/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \* Copyright (c) 2012 Texas Instruments Incorporated - http://www.ti.com

 \*

 \* Redistribution and use in source and binary forms, with or without

 \* modification, are permitted provided that the following conditions

 \*

 \* are met:

 \*

 \* Redistributions of source code must retain the above copyright

 \* notice, this list of conditions and the following disclaimer.

 \*

 \*

 \* Redistributions in binary form must reproduce the above copyright

 \* notice, this list of conditions and the following disclaimer in the

 \* documentation and/or other materials provided with the

 \*

 \* distribution.

 \*

 \*

 \* Neither the name of Texas Instruments Incorporated nor the names of

 \* its contributors may be used to endorse or promote products derived

 \* from this software without specific prior written permission.

 \*

 \*

 \* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS

 \* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT

 \* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR

 \* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT

 \* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,

 \* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT

 \* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,

 \* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY

 \* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT

 \* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE

 \* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

 \*

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \*

 \* Filename: evmtci6638k2k.gel

 \* Description: Utility GEL for use with the TCI6638K2K EVM. This GEL has

 \* functions that initialize the chip PLL's and the DDR3A interfaces.

 \*

 \* It also includes general utilities that are useful for EVM development

 \*

 \* More functions will be added as needed.

 \*

 \* Author: Randy Rosales

 \* Revision History:

 \*

 \* Revision 0.1

 \* - Initial revision based on the Kepler VDB GEL 32bit\_DDR-800\_Kepler\_PLL\_and\_DDR3\_Init\_Silicon\_Rev15.gel put together for Kepler bringup by Pragna Paranji

 \* - Added in GEL system clock frequency estimation functions written by WenZhong Liu to help debug core PLL instability

 \* + dspPollDSPClockFreq will estimate the clock based on a comparison with windows system clock time

 \* Revision 0.2 - pparanji

 \* - Added DDR3A memory test capabilities.

 \* Revision 0.3 - pparanji

 \* - Added the following configurations

 \* DDR3A 32bit - DDR800, DDR1066, DDR1333

 \*

 \* DSP core PLL @ 122.88 MHz to 614.4 MHz operation

 \* 122.88 MHz to 737.28 MHz operation

 \* 122.88 MHz to 983.04 MHz operation

 \* 122.88 MHz to 1.2 GHz operation

 \* 122.88 MHz to 1.2 GHz operation

 \*

 \* ARM PLL @ 100 MHz to 1.0 GHz operation

 \* 100 MHz to 1.4 GHz operation

 \* 175 MHz to 1.4 GHz operation

 \* Rev 1.2 - - pparanji

 \* ------ Updates DDR3A to single rank configurations

 \* Rev 1.3 - pparanji

 \* ------ Updated timing on DDR3A-1066 and DDR3A-1333

 \* Rev 1.4 - csmith

 \* ------ Updated PA PLL config and Tetris PLL config

 \*

 ---------------------------------------------------------------------------\*/

#define GEL\_VERSION 1.0

// Timeout definitions

int \_GEL\_Global\_Timeout1 = 0;

#define TIMEOUT\_ID 10

// Global timeout value

#define GTIMEOUT 2000

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Power definitions

#define PSC\_BASE 0x02350000

#define PSC\_PTCMD \*( unsigned int\* )( PSC\_BASE+0x120 )

#define PSC\_PTSTAT \*( unsigned int\* )( PSC\_BASE+0x128 )

#define PSC\_PDCTL\_BASE ( PSC\_BASE+0x300 )

#define PSC\_MDSTAT\_BASE ( PSC\_BASE+0x800 )

#define PSC\_MDCTL\_BASE ( PSC\_BASE+0xA00 )

// Modules on power domain 0

// Always on

// Modules on power domain 0

#define LPSC\_TSIP (4)

// Modules on power domain 1

#define LPSC\_DEBUG (5)

#define LPSC\_TETB (6)

// Modules on power domain 2

#define LPSC\_PA (7)

#define LPSC\_SGMII (8)

#define LPSC\_SA (9)

// Modules on power domain 3

#define LPSC\_PCIE (10)

// Modules on power domain 5

#define LPSC\_HYPERLINK (12)

// Modules on power domain 6

#define LPSC\_SR (13)

// Modules on power domain 7

#define LPSC\_MSMCRAM (14)

// Modules on power domain 8

#define LPSC\_C66X\_COREPAC\_0 (15)

// Modules on power domain 16

#define LPSC\_DDR3\_0 (23)

// Modules on power domain 18

#define LPSC\_PCIE\_1 (27)

// Modules on power domain 29

#define LPSC\_XGE (50)

// Modules on power domain 31

#define LPSC\_ARM (52)

// Power domains definitions

#define PD0 (0) // Power Domain-0

#define PD1 (1) // Power Domain-1

#define PD2 (2) // Power Domain-2

#define PD3 (3) // Power Domain-3

#define PD5 (5) // Power Domain-5

#define PD6 (6) // Power Domain-6

#define PD7 (7) // Power Domain-7

#define PD8 (8) // Power Domain-8

#define PD16 (16) // Power Domain-16

#define PD18 (18) // Power Domain-18

#define PD29 (29) // Power Domain-29

#define PD31 (31) // Power Domain-31

#define PSC\_SYNCRESET (0x1)

#define PSC\_DISABLE (0x2)

#define PSC\_ENABLE (0x3)

#define CHIP\_LEVEL\_REG 0x02620000

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PLL registers \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*Boot cfg registers\*/

#define KICK0 \*(unsigned int\*)(CHIP\_LEVEL\_REG + 0x0038)

#define KICK1 \*(unsigned int\*)(CHIP\_LEVEL\_REG + 0x003C)

#define KICK0\_UNLOCK (0x83E70B13)

#define KICK1\_UNLOCK (0x95A4F1E0)

#define KICK\_LOCK 0

#define TINPSEL \*(unsigned int\*)(CHIP\_LEVEL\_REG + 0x0300)

#define TOUTPSEL \*(unsigned int\*)(CHIP\_LEVEL\_REG + 0x0304)

#define MAINPLLCTL0 \*(unsigned int\*)(CHIP\_LEVEL\_REG + 0x0350)

#define MAINPLLCTL1 \*(unsigned int\*)(CHIP\_LEVEL\_REG + 0x0354)

#define MAIN\_PLLD\_OFFSET 0

#define MAIN\_PLLD\_MASK 0xFFFFFFC0

#define MAIN\_PLLM\_OFFSET 12

#define MAIN\_PLLM\_MASK 0xFFF80FFF

#define MAIN\_BWADJ0\_OFFSET 24

#define MAIN\_BWADJ0\_MASK 0x00FFFFFF

#define MAIN\_ENSAT\_OFFSET 6

#define MAIN\_ENSAT\_MASK 0xFFFFFFBF

#define MAIN\_BWADJ1\_OFFSET 0

#define MAIN\_BWADJ1\_MASK 0xFFFFFFF0

#define OBSCLKCTL \*(unsigned int\*)(CHIP\_LEVEL\_REG + 0x0C80) //TODO: Reserved in K2H datashet

/\* PA PLL Registers \*/

#define BYPASS\_BIT\_SHIFT 23

#define CLKF\_BIT\_SHIFT 6

#define CLKD\_BIT\_SHIFT 0

#define DEVSTAT (\*((unsigned int \*) 0x02620020))

#define PAPLLCTL0 \*(unsigned int\*)(CHIP\_LEVEL\_REG + 0x0358)

#define PAPLLCTL1 \*(unsigned int\*)(CHIP\_LEVEL\_REG + 0x035C)

#define PASSCLKSEL\_MASK (1 << 17) /\* Tells the configuration of the PASSCLKSEL pin \*/

#define PA\_PLL\_BYPASS\_MASK (1 << BYPASS\_BIT\_SHIFT) /\* Tells whether the PA PLL is in BYPASS mode or not \*/

#define PA\_PLL\_CLKOD\_MASK (0x00780000) /\* Tells the output divider value for the PA PLL \*/

#define PA\_PLL\_CLKF\_MASK (0x0007FFC0) /\* Tells the multiplier value for the PA PLL \*/

#define PA\_PLL\_CLKR\_MASK (0x0000003F) /\* Tells the divider value for the PA PLL \*/

#define PA\_PLL\_RESET\_MASK (0x00004000)

#define CHIP\_MISC1 \*(unsigned int\*)(CHIP\_LEVEL\_REG + 0x0C7C)

#define ARMPLL\_ENABLE\_OFFSET 13

#define DDR3APLLCTL0 \*(unsigned int\*)(CHIP\_LEVEL\_REG + 0x0360)

#define DDR3APLLCTL1 \*(unsigned int\*)(CHIP\_LEVEL\_REG + 0x0364)

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// PLL 1 definitions (DSP and ARM clock and subsystems)

#define PLL1\_BASE 0x02310000

#define PLL1\_PLLCTL (\*(unsigned int\*)(PLL1\_BASE + 0x100)) // PLL1 Control

#define PLL1\_SECCTL (\*(unsigned int\*)(PLL1\_BASE + 0x108)) // PLL1 Secondary Control

