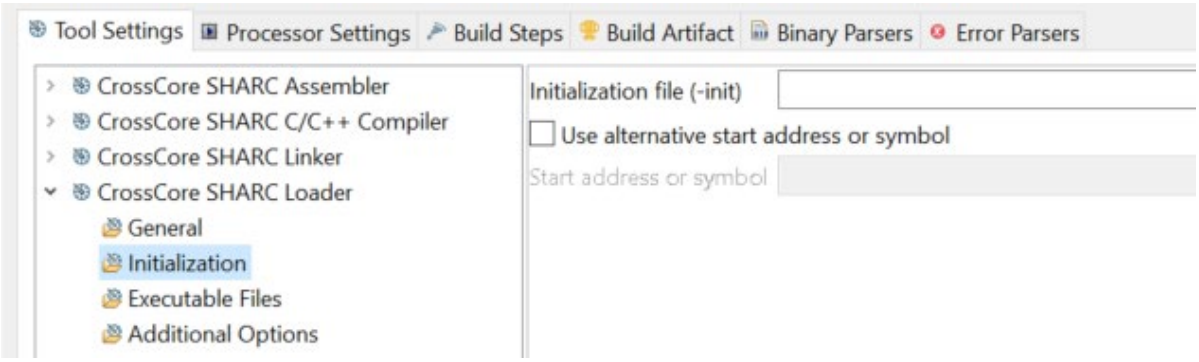

Figure 8: Initialization Code Selection in the Loader Options

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.

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:

- When using non-default CGU settings
- When using a custom board with a different memory device than the one available on the evaluation board

For example, use the following steps to modify the default preload code `sc594w_preload_Core0` project for the ADSP-SC594 processor.

1. Edit the `adi_pwr_SC594_family_1GHz_config.h` file to change the CGU and CDU configurations.
2. Edit the `adi_pwr_SC59x_config.h` file to configure the `cclkdclk_ratio` value.
3. Edit the `adi_dmc_SC594_family_800MHz_config.h` file as shown in Figure 10 using the output printed on console (shown in Figure 9) and the `adi_printDMCconfig()` API present in the `ADSP-SC594_DMCconfigGenerator_core1` project. Or, initialize the `ADI_DMC_CONFIG` structure in the `adi_dmc_SC59x_config.c` file (shown inFigure 11) using the

`DMC_Registers_List_2159x_SC59x.xlsx` spreadsheet (shown in Figure 3 and Figure 6). See the associated *zip file*[4] for the spreadsheet, and code examples. Refer to the DMC Initialization section for details.