#define PLL1\_PLLM (\*(unsigned int\*)(PLL1\_BASE + 0x110)) // PLL1 Multiplier

#define PLL1\_DIV1 (\*(unsigned int\*)(PLL1\_BASE + 0x118)) // DIV1 divider

#define PLL1\_DIV2 (\*(unsigned int\*)(PLL1\_BASE + 0x11C)) // DIV2 divider

#define PLL1\_DIV3 (\*(unsigned int\*)(PLL1\_BASE + 0x120)) // DIV3 divider

#define PLL1\_CMD (\*(unsigned int\*)(PLL1\_BASE + 0x138)) // CMD control

#define PLL1\_STAT (\*(unsigned int\*)(PLL1\_BASE + 0x13C)) // STAT control

#define PLL1\_ALNCTL (\*(unsigned int\*)(PLL1\_BASE + 0x140)) // ALNCTL control

#define PLL1\_DCHANGE (\*(unsigned int\*)(PLL1\_BASE + 0x144)) // DCHANGE status

#define PLL1\_CKEN (\*(unsigned int\*)(PLL1\_BASE + 0x148)) // CKEN control

#define PLL1\_CKSTAT (\*(unsigned int\*)(PLL1\_BASE + 0x14C)) // CKSTAT status

#define PLL1\_SYSTAT (\*(unsigned int\*)(PLL1\_BASE + 0x150)) // SYSTAT status

#define PLL1\_DIV4 (\*(unsigned int\*)(PLL1\_BASE + 0x160)) // DIV4 divider

#define PLL1\_DIV5 (\*(unsigned int\*)(PLL1\_BASE + 0x164)) // DIV5 divider

#define PLL1\_DIV6 (\*(unsigned int\*)(PLL1\_BASE + 0x168)) // DIV6 divider

#define PLL1\_DIV7 (\*(unsigned int\*)(PLL1\_BASE + 0x16C)) // DIV7 divider

#define PLL1\_DIV8 (\*(unsigned int\*)(PLL1\_BASE + 0x170)) // DIV8 divider

#define PLL1\_DIV9 (\*(unsigned int\*)(PLL1\_BASE + 0x174)) // DIV9 divider

#define PLL1\_DIV10 (\*(unsigned int\*)(PLL1\_BASE + 0x178)) // DIV10 divider

#define PLL1\_DIV11 (\*(unsigned int\*)(PLL1\_BASE + 0x17C)) // DIV11 divider

#define PLL1\_DIV12 (\*(unsigned int\*)(PLL1\_BASE + 0x180)) // DIV12 divider

#define PLL1\_DIV13 (\*(unsigned int\*)(PLL1\_BASE + 0x184)) // DIV13 divider

#define PLL1\_DIV14 (\*(unsigned int\*)(PLL1\_BASE + 0x188)) // DIV14 divider

#define PLL1\_DIV15 (\*(unsigned int\*)(PLL1\_BASE + 0x18C)) // DIV15 divider

#define PLL1\_DIV16 (\*(unsigned int\*)(PLL1\_BASE + 0x190)) // DIV16 divider

#define PLLPWRDN\_OFFSET 1

#define PLLPWRDN\_MASK 0xFFFFFFFD

#define PLLRST\_OFFSET 3

#define PLLRST\_MASK 0xFFFFFFF7

#define PLLENSRC\_OFFSET 5

#define PLLENSRC\_MASK 0xFFFFFFDF

#define PLLEN\_OFFSET 0

#define PLLEN\_MASK 0xFFFFFFFE

#define OUTPUT\_DIVIDE\_OFFSET 19

#define OUTPUT\_DIVIDE\_MASK 0xFF87FFFF

#define BYPASS\_OFFSET 23

#define BYPASS\_MASK 0xFF7FFFFF

#define PLLM\_OFFSET 0

#define PLLM\_MASK 0xFFFFFFC0

#define GOSET\_OFFSET 0

#define GOSET\_MASK 0xFFFFFFFE

#define GOSTAT\_OFFSET 0

#define GOSTAT\_MASK 0xFFFFFFFE

#define OUTPUT\_DIVIDE\_OFFSET 19

#define OUTPUT\_DIVIDE\_MASK 0xFF87FFFF

// ARMPLL definitions

#define SEC\_PLLCTL0\_PLLM\_OFFSET 6

#define SEC\_PLLCTL0\_PLLM\_MASK 0xFFFF003F

#define SEC\_PLLCTL0\_BWADJ\_OFFSET 24

#define SEC\_PLLCTL0\_BWADJ\_MASK 0x00FFFFFF

#define SEC\_PLLCTL0\_OD\_OFFSET 19

#define SEC\_PLLCTL0\_OD\_MASK 0xFF87FFFF

#define SEC\_PLLCTL0\_BYPASS\_OFFSET 23

#define SEC\_PLLCTL0\_BYPASS\_MASK 0xFF7FFFFF

#define SEC\_PLLCTL1\_RESET\_OFFSET 14

#define SEC\_PLLCTL1\_RESET\_MASK 0xFFFFBFFF

#define SEC\_PLLCTL1\_PWRDWN\_OFFSET 15

#define SEC\_PLLCTL1\_PWRDWN\_MASK 0xFFFF7FFF

#define SEC\_PLLCTL1\_ENSTAT\_OFFSET 6

#define SEC\_PLLCTL1\_ENSTAT\_MASK 0xFFFFFFBF

/\*----------------DDR3A Register definition---------------------\*/

#define DDR3A\_BASE\_ADDR (0x21010000)

#define DDR3A\_STATUS (\*(int\*)(DDR3A\_BASE\_ADDR + 0x00000004))

#define DDR3A\_SDCFG (\*(int\*)(DDR3A\_BASE\_ADDR + 0x00000008))

#define DDR3A\_SDRFC (\*(int\*)(DDR3A\_BASE\_ADDR + 0x00000010))

#define DDR3A\_SDTIM1 (\*(int\*)(DDR3A\_BASE\_ADDR + 0x00000018))

#define DDR3A\_SDTIM2 (\*(int\*)(DDR3A\_BASE\_ADDR + 0x0000001C))

#define DDR3A\_SDTIM3 (\*(int\*)(DDR3A\_BASE\_ADDR + 0x00000020))

#define DDR3A\_SDTIM4 (\*(int\*)(DDR3A\_BASE\_ADDR + 0x00000028))

#define DDR3A\_ZQCFG (\*(int\*)(DDR3A\_BASE\_ADDR + 0x000000C8))

#define DDR3A\_TMPALRT (\*(int\*)(DDR3A\_BASE\_ADDR + 0x000000CC))

#define DDR3A\_DDRPHYC (\*(int\*)(DDR3A\_BASE\_ADDR + 0x000000E4))

#define DDR3A\_PHY\_CFG\_BASE (0x02329000)

#define DDR3A\_PIR (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x00000004))

#define DDR3A\_PGCR0 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x00000008))

#define DDR3A\_PGCR1 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x0000000C))

#define DDR3A\_PGCR2 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x0000008C))

#define DDR3A\_PGSR0 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x00000010))

#define DDR3A\_PGSR1 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x00000014))

#define DDR3A\_PLLCR (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x00000018))

#define DDR3A\_PTR0 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x0000001C))

#define DDR3A\_PTR1 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x00000020))

#define DDR3A\_PTR2 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x00000024))

#define DDR3A\_PTR3 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x00000028))

#define DDR3A\_PTR4 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x0000002C))

#define DDR3A\_DSGCR (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x00000040))

#define DDR3A\_DCR (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x00000044))

#define DDR3A\_MR0 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x00000054))

#define DDR3A\_MR1 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x00000058))

#define DDR3A\_MR2 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x0000005C))

#define DDR3A\_DTCR (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x00000068))

#define DDR3A\_DTPR0 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x00000048))

#define DDR3A\_DTPR1 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x0000004C))

#define DDR3A\_DTPR2 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x00000050))

#define DDR3A\_ZQ0CR1 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x00000184))

#define DDR3A\_ZQ1CR1 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x00000194))

#define DDR3A\_ZQ2CR1 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x000001A4))

#define DDR3A\_ZQ3CR1 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x000001B4))

#define DDR3A\_DATX8\_8 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x000003C0))

#define DDR3\_TEST\_START\_ADDRESS (\*(int\*)(0x80000000))

#define IODDRM\_MASK 0x00000180

#define ZCKSEL\_MASK 0x01800000

#define CL\_MASK 0x00000072

#define WR\_MASK 0x00000E00

#define BL\_MASK 0x00000003

#define RRMODE\_MASK 0x00040000

#define UDIMM\_MASK 0x20000000

#define BYTEMASK\_MASK 0x0000FC00

#define MPRDQ\_MASK 0x00000080

#define PDQ\_MASK 0x00000070

#define NOSRA\_MASK 0x08000000

#define ECC\_MASK 0x00000001

#define RRMODE\_MASK 0x00040000

#define DDR3A\_DX0GSR0 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x000001C4)) //0x71

#define DDR3A\_DX1GSR0 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x00000204)) //0x81

#define DDR3A\_DX2GSR0 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x00000244)) //0x91

#define DDR3A\_DX3GSR0 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x00000284)) //0xA1

#define DDR3A\_DX4GSR0 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x000002C4)) //0xB1

#define DDR3A\_DX5GSR0 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x00000304)) //0xC1

#define DDR3A\_DX6GSR0 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x00000344)) //0xD1

#define DDR3A\_DX7GSR0 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x00000384)) //0xE1

#define DDR3A\_DX8GSR0 (\*(int\*)(DDR3A\_PHY\_CFG\_BASE + 0x000003C4)) //0xF1

#define DDR3A\_TEST\_START\_ADDRESS (0x80000000)

#define DDR3A\_TEST\_END\_ADDRESS (DDR3A\_TEST\_START\_ADDRESS + (4 \* 100))

#define DDR3A\_BASE\_ADDRESS 0x80000000

#define TETRIS\_BASE 0x01E80000

#define TETRIS\_CPU0\_PTCMD \*(unsigned int\*)(TETRIS\_BASE + 0x0400)

#define TETRIS\_CPU0\_PDSTAT \*(unsigned int\*)(TETRIS\_BASE + 0x0404)

#define TETRIS\_CPU0\_PDCTL \*(unsigned int\*)(TETRIS\_BASE + 0x0408)

#define TETRIS\_CPU1\_PTCMD \*(unsigned int\*)(TETRIS\_BASE + 0x040C)

#define TETRIS\_CPU1\_PDSTAT \*(unsigned int\*)(TETRIS\_BASE + 0x0410)

#define TETRIS\_CPU1\_PDCTL \*(unsigned int\*)(TETRIS\_BASE + 0x0414)

#define TETRIS\_CPU2\_PTCMD \*(unsigned int\*)(TETRIS\_BASE + 0x0418)

#define TETRIS\_CPU2\_PDSTAT \*(unsigned int\*)(TETRIS\_BASE + 0x041C)

#define TETRIS\_CPU2\_PDCTL \*(unsigned int\*)(TETRIS\_BASE + 0x0420)

#define TETRIS\_CPU3\_PTCMD \*(unsigned int\*)(TETRIS\_BASE + 0x0424)

#define TETRIS\_CPU3\_PDSTAT \*(unsigned int\*)(TETRIS\_BASE + 0x0428)

#define TETRIS\_CPU3\_PDCTL \*(unsigned int\*)(TETRIS\_BASE + 0x042C)

#define SECPLLCTL0 \*(unsigned int\*)(CHIP\_LEVEL\_REG + 0x0370)

#define SECPLLCTL1 \*(unsigned int\*)(CHIP\_LEVEL\_REG + 0x0374)

unsigned int read\_val;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \*

 \* NAME

 \* OnTargetConnect

 \*

 \* PURPOSE:

 \* Setup almost everything ready for a new debug session:

 \* DSP modules and EVM board modules, at target connection.

 \* Do nothing if target is in realtime mode.

 \* This routine is called when you connect to the target board.

 \*

 \* IMPORTANT: this routine won't attempt to connect to the target

 \* if the target is not in real-time mode and that the dsp boot

 \* mode switches are not set in emulation boot mode.

 \*

 \* USAGE

 \* This routine is a callback routine and called by CCS only.

 \*

 \* RETURN VALUE

 \* NONE

 \*

 \* REFERENCE

 \*

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

OnTargetConnect()

{

 /\*------------------------------------------------------\*/

 /\* GEL\_Reset() is used to deal with the worst case \*/

 /\* senario of unknown target state. If for some reason \*/

 /\* a reset is not desired upon target connection, \*/

 /\* GEL\_Reset() may be removed and replaced with \*/

 /\* something "less brutal" like a cache initialization \*/

 /\* function. \*/

 /\*------------------------------------------------------\*/

 //GEL\_Reset();

 //xmc\_setup();

 //ddr3\_setup();

 GEL\_TextOut("\nConnecting Target...\n");

 // Check if target is not in real-time mode. If it is in stop mode,

 // initialize everything. In real-time mode, do nothing to connect

 // unobtrusively...

 if (!GEL\_IsInRealtimeMode())

 {

 // Validates if emulation boot mode

 if (DEVSTAT & 0x0000000E)

 {

 GEL\_TextOut("No initialization performed since bootmode = %x \n",,,,,(DEVSTAT >> 1 ) & 0xF);

 GEL\_TextOut("You can manually initialize with GlobalDefaultSetup\n");

 }

 else

 {

 // Comment the following line at production application test

 // when the application need to initialize everything, but not the

 // GEL file.

 Global\_Default\_Setup\_Silent();

 }

 } else {

 GEL\_TextOut("No initialization performed in real time mode\n");

 }

}

/\*--------------------------------------------------------------\*/

/\* OnReset() \*/

/\* This function is called by CCS when you do Debug->Resest. \*/

/\* The goal is to put the C6x into a known good state with \*/

/\* respect to cache, edma and interrupts. \*/

/\*--------------------------------------------------------------\*/

OnReset( int nErrorCode )

{

}

/\*--------------------------------------------------------------\*/

/\* xmc\_setup() \*/

/\* XMC MPAX register setting to access DDR3 config space \*/

/\*--------------------------------------------------------------\*/

#define XMC\_BASE\_ADDR (0x08000000)

#define XMPAX2\_L (\*(int\*)(XMC\_BASE\_ADDR + 0x00000010))

#define XMPAX2\_H (\*(int\*)(XMC\_BASE\_ADDR + 0x00000014))

xmc\_setup()

{

 /\* mapping for ddr emif registers XMPAX\*2 \*/

 XMPAX2\_L = 0x121010FF; /\* replacement addr + perm \*/

 XMPAX2\_H = 0x2101000B; /\* base addr + seg size (64KB)\*/ //"1B"-->"B" by xj

 GEL\_TextOut("XMC setup complete.\n");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \*

 \* NAME

 \* Global\_Default\_Setup\_Silent

 \*

 \* PURPOSE:

 \* Setup almost everything ready for a new debug session:

 \* DSP modules and EVM board modules.

 \*

 \* USAGE

 \* This routine can be called as:

 \*

 \* Global\_Default\_Setup\_Silent()

 \*

 \* RETURN VALUE

 \* NONE

 \*

 \* REFERENCE

 \*

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Global\_Default\_Setup\_Silent()

{

 float gel\_ver = GEL\_VERSION;

 // Set DSP cache to pre defined values...

 GEL\_TextOut( "TCI6636K2E GEL file Ver is %f \n",,,,, (float) (gel\_ver/1.0));

 //Set\_DSP\_Cache();

 // Only core 0 can set these

 if (DNUM == 0)

 {

 // Setup main PLL DSP @ 1 GHz

 Set\_Pll1(3); // call Set\_Pll1 with index = 3 -> 100 MHz to 1 GHz operation

 // Setup all Power Domains on

 Set\_Psc\_All\_On();

 // Setup PA PLL

 PaPllConfig();

 GEL\_TextOut("DDR begin\n");

 xmc\_setup();

 ddr3A\_64bit\_DDR1600\_setup();

 GEL\_TextOut("DDR done\n");

 }

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \*

 \* NAME

 \* Set\_PSC\_State

 \*

 \* PURPOSE:

 \* Set a new power state for the specified domain id in a power controler

 \* domain. Wait for the power transition to complete.

 \*

 \* USAGE

 \* This routine can be called as:

 \*

 \* Set\_PSC\_State(unsigned int pd,unsigned int id,unsigned int state)

 \*

 \* pd - (i) power domain.

 \*

 \* id - (i) module id to use for module in the specified power domain

 \*

 \* state - (i) new state value to set

 \* 0 = RESET

 \* 1 = SYNC RESET

 \* 2 = DISABLE

 \* 3 = ENABLE

 \*

 \* RETURN VALUE

 \* 0 if ok, !=0 for error

 \*

 \* REFERENCE

 \*

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Set\_PSC\_State(unsigned int pd,unsigned int id,unsigned int state)

{

 unsigned int\* mdctl;

 unsigned int\* mdstat;

 unsigned int\* pdctl;

 int ret=0;

 // Only core0 can set PSC

 if (DNUM == 0)

 {

 mdctl = ( unsigned int\* )(PSC\_MDCTL\_BASE + ( 4 \* id ));

 mdstat = ( unsigned int\* )( PSC\_MDSTAT\_BASE + ( 4 \* id ));

 pdctl = ( unsigned int\* )(PSC\_PDCTL\_BASE + ( 4 \* pd ));

 // If state is already set, do nothing

 if ( ( \*mdstat & 0x1f ) == state )

 {

 return(0);

 }

 // Wait for GOSTAT to clear

 Set\_Timeout(GTIMEOUT);

 while( Get\_Timeout() && (PSC\_PTSTAT & (0x1 << pd)) != 0 );

 // Check if we got timeout error while waiting

 if (!Get\_Timeout())

 {

 GEL\_TextOut( "Set\_PSC\_State... Timeout Error #01 pd=%d, md=%d!\n",,2,,,pd,id);

 ret=1;

 }

 else

 {

 // Set power domain control

 \*pdctl = (\*pdctl) | 0x00000001;

 // Set MDCTL NEXT to new state

 \*mdctl = ((\*mdctl) & ~(0x1f)) | state;

 // Start power transition by setting PTCMD GO to 1

 PSC\_PTCMD = (PSC\_PTCMD) | (0x1<<pd);

 // Wait for PTSTAT GOSTAT to clear

 Set\_Timeout(GTIMEOUT);

 while( Get\_Timeout() && (PSC\_PTSTAT & (0x1 << pd)) != 0 );

 // Check if we got timeout error while waiting

 if (!Get\_Timeout())

 {

 GEL\_TextOut( "Set\_PSC\_State... Timeout Error #02 pd=%d, md=%d!\n",,2,,,pd,id);

 ret=2;

 }

 else

 {

 // Verify state changed

 Set\_Timeout(GTIMEOUT);

 while(Get\_Timeout() && ( \*mdstat & 0x1f ) != state );

 // Check if we got timeout error while waiting

 if (!Get\_Timeout())

 {

 GEL\_TextOut( "Set\_PSC\_State... Timeout Error #03 pd=%d, md=%d!\n",,2,,,pd,id);

 ret=3;

 }

 }

 }

 // Kill the currently running timeout

 Kill\_Timeout();

 }

 else

 {

 GEL\_TextOut("DSP core #%d cannot set PSC.\n",,2,,,DNUM);

 }

 return(ret);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \*

 \* NAME

 \* Set\_Timeout

 \*

 \* PURPOSE:

 \* Starts a timeout period of msec. The running timeout period can be

 \* query with Get\_Timeout. To kill a running timeout before the end,

 \* call Kill\_Timeout. Only one timeout period can be used at any time.

 \* A timeout period can be used to measure a period of time while doing

 \* anything else. Not accurate, sets timer at least as big as desired.

 \*

 \* USAGE

 \* This routine can be called as:

 \*

 \* Set\_Timeout(msec)

 \*

 \* msec - (i) timeout period in msec (not very precise < sec range)

 \*

 \* RETURN VALUE

 \* NONE

 \*

 \* REFERENCE

 \*

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Set\_Timeout(msec)

{

 // Cancel the current timer if not already expired

 GEL\_CancelTimer(TIMEOUT\_ID);

 // Starts the timeout period

 \_GEL\_Global\_Timeout1=1;

 // Setup a callback routine with specified timeout

 GEL\_SetTimer(msec, TIMEOUT\_ID, "\_Timeout\_Callback()");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \*

 \* NAME

 \* Get\_Timeout

 \*

 \* PURPOSE:

 \* Query the running state of a timeout period started by Set\_Timeout.

 \* (see Set\_Timeout for more info).

 \*

 \* USAGE

 \* This routine can be called as:

 \*

 \* Get\_Timeout()

 \*

 \* RETURN VALUE

 \* 0:expired, 1:running

 \*

 \* REFERENCE

 \*

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Get\_Timeout()

{

 if (!\_GEL\_Global\_Timeout1)

 {

 // Cancel the current timer

 GEL\_CancelTimer(TIMEOUT\_ID);

 }

 // Return the global timeout status 1=running, 0=expired

 return \_GEL\_Global\_Timeout1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \*

 \* NAME

 \* Kill\_Timeout

 \*

 \* PURPOSE:

 \* Cancel a running timeout period before it expires

 \* (see Set\_Timeout for more info).

 \*

 \* USAGE

 \* This routine can be called as:

 \*

 \* Kill\_Timeout()

 \*

 \* RETURN VALUE

 \* NONE

 \*

 \* REFERENCE

 \*

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Kill\_Timeout()

{

 // Cancel the current timer

 GEL\_CancelTimer(TIMEOUT\_ID);

 // The timeout period is expired

 \_GEL\_Global\_Timeout1=0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \*

 \* NAME

 \* \_Timeout\_Callback

 \*

 \* PURPOSE:

 \* Internal Callback function used by Set\_timeout

 \* (see Set\_Timeout for more info).

 \*

 \* USAGE

 \* This routine must not be called by itself.

 \*

 \* RETURN VALUE

 \* NONE

 \*

 \* REFERENCE

 \*

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

\_Timeout\_Callback()

{

 // The timeout period is expired

 \_GEL\_Global\_Timeout1=0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \*

 \* NAME

 \* Set\_Psc\_All\_On

 \*

 \* PURPOSE:

 \* Enable all PSC modules and DSP power domains on ALWAYSON, and wait

 \* for these power transitions to complete.

 \*

 \* USAGE

 \* This routine can be called as:

 \*

 \* Set\_Psc\_All\_On()

 \*

 \* RETURN VALUE

 \* NONE

 \*

 \* REFERENCE

 \*

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

hotmenu Set\_Psc\_All\_On( )

{

 unsigned int i=0;

 // Only core0 can set PSC

 if (DNUM == 0)

 {

 GEL\_TextOut( "Power on all PSC modules and DSP domains... \n");

 Set\_PSC\_State(PD0, LPSC\_TSIP, PSC\_ENABLE);

 Set\_PSC\_State(PD1, LPSC\_DEBUG, PSC\_ENABLE);

 Set\_PSC\_State(PD1, LPSC\_TETB, PSC\_ENABLE);

 Set\_PSC\_State(PD2, LPSC\_PA, PSC\_ENABLE);

 Set\_PSC\_State(PD2, LPSC\_SGMII, PSC\_ENABLE);

 Set\_PSC\_State(PD2, LPSC\_SA, PSC\_ENABLE);

 Set\_PSC\_State(PD3, LPSC\_PCIE, PSC\_ENABLE);

 Set\_PSC\_State(PD5, LPSC\_HYPERLINK, PSC\_ENABLE);

 Set\_PSC\_State(PD6, LPSC\_SR, PSC\_ENABLE);

 Set\_PSC\_State(PD7, LPSC\_MSMCRAM, PSC\_ENABLE);

 Set\_PSC\_State(PD8, LPSC\_C66X\_COREPAC\_0, PSC\_ENABLE);

 Set\_PSC\_State(PD16, LPSC\_DDR3\_0, PSC\_ENABLE);

 Set\_PSC\_State(PD18, LPSC\_PCIE\_1, PSC\_ENABLE);

 Set\_PSC\_State(PD29, LPSC\_XGE, PSC\_ENABLE);

 Set\_PSC\_State(PD31, LPSC\_ARM, PSC\_ENABLE);

 GEL\_TextOut( "Power on all PSC modules and DSP domains... Done.\n" );

 }

 else

 {

 GEL\_TextOut("DSP core #%d cannot set PSC.\n",,2,,,DNUM);

 }

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*

 Set\_Pll1() - This function executes the main PLL initialization

 sequence needed to get the main PLL up after coming out of an initial power up

 before it is locked or after it is already locked.

 Index value determines multiplier, divier used and clock reference assumed for

 output display.