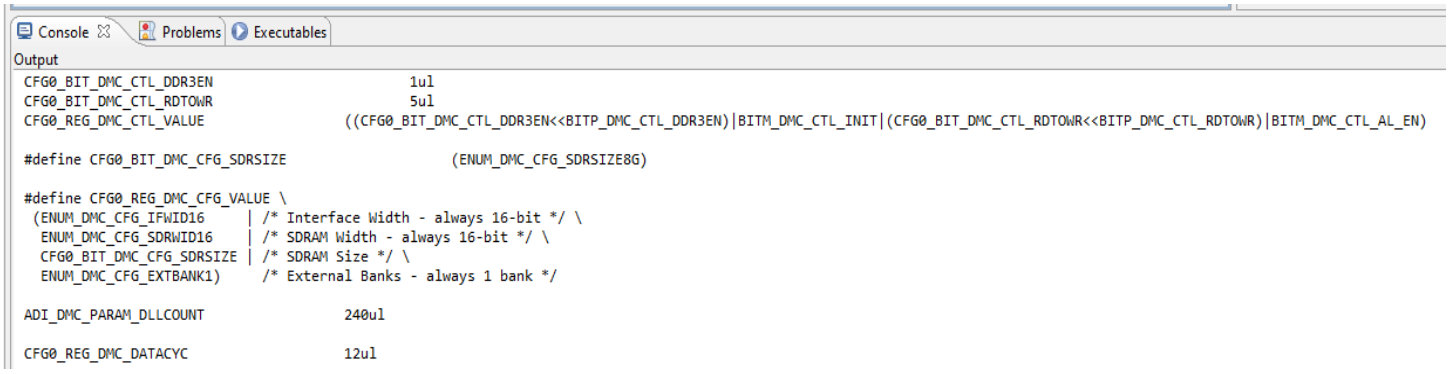

Figure 9: adi_printDMCconfig() API Snapshot

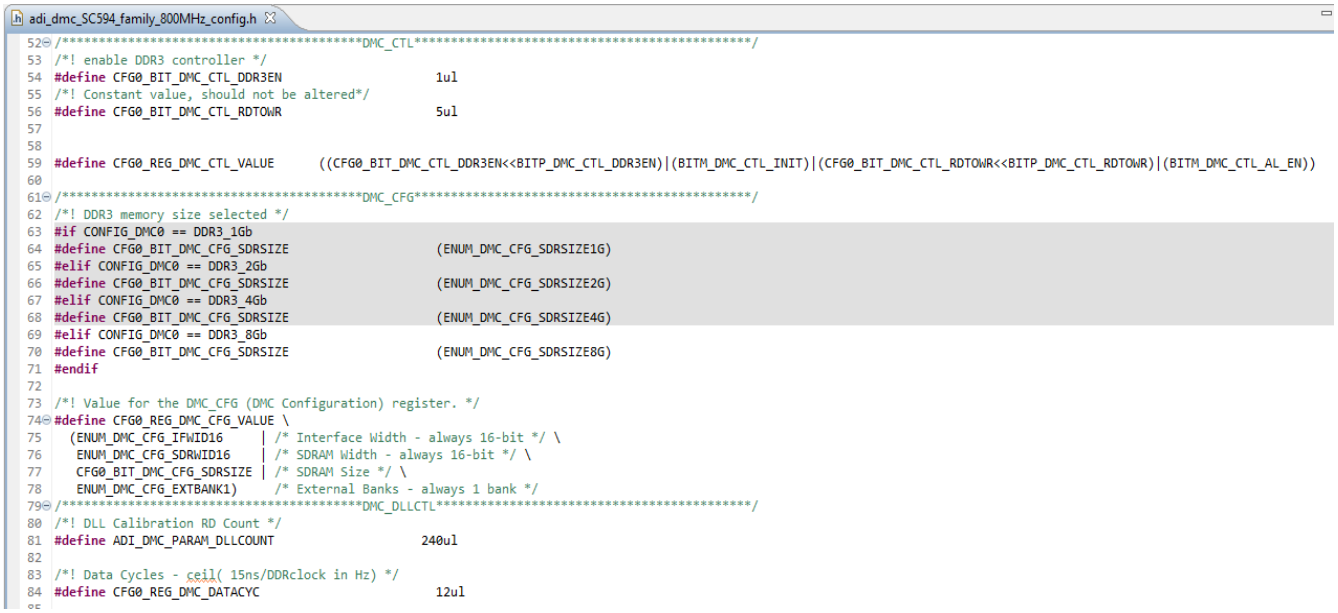

Figure 10: adi_dmc_SC594_family_800MHz_config.h Snapshot

adi_dmc_SC59x_config.c

```
44 uint32_t adi_dmc_cfg0_init(void)
45 {
46     uint32_t status = 0u;
47
48     static ADI_DMC_CONFIG config =
49     {
50             CFG0_REG_DDR_DLLCTLCFG,
51             CFG0_REG_DMC_MR2MR3,
52             CFG0_REG_DMC_CTL_VALUE,
53             CFG0_REG_DMC_MRMR1,
54             CFG0_REG_DMC_TR0_VALUE,
55             CFG0_REG_DMC_TR1_VALUE,
56             CFG0_REG_DMC_TR2_VALUE,
57             0x00785A64ul, /* 0x78 (Data/DQS ODT)
58                              0x5a (90ohms Data/DQS/DM/CLK Drive Strength)
59                              0x64 (100ohms Address/Command Drive Strength) */
60             0ul,
61             0x70000000ul
62     };
63
64     /* Initialize DMC PHY registers */
65     adi_dmc_phy_calibration(&config);
66
67     /* Initialize DMC Controller */
68     if(adi_dmc_ctrl_init(&config) != ADI_DMC_SUCCESS)
69     {
70        /* Assign error status return value */
71        status = 1u;
72     }
73
74     return status;
75 }
76
77 /*@}*/
```

Figure 11: adi_dmc_SC59x_config.c Initializing ADI_DMC_CONFIG Structure Snapshot

References

[1] *ADSP-21594/ADSP-SC591/SC592/SC594: SHARC+ Dual-Core DSP with Arm Cortex-A5 Data Sheet (Rev. A), April 2022. Analog Devices, Inc.*

[2] *ADSP-21591/21593/21594/ADSP-SC591/SC592/SC594: SHARC+ Dual-Core DSP with Arm Cortex-A5 Data Sheet (Rev. A), April 2022. Analog Devices, Inc.*

[3] *ADSP-2159x/ADSP-SC591/592/594 SHARC+ Processor Hardware Reference (Rev 0.4), April 2022. Analog Devices, Inc.*

[4] *Associated zip File (EE442v01.zip) forADSP-2159x/SC59x Programming Guidelines for Dynamic Memory Controller (EE-442). September 2022. Analog Devices, Inc.*

[5] *Silicon Anomaly List of the SHARC+®ADSP-21591/21593/21594/ADSP-SC591/ SC592/SC594 product(s), July 2022, Analog Devices, Inc.*

[6] *JESD79-3F, July 2012. JEDEC SOLID STATE TECHNOLOGY ASSOCIATION*

Document History

Revision	Description
Rev 1 – June 29, 2022 *by Deepak SH*	Initial Release.