 \*/

Set\_Pll1(int index)

{

 int i, TEMP;

 unsigned int BYPASS\_val;

 unsigned int BWADJ\_val;

 unsigned int OD\_val;

 float CLKIN\_val;

 unsigned int PLLM\_val;

 unsigned int PLLD\_val;

 unsigned int PLLDIV3\_val; //example value for SYSCLK2 (from 6614 spec) Default /2 - Fast Peripherals, (L2, MSMC, DDR3 EMIF, EDMA0...)

 unsigned int PLLDIV4\_val; //example value for SYSCLK3 (from 6614 spec) Default /3 - Switch Fabric

 unsigned int PLLDIV7\_val; //example value for SYSCLK6 (from 6614 spec) Defualt /6 - Slow Peripherals (UART, SPI, I2C, GPIO...)

 unsigned int debug\_info\_on;

 unsigned int delay;

 if(index == 1){ // 100 MHz -> 614.28 MHz

 CLKIN\_val = 100; // setup CLKIN to 614.28 MHz

 PLLM\_val = 43; // setup PLLM (PLL multiplier) to x43

 PLLD\_val = 1; // setup PLLD (reference divider) to /1

 OD\_val = 7; // setup OD to /7

 }

 else if(index == 2){ // 100MHz -> 737.5 MHz

 CLKIN\_val = 100; // setup CLKIN to 100 MHz

 PLLM\_val = 59; // setup PLLM (PLL multiplier) to x59

 PLLD\_val = 1; // setup PLLD (reference divider) to /1

 OD\_val = 8; // setup OD to /8

 }

 else if(index == 3){ // 100MHz -> 1 GHz

 CLKIN\_val = 100; // setup CLKIN to 100 MHz

 PLLM\_val = 20; // setup PLLM (PLL multiplier) to x20

 PLLD\_val = 1; // setup PLLD (reference divider) to /1

 OD\_val = 2; // setup OD to /2

 }

 else if(index == 4){ // 100 MHz -> 1.2 GHz

 CLKIN\_val = 100; // setup CLKIN to 100 MHz

 PLLM\_val = 24; // setup PLLM (PLL multiplier) to x24

 PLLD\_val = 1; // setup PLLD (reference divider) to /1

 OD\_val = 2; // setup OD to /2

 }

 else if(index == 5){ // 100 MHz -> 1.35 GHz

 CLKIN\_val = 100; // setup CLKIN to 100 MHz

 PLLM\_val = 27; // setup PLLM (PLL multiplier) to x27

 PLLD\_val = 1; // setup PLLD (reference divider) to /1

 OD\_val = 2; // setup OD to /2

 }

 PLLDIV3\_val = 3; // setup PLL output divider 3 to /3

 PLLDIV4\_val = 5; // setup PLL output divider 4 to /3

 PLLDIV7\_val = 6; // setup PLL output divider 7 to /6

 BYPASS\_val = PLL1\_SECCTL & ~BYPASS\_MASK; // get value of the BYPASS field

 BWADJ\_val = (PLLM\_val) >> 1; // setup BWADJ to be 1/2 the value of PLLM

 //OD\_val = 2; // setup OD to a fixed /2

 debug\_info\_on = 1;

 delay = 1000; // fix this!

 /\* Step 1: Unlock Boot Config Registers \*/

 KICK0 = KICK0\_UNLOCK;

 KICK1 = KICK1\_UNLOCK;

 /\* Step 2: Check if SECCTL bypass is low or high indicating what state the Main PLL is currently in. if

 the Main PLL is in bypass still (not yet setup) execute the following steps. \*/

 if(BYPASS\_val != 0x00000000){ // PLL bypass enabled - Execute PLL setup for PLL fresh out of power on reset

 if(debug\_info\_on){

 GEL\_TextOut("Detected PLL bypass enabled: SECCTL[BYPASS] = %x\n",,,,, BYPASS\_val);

 }

 /\* Step 2a: Set MAINPLLCTL1[ENSAT] = 1 - This enables proper biasing of PLL analog circuitry \*/

 MAINPLLCTL1 |= (1 << MAIN\_ENSAT\_OFFSET);

 if(debug\_info\_on){

 GEL\_TextOut("(2a) MAINPLLCTL1 = %x\n",,,,, MAINPLLCTL1);

 }

 /\* Step 2b: Set PLLCTL[PLLEN] = 0 This enables bypass in PLL controller MUX \*/

 PLL1\_PLLCTL &= ~(1 << PLLEN\_OFFSET);

 if(debug\_info\_on){

 GEL\_TextOut("(2b) PLLCTL = %x\n",,,,, PLL1\_PLLCTL);

 }

 /\* Step 2c: Set PLLCTL[PLLENSRC] = 0 - This enables PLLEN to control PLL controller MUX \*/

 PLL1\_PLLCTL &= ~(1 << PLLENSRC\_OFFSET);

 if(debug\_info\_on){

 GEL\_TextOut("(2c) PLLCTL = %x\n",,,,, PLL1\_PLLCTL);

 }

 /\* Step 2d: Wait 4 reference clock cycles (slowest of ALTCORE or SYSCLK) to make sure

 that the PLL controller MUX switches properly to bypass. \*/

 if(debug\_info\_on){

 GEL\_TextOut("(2d) Delay...\n",,,,,);

 }

 for(i = 0; i < delay; i++); // this delay is much more than required

 /\* Step 2e: Set SECCTL[BYPASS] = 1 - enables bypass in PLL MUX \*/

 PLL1\_SECCTL |= (1 << BYPASS\_OFFSET);

 if(debug\_info\_on){

 GEL\_TextOut("(2e) SECCTL = %x\n",,,,, PLL1\_SECCTL);

 }

 /\* Step 2f: Set PLLCTL[PLLPWRDN] = 1 - power down the PLL \*/

 PLL1\_PLLCTL |= (1 << PLLPWRDN\_OFFSET);

 if(debug\_info\_on){

 GEL\_TextOut("(2f) PLLCTL = %x\n",,,,, PLL1\_PLLCTL);

 }

 /\* Step 2g: Wait for at least 5us for the PLL to power down \*/

 if(debug\_info\_on){

 GEL\_TextOut("(2g) Delay...\n",,,,,);

 }

 for(i = 0; i < delay; i++); // this delay is much more than required

 /\* Step 2h: Set PLLCTL[PLLPWRDN] = 0 - Power the PLL back up \*/

 PLL1\_PLLCTL &= ~(1 << PLLPWRDN\_OFFSET);

 if(debug\_info\_on){

 GEL\_TextOut("(2h) PLLCTL = %x\n",,,,, PLL1\_PLLCTL);

 }

 }

 else{ // PLL bypass disabled - Execute PLL setup for PLL that has previously been locked (skip to Step 3)

 if(debug\_info\_on){

 GEL\_TextOut("Detected PLL bypass disabled: SECCTL[BYPASS] = %x\n",,,,, BYPASS\_val);

 }

 /\* Step 3a: Set PLLCTL[PLLEN] = 0 This enables bypass in PLL controller MUX \*/

 PLL1\_PLLCTL &= ~(1 << PLLEN\_OFFSET);

 if(debug\_info\_on){

 GEL\_TextOut("(3a) PLLCTL = %x\n",,,,, PLL1\_PLLCTL);

 }

 /\* Step 3b: Set PLLCTL[PLLENSRC] = 0 - This enables PLLEN to control PLL controller MUX \*/

 PLL1\_PLLCTL &= ~(1 << PLLENSRC\_OFFSET);

 if(debug\_info\_on){

 GEL\_TextOut("(3b) PLLCTL = %x\n",,,,, PLL1\_PLLCTL);

 }

 /\* Step 3c: Wait 4 reference clock cycles (slowest of ALTCORE or SYSCLK) to make sure

 that the PLL controller MUX switches properly to bypass. \*/

 if(debug\_info\_on){

 GEL\_TextOut("(3c) Delay...\n",,,,,);

 }

 for(i = 0; i < delay; i++); // this delay is much more than required

 }

 /\* Step 4: Programming PLLM[5:0] in the PLLM register of the PLL controller and

 programming PLLM[12:6] in the MAINPLLCTL0 register \*/

 PLL1\_PLLM &= PLLM\_MASK; // clear the PLLM[5:0] bit field

 PLL1\_PLLM |= ~PLLM\_MASK & (PLLM\_val - 1); // set the PLLM[5:0] bit field to the 6 LSB of PLLM\_val

 if(debug\_info\_on){

 GEL\_TextOut("(4)PLLM[PLLM] = %x\n",,,,, PLL1\_PLLM);

 }

 MAINPLLCTL0 &= MAIN\_PLLM\_MASK; // clear the PLLM[12:6] bit field

 MAINPLLCTL0 |= ~MAIN\_PLLM\_MASK & (( (PLLM\_val - 1) >> 6) << MAIN\_PLLM\_OFFSET); // set the PLLM[12:6] bit field to the 7 MSB of PLL\_val

 if(debug\_info\_on){

 GEL\_TextOut("MAINPLLCTL0 = %x\n",,,,, MAINPLLCTL0);

 }

 /\* Step 5: Programming BWADJ[7:0] in the MAINPLLCTL0 register and BWADJ[11:8] in MAINPLLCTL1 register \*/

 MAINPLLCTL0 &= MAIN\_BWADJ0\_MASK; // clear the MAIN\_BWADJ0 bit field

 MAINPLLCTL0 |= ~MAIN\_BWADJ0\_MASK & ((BWADJ\_val - 1) << MAIN\_BWADJ0\_OFFSET); // set the MAIN\_BWADJ[7:0] bit field to the 8 LSB of BWADJ\_val

 if(debug\_info\_on){

 GEL\_TextOut("(5) MAINPLLCTL0 = %x\n",,,,, MAINPLLCTL0);

 }

 MAINPLLCTL1 &= MAIN\_BWADJ1\_MASK; // clear the MAIN\_BWADJ1 bit field

 MAINPLLCTL1 |= ~MAIN\_BWADJ1\_MASK & (( (BWADJ\_val - 1) >> 8) << MAIN\_BWADJ1\_OFFSET); // set the MAIN\_BWADJ[11:8] bit field to the 4 MSB of BWADJ\_val

 if(debug\_info\_on){

 GEL\_TextOut("(5) MAINPLLCTL1 = %x\n",,,,, MAINPLLCTL1);

 }

 /\* Step 6: Programming PLLD[5:0] in the MAINPLLCTL0 register \*/

 MAINPLLCTL0 &= MAIN\_PLLD\_MASK; // clear the PLLD bit field

 MAINPLLCTL0 |= ~MAIN\_PLLD\_MASK & (PLLD\_val - 1); // set the PLLD[5:0] bit field of PLLD to PLLD\_val

 if(debug\_info\_on){

 GEL\_TextOut("(6) MAINPLLCTL0 = %x\n",,,,, MAINPLLCTL0);

 }

 /\* Step 7: Programming OD[3:0] in the SECCTL register \*/

 PLL1\_SECCTL &= OUTPUT\_DIVIDE\_MASK; // clear the OD bit field

 PLL1\_SECCTL |= ~OUTPUT\_DIVIDE\_MASK & (OD\_val - 1) << OUTPUT\_DIVIDE\_OFFSET; // set the OD[3:0] bit field of PLLD to OD\_val

 if(debug\_info\_on){

 GEL\_TextOut("(7) SECCTL = %x\n",,,,, PLL1\_SECCTL);

 }

 /\* Step 8: Following steps are needed to change the default output dividers \*/

 /\* Step 8a: Check that the GOSTAT bit in PLLSTAT is cleared to show that no GO

 operation is currently in progress\*/

 if(debug\_info\_on){

 GEL\_TextOut("(8a) Delay...\n",,,,,);

 }

 while((PLL1\_STAT) & 0x00000001);

 /\* Step 8b: Program the RATIO field in PLLDIVn to the desired new divide-down rate.

 If RATIO field is changed, the PLL controller will flag the change in the

 corresponding bit of DCHANGE\*/

 PLL1\_DIV3 = (PLLDIV3\_val-1) | 0x8000; //Set PLLDIV3

 PLL1\_DIV4 = (PLLDIV4\_val-1) | 0x8000; //Set PLLDIV4

 PLL1\_DIV7 = (PLLDIV7\_val-1) | 0x8000; //Set PLLDIV7

 if(debug\_info\_on){

 GEL\_TextOut("PLL1\_DIV3 = %x\n",,,,, PLL1\_DIV3);

 GEL\_TextOut("PLL1\_DIV4 = %x\n",,,,, PLL1\_DIV4);

 GEL\_TextOut("PLL1\_DIV7 = %x\n",,,,, PLL1\_DIV7);

 }

 /\* Step 8c: Set GOSET bit in PLLCMD to initiate the GO operation to change the divide

 values and align the SYSCLKs as programmed \*/

 PLL1\_CMD |= 0x00000001;

 /\*Step 8d/e: Read the GOSTAT bit in PLLSTAT to make sure the bit returns to 0 to

 indicate that the GO operation has completed \*/

 if(debug\_info\_on){

 GEL\_TextOut("(8d/e) Delay...\n",,,,,);

 }

 while((PLL1\_STAT) & 0x00000001);

 /\* Step 9: Set PLLCTL[PLLRST] = 1 - Assert PLL reset (Previously Step 3)\*/

 PLL1\_PLLCTL |= (1 << PLLRST\_OFFSET);

 /\* Step 10: Wait for the at least 7us for the PLL reset properly (128 CLKIN1 cycles) \*/

 if(debug\_info\_on){

 GEL\_TextOut("(10) Delay...\n",,,,,);

 }

 for(i=0;i<delay;i++);

 /\* Step 11: Set PLLCTL[PLLRST] = 0 - De-Assert PLL reset \*/

 PLL1\_PLLCTL &= ~(1 << PLLRST\_OFFSET);

 /\* Step 12: Wait for PLL to lock (2000 CLKIN1 cycles) \*/

 if(debug\_info\_on){

 GEL\_TextOut("(12) Delay...\n",,,,,);

 }

 for(i=0;i<delay;i++);

 /\* Step 13: In SECCTL, write BYPASS = 0 (enable PLL mux to switch to PLL mode) \*/

 PLL1\_SECCTL &= ~(1 << BYPASS\_OFFSET);

 if(debug\_info\_on){

 GEL\_TextOut("(13) SECCTL = %x\n",,,,, PLL1\_SECCTL);

 }

 if(debug\_info\_on){

 GEL\_TextOut("(Delay...\n",,,,,);

 }

 for(i=0;i<delay;i++);

 if(debug\_info\_on){

 GEL\_TextOut("(Delay...\n",,,,,);

 }

 for(i=0;i<delay;i++);

 /\* Step 14: In PLLCTL, write PLLEN = 1 to enable PLL mode \*/

 PLL1\_PLLCTL |= (1 << PLLEN\_OFFSET);

 if(debug\_info\_on){

 GEL\_TextOut("(14) PLLCTL = %x\n",,,,, PLL1\_PLLCTL);

 }

 /\* Step 15: Lock Boot Config Registers \*/

 KICK0 = 0x00000000;

 KICK1 = 0x00000000;

 GEL\_TextOut("PLL has been configured (CLKIN \* PLLM / PLLD / PLLOD = PLLOUT):\n",,,,,);

 GEL\_TextOut("PLL has been configured (%f MHz \* %d / %d / %d = %f MHz)\n",,,,, CLKIN\_val, PLLM\_val, PLLD\_val,OD\_val,(CLKIN\_val \* PLLM\_val / PLLD\_val / OD\_val) );

}

//TODO: May need to change, as currently information is not available

Set\_Tetris\_Pll(int index)

{

 unsigned int BWADJ\_val;

 unsigned int OD\_val;

 unsigned int PLLM\_val;

 float CLKIN\_val;

 int i;

 GEL\_TextOut("Switching on ARM Core 0\n",,,,,);

 TETRIS\_CPU0\_PDCTL = 0x00000000;

 TETRIS\_CPU0\_PTCMD = 0x00000001;

 GEL\_TextOut("Switching on ARM Core 1\n",,,,,);

 TETRIS\_CPU1\_PDCTL = 0x00000000;

 TETRIS\_CPU1\_PTCMD = 0x00000001;

 GEL\_TextOut("Switching on ARM Core 2\n",,,,,);

 TETRIS\_CPU2\_PDCTL = 0x00000000;

 TETRIS\_CPU2\_PTCMD = 0x00000001;

 GEL\_TextOut("Switching on ARM Core 3\n",,,,,);

 TETRIS\_CPU3\_PDCTL = 0x00000000;

 TETRIS\_CPU3\_PTCMD = 0x00000001;

 if(index == 1){ // 100 MHz -> 1.0 GHz

 CLKIN\_val = 125; // setup CLKIN to 125 MHz

 PLLM\_val = 16; // setup PLLM (PLL multiplier) to x20

 OD\_val = 2; // setup OD to a fixed /2

 }

 else if(index == 2){ // 100 MHz -> 1.4 GHz

 CLKIN\_val = 125; // setup CLKIN to 125 MHz

 PLLM\_val = 22; // setup PLLM (PLL multiplier) to x28

 OD\_val = 2; // setup OD to a fixed /2

 }

 else if(index == 3){ // 174.825MHz -> 1.4 GHz //TODO: From where this freq will be available? Not shown in schematic.

 CLKIN\_val = 174.825; // setup CLKIN to 174.825 MHz

 PLLM\_val = 16; // setup PLLM (PLL multiplier) to x16

 OD\_val = 2; // setup OD to a fixed /2

 }

 BWADJ\_val = (PLLM\_val-1) >> 1; // setup BWADJ to be 1/2 the value of PLLM

 OD\_val = 2; // setup OD to a fixed /2

 /\* Step 1: Unlock Boot Config Registers \*/

 KICK0 = KICK0\_UNLOCK;

 KICK1 = KICK1\_UNLOCK;

 //Step 1 : Assert SEC PLL Reset

 SECPLLCTL1 = ((1 << SEC\_PLLCTL1\_RESET\_OFFSET) | (1 << SEC\_PLLCTL1\_ENSTAT\_OFFSET));

 //Step 2 : Change CLKF/OD/BWADJ etc. for SEC PLL

 SECPLLCTL0 = ((BWADJ\_val << SEC\_PLLCTL0\_BWADJ\_OFFSET) |

 ((OD\_val-1) << SEC\_PLLCTL0\_OD\_OFFSET)|

 ((PLLM\_val-1) << SEC\_PLLCTL0\_PLLM\_OFFSET));

 //Step 3 : Make sure the resets are held for 5us

 for(i = 0; i < 200000; i++);

 //Step 4 : Remove SEC PLL reset

 SECPLLCTL1 = (1 << SEC\_PLLCTL1\_ENSTAT\_OFFSET);

 //Step 5 : Wait for PLL to lock (4000 CLKIN1 cycles)

 for(i = 0; i < 4000; i++);

 //Step 6 : Get the PLL out of Bypass

 //SECPLLCTL0 &= ~(1 << SEC\_PLLCTL0\_BYPASS\_OFFSET);

 CHIP\_MISC1 |= (1 << ARMPLL\_ENABLE\_OFFSET);

 //Step 6 : Lock Boot Config Registers

 KICK0 = 0x00000000;

 KICK1 = 0x00000000;

 GEL\_TextOut("ARM PLL has been configured (%f MHz \* %d / %d = %f MHz)\n",,,,, CLKIN\_val, PLLM\_val, OD\_val, (CLKIN\_val \* PLLM\_val)/OD\_val);

}

/\* Set the desired PA PLL configuration \*/

PaPllConfig()

{

 unsigned int passclksel = (DEVSTAT & PASSCLKSEL\_MASK);

 unsigned int papllctl0val\_orig = PAPLLCTL0;

 unsigned int papllctl1val\_orig = PAPLLCTL1;

 unsigned int papllctl0val\_final;

 unsigned int papllctl1val\_final;

 unsigned int papllclkf = 19; //204; // 204; 20 (if PASSREFCLK == 100mhz) Multiply by clkf + 1

 unsigned int papllclkd = 0; //11; // 11; 1 (if PASSREFCLK == 100mhz) Divide by clkd + 1

 unsigned int i = 0;

 if (passclksel != PASSCLKSEL\_MASK) GEL\_TextOut("WARNING: SYSCLK is the input to the PA PLL.\n");

 /\* Unlock Chip Level Registers \*/

 KICK0 = KICK0\_UNLOCK;

 KICK1 = KICK1\_UNLOCK;

 // PAPLLCTL1 = PAPLLCTL1 | 0x00000040; //Set ENSAT = 1; Set PLL Select to 0 (for SYCLK0 as input of PASS)

 PAPLLCTL1 = PAPLLCTL1 | 0x00002040; //Set ENSAT = 1; Set PLL Select to 1 (for PA PLL as input of PASS)

 /\*in PAPLLCTL0, clear bypass bit to set the PA PLL in Bypass Mode\*/

 //PAPLLCTL0 &= ~(1<<BYPASS\_BIT\_SHIFT); // Not setting Bypass bit

 PAPLLCTL0 |= (1<<BYPASS\_BIT\_SHIFT); // Actually setting bypass bit

 /\*Wait 4 cycles for the slowest of PLLOUT or reference clock source (CLKIN)\*/

 for(i=0;i<100;i++);

 /\*In PAPLLCTL1, write PLL reset bit to put the PLL in reset\*/

 PAPLLCTL1 |= PA\_PLL\_RESET\_MASK;

 /\* Program the multiplier value \*/

 PAPLLCTL0 &= (~PA\_PLL\_CLKF\_MASK); //clear multiplier value

 PAPLLCTL0 &= (~PA\_PLL\_CLKR\_MASK); //clear divider value

 PAPLLCTL0 |= (papllclkf<<CLKF\_BIT\_SHIFT); // set PLLM

 PAPLLCTL0 |= (papllclkd<<CLKD\_BIT\_SHIFT); // set PLLD

 PAPLLCTL0 &= 0x00FFFFFF;

 PAPLLCTL0 |= ((((papllclkf + 1)>>1)-1)<<24);

 //PAPLLCTL1 = PAPLLCTL1 | 0x00002000;

 /\*Wait for PLL to properly reset (128 CLKIN1 cycles) \*/

 for(i=0;i<1000;i++);

 /\* take the PA PLL out of reset \*/

 PAPLLCTL1 &= (~PA\_PLL\_RESET\_MASK);

 /\*Wait for PLL to lock (2000 CLKIN1 cycles) \*/

 for(i=0;i<5000;i++);

 /\* enable PLL mode \*/

 PAPLLCTL0 &= ~(1<<BYPASS\_BIT\_SHIFT); // actually setting PLL MODE

 for(i=0;i<4000;i++);

 /\* Lock Chip Level Registers \*/

 KICK0 = KICK\_LOCK;

 KICK1 = KICK\_LOCK;

 papllctl0val\_final = PAPLLCTL0;

 papllctl1val\_final = PAPLLCTL1;

 GEL\_TextOut("Completed PA PLL Setup\n");

 GEL\_TextOut("PAPLLCTL0 - before: 0x%x\t after: 0x%x\n",,,,, papllctl0val\_orig, papllctl0val\_final);

 GEL\_TextOut("PAPLLCTL1 - before: 0x%x\t after: 0x%x\n",,,,, papllctl1val\_orig, papllctl1val\_final);

 if ((papllctl0val\_final != 0x09080500) || (papllctl1val\_final != 0x00002040))

 {

 return 1;

 }

 return 0;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//--------DDR3A Memory test----------------------

ddr3A\_memory\_test ()

{

 unsigned int index, value;

 GEL\_TextOut( "DDR3A memory test... Started\n" );

 /\* Write a pattern \*/

 for (index = DDR3A\_TEST\_START\_ADDRESS; index < DDR3A\_TEST\_END\_ADDRESS; index += 4) {

 \*index = index;

 }

 /\* Read and check the pattern \*/

 for (index = DDR3A\_TEST\_START\_ADDRESS; index < DDR3A\_TEST\_END\_ADDRESS; index += 4) {

 value = \*index;

 if (value != index) {

 GEL\_TextOut( "DDR3A memory test... Failed\n" );

 return -1;

 }

 }

 /\* Write a pattern for complementary values \*/

 for (index = DDR3A\_TEST\_START\_ADDRESS; index < DDR3A\_TEST\_END\_ADDRESS; index += 4) {

 \*index = ~index;

 }

 /\* Read and check the pattern \*/

 for (index = DDR3A\_TEST\_START\_ADDRESS; index < DDR3A\_TEST\_END\_ADDRESS; index += 4) {

 value = \*index;

 if (value != ~index) {

 GEL\_TextOut( "DDR3A memory test... Failed\n" );

 return -1;

 }

 }

 GEL\_TextOut( "DDR3A memory test... Passed\n" );

 return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \*

 \* NAME

 \* Setup\_Memory\_Map

 \*

 \* PURPOSE:

 \* Setup the Memory Map for EVMC6678L.

 \* Defined memory location avoid debugger access outside these locations.

 \*

 \* USAGE

 \* This routine can be called as:

 \*

 \* Setup\_Memory\_Map()

 \*

 \* RETURN VALUE

 \* NONE

 \*

 \* REFERENCE

 \* Based on TMS320C6678 datasheet.

 \*

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

hotmenu Setup\_Memory\_Map( )

{

 GEL\_TextOut("Setup\_Memory\_Map...\n",,);

 GEL\_MapOn( );

 GEL\_MapReset( );

 GEL\_MapAddStr( 0x00000000, 0, 0x21400000, "R|W|AS4", 0 ); //

 GEL\_MapAddStr( 0x21400000,0, 0x00000080, "R|W|AS4", 0 ); // Hyperlink Config (remote)

 //GEL\_MapAddStr( 0x21400080,0, 0x00000080, "R|W|AS4", 0 ); // Hyperlink Config (remote)

 GEL\_MapAddStr( 0x21400200, 0, 0xdebffe00, "R|W|AS4", 0 ); //

 GEL\_TextOut( "Setup\_Memory\_Map... Done.\n" );

}

/\*----------------------------------------------------- DDR3A : DDR800, 32bit--------------------------------------------------------------------------\*/

ddr3A\_32bit\_DDR800\_setup()

{

 unsigned int multiplier = 3;

 unsigned int divider = 0;

 int temp;

 unsigned int OD\_val = 2;

 KICK0 = 0x83E70B13;

 KICK1 = 0x95A4F1E0;

 //1. Poll for IDONE=1 in the PHY General Status Register 0 (address offset 0x010).

 do {

 read\_val = DDR3A\_PGSR0;

 } while ((read\_val&0x00000001) != 0x00000001);

 //4. Clocks are enabled and frequency is stable---------------------------------------

 //DDR3A PLL setup

 GEL\_TextOut ( "DDR3 PLL (PLL2) Setup ... \n");

 //DDR3APLLCTL0 = DDR3APLLCTL0 & 0xFF7FFFFF;

 // Set ENSAT = 1

 DDR3APLLCTL1 |= 0x00000040;

 // Put the PLL in PLL Mode

 DDR3APLLCTL0 |= 0x00800000;

 // In PLL Controller, reset the PLL (bit 13 in DDR3APLLCTL1 register)

 DDR3APLLCTL1 |= 0x00002000;

 // Program the necessary multipliers/dividers and BW adjustments

 // Set the divider values

 DDR3APLLCTL0 &= ~(0x0000003F);

 DDR3APLLCTL0 |= (divider & 0x0000003F);

 /\* Step 7: Programming OD[3:0] in the SECCTL register \*/

 DDR3APLLCTL0 &= OUTPUT\_DIVIDE\_MASK; // clear the OD bit field

 DDR3APLLCTL0 |= ~OUTPUT\_DIVIDE\_MASK & (OD\_val - 1) << OUTPUT\_DIVIDE\_OFFSET; // set the OD[3:0] bit field of PLLD to OD\_val

 /\* Set the Multipler values \*/

 DDR3APLLCTL0 &= ~(0x0007FFC0);

 DDR3APLLCTL0 |= ((multiplier << 6) & 0x0007FFC0 );

 temp = ((multiplier + 1) >> 1) - 1;

 DDR3APLLCTL0 &= ~(0xFF000000);

 DDR3APLLCTL0 |= ((temp << 24) & 0xFF000000);

 DDR3APLLCTL1 &= ~(0x0000000F);

 DDR3APLLCTL1 |= ((temp >> 8) & 0x0000000F);

 //In DDR3PLLCTL1, write PLLRST = 0 to bring PLL out of reset

 DDR3APLLCTL1 &= ~(0x00002000);

 // Put the PLL in PLL Mode

 DDR3APLLCTL0 &= ~(0x00800000); // ReSet the Bit 23

 GEL\_TextOut( "DDR3 PLL Setup complete, DDR3A clock now running at 400MHz.\n" );

 //DDR3A PLL setup complete ---------------------------------------

 /\*------------------------- Start PHY Configuration -------------------------------\*/

 //DDR3A\_PGCR1 = 0x0280C487;

 //5.a Program FRQSEL in the PLL Control Register (address offset 0x018).

 DDR3A\_PLLCR = 0xDC000; //Set FRQSEL=11, for ctl\_clk between 166-275MHz

 //5.b. Program WLSTEP=1, IODDRM=1, and ZCKSEL in the PHY General Configuration Register 1 (address offset 0x00C).

 DDR3A\_PGCR1 |= (1 << 2); //WLSTEP = 1

 DDR3A\_PGCR1 &= ~(IODDRM\_MASK);

 DDR3A\_PGCR1 |= (( 1 << 7) & IODDRM\_MASK);

 DDR3A\_PGCR1 &= ~(ZCKSEL\_MASK);

 DDR3A\_PGCR1 |= (( 1 << 23) & ZCKSEL\_MASK);

 //5.c. Program PHY Timing Parameters Register 0-4 (address offset 0x01C - 0x02C).

 DDR3A\_PTR0 = 0x42C21590;

 DDR3A\_PTR1 = 0xCFC712B3;

 // Maintaining default values of Phy Timing Parameters Register 2 in PUB

 DDR3A\_PTR3 = 0x04430D40;//0x18061A80;

 DDR3A\_PTR4 = 0x06413880;//0x0AAE7100;

 //5.d. Program PDQ, MPRDQ, and BYTEMASK in the DRAM Configuration Register (address offset 0x044).

 // All other fields must be left at their default values.

 DDR3A\_DCR &= ~(PDQ\_MASK); //PDQ = 0

 DDR3A\_DCR &= ~(MPRDQ\_MASK); //MPRDQ = 0

 DDR3A\_DCR &= ~(BYTEMASK\_MASK);

 DDR3A\_DCR |= (( 1 << 10) & BYTEMASK\_MASK);

 DDR3A\_DCR &= ~(NOSRA\_MASK);

 DDR3A\_DCR |= (( 1 << 27) & NOSRA\_MASK);

 DDR3A\_DCR &= ~(UDIMM\_MASK);

 DDR3A\_DCR |= (( 1 << 29) & UDIMM\_MASK);

 //5.e. Program DRAM Timing Parameters Register 0-2 (address offset 0x048 - 0x050).

 DDR3A\_DTPR0 = 0x008F6633; //0x50CE6644;

 DDR3A\_DTPR1 = 0x12820180;

 DDR3A\_DTPR2 = 0x50022A00;

 //5.f. Program BL=0, CL, WR, and PD=1 in the Mode Register 0 (address offset 0x054).

 //All other fields must be left at their default values.

 DDR3A\_MR0 = 0x00001620;

 //5.g. Program DIC, RTT, and TDQS in the Mode Register 1 (address offset 0x058).

 //All other fields must be left at their default values.

 DDR3A\_MR1 = 0x00000006;

 //---------------------------------------------------------------------------------------------------------

 //5.h. Program Mode Register 2 (address offset 0x05C).

 // Maintaining default values of Program Mode Register 2

 DDR3A\_MR2 = 0x00000040;

 //5.i. Program DTMPR=1, DTEXD, DTEXG, RANKEN=1 or 3, and RFSHDT=7 in the Data Training Configuration Register (address offset 0x068).

 //All other fields must be left at their default values.

 DDR3A\_DTCR = 0x710035C7; //0x710035C7;

 //5.j. Program tREFPRD=(5\*tREFI/ddr\_clk\_period) in the PHY General Configuration Register 2 (address offset 0x08C).

 //All other fields must be left at their default values.

 DDR3A\_PGCR2 = 0x00F03D09; //NOBUB = 0, FXDLAT = 0

 //DDR3A\_PGCR2 = 0x00F83D09; //NOBUB = 0, FXDLAT = 1

 //Set Impedence Register

 DDR3A\_ZQ0CR1 = 0x0000007B;

 DDR3A\_ZQ1CR1 = 0x0000007B;

 DDR3A\_ZQ2CR1 = 0x0000007B;

 //6. Re-trigger PHY initialization in DDR PHY through the VBUSP interface.

 //6.a. Program 0x00000033 to the PHY Initialization Register (address offset 0x004) to re-trigger PLL, ZCAL, and DCAL initialization.

 DDR3A\_PIR = 0x00000033;

 //6.b. Poll for IDONE=1 in the PHY General Status Register 0 (address offset 0x010).

 do {

 read\_val = DDR3A\_PGSR0;

 } while ((read\_val&0x00000001) != 0x00000001);

 //---------------------------------------------------------------------------------------------------------

 // 7. Trigger DDR3 initialization and leveling/training in DDR PHY through the VBUSP interface.

 // a. If using a 16-bit wide DDR interface, program DXEN=0 in the DATX8 2-7 General Configuration Registers (address offsets 0x240, 0x280, 0x2C0, 0x300, 0x340, and 0x380) to disable the leveling/training for the upper byte lanes.

 // b. If using a 32-bit wide DDR interface, program DXEN=0 in the DATX8 4-7 General Configuration Registers (address offsets 0x2C0, 0x300, 0x340, and 0x380) to disable the leveling/training for the upper byte lanes.

 // c. If ECC is not required, program DXEN=0 in the DATX8 8 General Configuration Register (address offset 0x3C0) to disable the leveling/training for the ECC byte lane.

 // NOTE: Setup supports 64-bit by default, ECC enable by default.

 //7.d. Program 0x0000XF81 to the PHY Initialization Register (address offset 0x004) to trigger DDR3 initialization and leveling/training sequences

 DDR3A\_PIR = 0x0000FF81; //WLADJ - ON

 //DDR3A\_PIR = 0x00000781; //WLADJ - OFF

 //---------------------------------------------------------------------------------------------------------

 //7.e. Poll for IDONE=1 in the PHY General Status Register 0 (address offset 0x010).

 do {

 read\_val = DDR3A\_PGSR0;

 } while ((read\_val&0x00000001) != 0x00000001);

 /\* End PHY Configuration \*/

 //---------------------------------------------------------------------------------------------------------

 /\* START EMIF INITIALIZATION

 ++++++++++++++++++SDCFG Register Calculation+++++++++++++++++++

 | 31 - 29 | 28 |27 - 25 | 24 | 23 - 22| 21 - 17 |

 |SDRAM\_TYPE|Rsvd|DDR\_TERM| DDQS | DYN\_ODT| Rsvd |

 | 0x011 | 0 | 0x011 | 0x1 | 0x00 | 0x0 |

 | 16-14 |13 - 12 | 11 - 8 | 7 |6 - 5 | 4 | 3 | 2 | 1 - 0 |

 | CWL | NM | CL | Rsvd |IBANK | Rsvd|EBANK| Rsvd|PAGE\_SIZE|

 | 0x11 | 0x00 | 0x1110 | 0x0 | 0x11 | 0x0 | 0 | 0 | 0x10 |

 SDCFG = 0x0110 0011 0010 0010 0011 0011 1011 0010

 SDCFG = 0x6700486A;//0x63223332

 SDRAM\_TYPE = 3

 DDR\_TERM = 3 (RZQ/4 = 1; RZQ/6=3)

 DDQS = 1

 DYN\_ODT = 0

 CWL = 3 (CWL5=0; CWL6=1; CWL7=2; CWL8=3)

 NM = 0 (64-bit=0, 32-bit=1, 16-bit=2)

 CL = 14 (CL5=2; CL6=4; CL7=6; CL8=8; CL9=10; CL10=12; CL11=14)

 IBANK = 3 (8bank)

 EBANK = 0 (0 - pad\_cs\_o\_n[0] , 1 - pad\_cs\_o\_n[1:0])

 PAGE\_SIZE = 2 (1024page-size=2; 2048page-size=3)

 \*/

 /\* Start DDR3A EMIF Configuration \*/

 //8. Configure the EMIF through the VBUSM interface.

 //8.a. Program all EMIF MMR\92s.

 DDR3A\_SDCFG = 0x62001462 ; //0x6200046A

 DDR3A\_SDTIM1 = 0x0A385022;

 DDR3A\_SDTIM2 = 0x00001CA5;

 DDR3A\_SDTIM3 = 0x210DFF22;

 DDR3A\_SDTIM4 = 0x533F03FF;

 DDR3A\_ZQCFG = 0x70073200;

 //8.b. Program reg\_initref\_dis=0 in the SDRAM Refresh Control Register (address offset 0x10).

 DDR3A\_SDRFC = 0x00000C34;

 GEL\_TextOut("DDR3A initialization complete \n");

 /\* End DDR3A EMIF Configuration \*/

}

/\*----------------------------------------------------- DDR3A : DDR1066, 32bit--------------------------------------------------------------------------\*/

ddr3A\_32bit\_DDR1066\_setup()

{

 unsigned int multiplier = 15;

 unsigned int divider = 0;

 int temp;

 unsigned int OD\_val = 6;

 KICK0 = 0x83E70B13;

 KICK1 = 0x95A4F1E0;

 //1. Poll for IDONE=1 in the PHY General Status Register 0 (address offset 0x010).

 do {

 read\_val = DDR3A\_PGSR0;

 } while ((read\_val&0x00000001) != 0x00000001);

 //4. Clocks are enabled and frequency is stable---------------------------------------

 //DDR3A PLL setup

 GEL\_TextOut ( "DDR3 PLL (PLL2) Setup ... \n");

 //DDR3APLLCTL0 = DDR3APLLCTL0 & 0xFF7FFFFF;

 // Set ENSAT = 1

 DDR3APLLCTL1 |= 0x00000040;

 // Put the PLL in PLL Mode

 DDR3APLLCTL0 |= 0x00800000;

 // In PLL Controller, reset the PLL (bit 13 in DDR3APLLCTL1 register)

 DDR3APLLCTL1 |= 0x00002000;

 // Program the necessary multipliers/dividers and BW adjustments

 // Set the divider values

 DDR3APLLCTL0 &= ~(0x0000003F);

 DDR3APLLCTL0 |= (divider & 0x0000003F);

 /\* Step 7: Programming OD[3:0] in the SECCTL register \*/

 DDR3APLLCTL0 &= OUTPUT\_DIVIDE\_MASK; // clear the OD bit field

 DDR3APLLCTL0 |= ~OUTPUT\_DIVIDE\_MASK & (OD\_val - 1) << OUTPUT\_DIVIDE\_OFFSET; // set the OD[3:0] bit field of PLLD to OD\_val

 /\* Set the Multipler values \*/

 DDR3APLLCTL0 &= ~(0x0007FFC0);

 DDR3APLLCTL0 |= ((multiplier << 6) & 0x0007FFC0 );

 temp = ((multiplier + 1) >> 1) - 1;

 DDR3APLLCTL0 &= ~(0xFF000000);

 DDR3APLLCTL0 |= ((temp << 24) & 0xFF000000);

 DDR3APLLCTL1 &= ~(0x0000000F);

 DDR3APLLCTL1 |= ((temp >> 8) & 0x0000000F);

 //In DDR3PLLCTL1, write PLLRST = 0 to bring PLL out of reset

 DDR3APLLCTL1 &= ~(0x00002000);

 // Put the PLL in PLL Mode

 DDR3APLLCTL0 &= ~(0x00800000); // ReSet the Bit 23

 GEL\_TextOut( "DDR3 PLL Setup complete, DDR3A clock now running at 533MHz.\n" );

 //DDR3A PLL setup complete ---------------------------------------

 /\*------------------------- Start PHY Configuration -------------------------------\*/

 //DDR3A\_PGCR1 = 0x0280C487;

 //5.a Program FRQSEL in the PLL Control Register (address offset 0x018).

 DDR3A\_PLLCR = 0xDC000; //Set FRQSEL=11, for ctl\_clk between 166-275MHz

 //5.b. Program WLSTEP=1, IODDRM=1, and ZCKSEL in the PHY General Configuration Register 1 (address offset 0x00C).

 DDR3A\_PGCR1 |= (1 << 2); //WLSTEP = 1

 DDR3A\_PGCR1 &= ~(IODDRM\_MASK);

 DDR3A\_PGCR1 |= (( 1 << 7) & IODDRM\_MASK);

 DDR3A\_PGCR1 &= ~(ZCKSEL\_MASK);

 DDR3A\_PGCR1 |= (( 1 << 23) & ZCKSEL\_MASK);

 //5.c. Program PHY Timing Parameters Register 0-4 (address offset 0x01C - 0x02C).

 DDR3A\_PTR0 = 0x426213CF;

 DDR3A\_PTR1 = 0xCFC712B3;

 // Maintaining default values of Phy Timing Parameters Register 2 in PUB

 DDR3A\_PTR3 = 0x05B411AA;//0x09041104;//0x18061A80;

 DDR3A\_PTR4 = 0x0855A0AA;//0x0AAE7100;

 //5.d. Program PDQ, MPRDQ, and BYTEMASK in the DRAM Configuration Register (address offset 0x044).

 // All other fields must be left at their default values.

 DDR3A\_DCR &= ~(PDQ\_MASK); //PDQ = 0

 DDR3A\_DCR &= ~(MPRDQ\_MASK); //MPRDQ = 0

 DDR3A\_DCR &= ~(BYTEMASK\_MASK);

 DDR3A\_DCR |= (( 1 << 10) & BYTEMASK\_MASK);

 DDR3A\_DCR &= ~(NOSRA\_MASK);

 DDR3A\_DCR |= (( 1 << 27) & NOSRA\_MASK);

 //DDR3A\_DCR &= ~(UDIMM\_MASK);

 //DDR3A\_DCR |= (( 1 << 29) & UDIMM\_MASK);

 //RRMODE

 //DDR3A\_DSGCR &= ~(RRMODE\_MASK); //RR\_MODE = 0

 //DDR3A\_DSGCR &= ~(RRMODE\_MASK); //RR\_MODE = 1

 //DDR3A\_DSGCR |= (( 1 << 18) & RRMODE\_MASK);

 //5.e. Program DRAM Timing Parameters Register 0-2 (address offset 0x048 - 0x050).

 DDR3A\_DTPR0 = 0x54D47744;//0x6D148844; //0x6D148844; 0x69137764 ---changed in rev 1.3

 DDR3A\_DTPR1 = 0x1282AA00;//0x12845A00;

 DDR3A\_DTPR2 = 0x50023200;

 //5.f. Program BL=0, CL, WR, and PD=1 in the Mode Register 0 (address offset 0x054).

 //All other fields must be left at their default values.

 DDR3A\_MR0 = 0x00001830; //0x00001870;

 //5.g. Program DIC, RTT, and TDQS in the Mode Register 1 (address offset 0x058).

 //All other fields must be left at their default values.

 DDR3A\_MR1 = 0x00000006; //0x00000044; ---changed in rev 1.3

 //---------------------------------------------------------------------------------------------------------

 //5.h. Program Mode Register 2 (address offset 0x05C).

 // Maintaining default values of Program Mode Register 2

 DDR3A\_MR2 = 0x00000048; //18 ---changed in rev 1.3

 //5.i. Program DTMPR=1, DTEXD, DTEXG, RANKEN=1 or 3, and RFSHDT=7 in the Data Training Configuration Register (address offset 0x068).

 //All other fields must be left at their default values.

 DDR3A\_DTCR = 0x710035C7; //0x730035C7;

 //5.j. Program tREFPRD=(5\*tREFI/ddr\_clk\_period) in the PHY General Configuration Register 2 (address offset 0x08C).

 //All other fields must be left at their default values.

 DDR3A\_PGCR2 = 0x00F05161; //NOBUB = 0, FXDLAT = 0

 //DDR3A\_PGCR2 = 0x00F83D09; //NOBUB = 0, FXDLAT = 1

 //Set Impedence Register

 DDR3A\_ZQ0CR1 = 0x0000007B;

 DDR3A\_ZQ1CR1 = 0x0000007B;

 DDR3A\_ZQ2CR1 = 0x0000007B;

 //6. Re-trigger PHY initialization in DDR PHY through the VBUSP interface.

 //6.a. Program 0x00000033 to the PHY Initialization Register (address offset 0x004) to re-trigger PLL, ZCAL, and DCAL initialization.

 DDR3A\_PIR = 0x00000033;

 //6.b. Poll for IDONE=1 in the PHY General Status Register 0 (address offset 0x010).

 do {

 read\_val = DDR3A\_PGSR0;

 } while ((read\_val&0x00000001) != 0x00000001);

 //---------------------------------------------------------------------------------------------------------

 // 7. Trigger DDR3 initialization and leveling/training in DDR PHY through the VBUSP interface.

 // a. If using a 16-bit wide DDR interface, program DXEN=0 in the DATX8 2-7 General Configuration Registers (address offsets 0x240, 0x280, 0x2C0, 0x300, 0x340, and 0x380) to disable the leveling/training for the upper byte lanes.

 // b. If using a 32-bit wide DDR interface, program DXEN=0 in the DATX8 4-7 General Configuration Registers (address offsets 0x2C0, 0x300, 0x340, and 0x380) to disable the leveling/training for the upper byte lanes.

 // c. If ECC is not required, program DXEN=0 in the DATX8 8 General Configuration Register (address offset 0x3C0) to disable the leveling/training for the ECC byte lane.

 // NOTE: Setup supports 64-bit by default, ECC enable by default.

 //7.d. Program 0x0000XF81 to the PHY Initialization Register (address offset 0x004) to trigger DDR3 initialization and leveling/training sequences

 DDR3A\_PIR = 0x0000FF81; //WLADJ - ON

 //DDR3A\_PIR = 0x00000781; //WLADJ - OFF

 //---------------------------------------------------------------------------------------------------------

 //7.e. Poll for IDONE=1 in the PHY General Status Register 0 (address offset 0x010).

 do {

 read\_val = DDR3A\_PGSR0;

 } while ((read\_val&0x00000001) != 0x00000001);

 /\* End PHY Configuration \*/

 //---------------------------------------------------------------------------------------------------------

 /\* START EMIF INITIALIZATION

 ++++++++++++++++++SDCFG Register Calculation+++++++++++++++++++

 | 31 - 29 | 28 |27 - 25 | 24 | 23 - 22| 21 - 17 |

 |SDRAM\_TYPE|Rsvd|DDR\_TERM| DDQS | DYN\_ODT| Rsvd |

 | 0x011 | 0 | 0x011 | 0x1 | 0x00 | 0x0 |

 | 16-14 |13 - 12 | 11 - 8 | 7 |6 - 5 | 4 | 3 | 2 | 1 - 0 |

 | CWL | NM | CL | Rsvd |IBANK | Rsvd|EBANK| Rsvd|PAGE\_SIZE|

 | 0x11 | 0x00 | 0x1110 | 0x0 | 0x11 | 0x0 | 0 | 0 | 0x10 |

 SDCFG = 0x0110 0011 0010 0010 0011 0011 1011 0010

 SDCFG = 0x6700486A;//0x63223332

 SDRAM\_TYPE = 3

 DDR\_TERM = 3 (RZQ/4 = 1; RZQ/6=3)

 DDQS = 1

 DYN\_ODT = 0

 CWL = 3 (CWL5=0; CWL6=1; CWL7=2; CWL8=3)

 NM = 0 (64-bit=0, 32-bit=1, 16-bit=2)

 CL = 14 (CL5=2; CL6=4; CL7=6; CL8=8; CL9=10; CL10=12; CL11=14)

 IBANK = 3 (8bank)

 EBANK = 0 (0 - pad\_cs\_o\_n[0] , 1 - pad\_cs\_o\_n[1:0])

 PAGE\_SIZE = 2 (1024page-size=2; 2048page-size=3)

 \*/

 /\* Start DDR3A EMIF Configuration \*/

 //8. Configure the EMIF through the VBUSM interface.

 //8.a. Program all EMIF MMR\92s.

 DDR3A\_SDCFG = 0x62005662; //0x62005662;

 DDR3A\_SDTIM1 = 0x0E4C6833; //0x0E4C6833;//0x0E4C6833;

 DDR3A\_SDTIM2 = 0x00001CC6; //0x00001CE7;

 DDR3A\_SDTIM3 = 0x3169FF32; //0x323DFF32;

 DDR3A\_SDTIM4 = 0x533F054F; //0x533F08AF;

 DDR3A\_ZQCFG = 0x70073200;//0xF0073200;

 //8.b. Program reg\_initref\_dis=0 in the SDRAM Refresh Control Register (address offset 0x10).

 DDR3A\_SDRFC = 0x00001045;

 GEL\_TextOut("DDR3A initialization complete \n");

 /\* End DDR3A EMIF Configuration \*/

}

/\*--------------------------------------------------

 DDR3A : DDR1333,32bit

---------------------------------------------------\*/

ddr3A\_32bit\_DDR1333\_setup()

{

 unsigned int multiplier = 19;

 unsigned int divider = 0;

 int temp;

 unsigned int OD\_val = 6;

 KICK0 = 0x83E70B13;

 KICK1 = 0x95A4F1E0;

 //1. Poll for IDONE=1 in the PHY General Status Register 0 (address offset 0x010).

 do {

 read\_val = DDR3A\_PGSR0;

 } while ((read\_val&0x00000001) != 0x00000001);

 //4. Clocks are enabled and frequency is stable---------------------------------------

 //DDR3A PLL setup

 GEL\_TextOut ( "DDR3 PLL (PLL2) Setup ... \n");

 //DDR3APLLCTL0 = DDR3APLLCTL0 & 0xFF7FFFFF;

 // Set ENSAT = 1

 DDR3APLLCTL1 |= 0x00000040;

 // Put the PLL in PLL Mode

 DDR3APLLCTL0 |= 0x00800000;

 // In PLL Controller, reset the PLL (bit 13 in DDR3APLLCTL1 register)

 DDR3APLLCTL1 |= 0x00002000;

 // Program the necessary multipliers/dividers and BW adjustments

 // Set the divider values

 DDR3APLLCTL0 &= ~(0x0000003F);

 DDR3APLLCTL0 |= (divider & 0x0000003F);

 /\* Step 7: Programming OD[3:0] in the SECCTL register \*/

 DDR3APLLCTL0 &= OUTPUT\_DIVIDE\_MASK; // clear the OD bit field

 DDR3APLLCTL0 |= ~OUTPUT\_DIVIDE\_MASK & (OD\_val - 1) << OUTPUT\_DIVIDE\_OFFSET; // set the OD[3:0] bit field of PLLD to OD\_val

 /\* Set the Multipler values \*/

 DDR3APLLCTL0 &= ~(0x0007FFC0);

 DDR3APLLCTL0 |= ((multiplier << 6) & 0x0007FFC0 );

 temp = ((multiplier + 1) >> 1) - 1;

 DDR3APLLCTL0 &= ~(0xFF000000);

 DDR3APLLCTL0 |= ((temp << 24) & 0xFF000000);

 DDR3APLLCTL1 &= ~(0x0000000F);

 DDR3APLLCTL1 |= ((temp >> 8) & 0x0000000F);

 //In DDR3PLLCTL1, write PLLRST = 0 to bring PLL out of reset

 DDR3APLLCTL1 &= ~(0x00002000);

 // Put the PLL in PLL Mode

 DDR3APLLCTL0 &= ~(0x00800000); // ReSet the Bit 23

 GEL\_TextOut( "DDR3 PLL Setup complete, DDR3A clock now running at 666 MHz.\n" );

 //DDR3A PLL setup complete ---------------------------------------

 /\*------------------------- Start PHY Configuration -------------------------------\*/

 //DDR3A\_PGCR1 = 0x0280C487;

 //5.a Program FRQSEL in the PLL Control Register (address offset 0x018).

 DDR3A\_PLLCR = 0x0005C000; //Set FRQSEL=01, for ctl\_clk between 225-385MHz

 //5.b. Program WLSTEP=1, IODDRM=1, and ZCKSEL in the PHY General Configuration Register 1 (address offset 0x00C).

 DDR3A\_PGCR1 |= (1 << 2); //WLSTEP = 1

 DDR3A\_PGCR1 &= ~(IODDRM\_MASK);

 DDR3A\_PGCR1 |= (( 1 << 7) & IODDRM\_MASK);

 DDR3A\_PGCR1 &= ~(ZCKSEL\_MASK);

 DDR3A\_PGCR1 |= (( 1 << 23) & ZCKSEL\_MASK);

 //5.c. Program PHY Timing Parameters Register 0-4 (address offset 0x01C - 0x02C).

 DDR3A\_PTR0 = 0x426213CF;

 DDR3A\_PTR1 = 0xCFC712B3;

 // Maintaining default values of Phy Timing Parameters Register 2 in PUB

 DDR3A\_PTR3 = 0x07151615;//0x072515C2; //0x0B4515C2;//0x18061A80;

 DDR3A\_PTR4 = 0x0A6A08D5;//0x0AAE7100;

 //5.d. Program PDQ, MPRDQ, and BYTEMASK in the DRAM Configuration Register (address offset 0x044).

 // All other fields must be left at their default values.

 DDR3A\_DCR &= ~(PDQ\_MASK); //PDQ = 0

 DDR3A\_DCR &= ~(MPRDQ\_MASK); //MPRDQ = 0

 DDR3A\_DCR &= ~(BYTEMASK\_MASK);

 DDR3A\_DCR |= (( 1 << 10) & BYTEMASK\_MASK);

 DDR3A\_DCR &= ~(NOSRA\_MASK);

 DDR3A\_DCR |= (( 1 << 27) & NOSRA\_MASK);

 //DDR3A\_DCR &= ~(UDIMM\_MASK);

 //DDR3A\_DCR |= (( 1 << 29) & UDIMM\_MASK);

 //5.e. Program DRAM Timing Parameters Register 0-2 (address offset 0x048 - 0x050).

 DDR3A\_DTPR0 = 0xAD189955;//0x85589975;//0x8558AA55;

 DDR3A\_DTPR1 = 0x12835A80;//0x12835A80;//0x12857280;

 DDR3A\_DTPR2 = 0x5002C200;

 //5.f. Program BL=0, CL, WR, and PD=1 in the Mode Register 0 (address offset 0x054).

 //All other fields must be left at their default values.

 DDR3A\_MR0 = 0x00001A60; //50

 //5.g. Program DIC, RTT, and TDQS in the Mode Register 1 (address offset 0x058).

 //All other fields must be left at their default values.

 DDR3A\_MR1 = 0x00000006;

 //---------------------------------------------------------------------------------------------------------

 //5.h. Program Mode Register 2 (address offset 0x05C).

 // Maintaining default values of Program Mode Register 2

 DDR3A\_MR2 = 0x00000050;

 //5.i. Program DTMPR=1, DTEXD, DTEXG, RANKEN=1 or 3, and RFSHDT=7 in the Data Training Configuration Register (address offset 0x068).

 //All other fields must be left at their default values.

 DDR3A\_DTCR = 0x710035C7; //0x730035C7;

 //5.j. Program tREFPRD=(5\*tREFI/ddr\_clk\_period) in the PHY General Configuration Register 2 (address offset 0x08C).

 //All other fields must be left at their default values.

 DDR3A\_PGCR2 = 0x00F065B9; //NOBUB = 0, FXDLAT = 0

 //DDR3A\_PGCR2 = 0x00F83D09; //NOBUB = 0, FXDLAT = 1

 //Set Impedence Register

 DDR3A\_ZQ0CR1 = 0x0000007B;

 DDR3A\_ZQ1CR1 = 0x0000007B;

 DDR3A\_ZQ2CR1 = 0x0000007B;

 //DDR3A\_ZQ3CR1 = 0x0000005D;

 //6. Re-trigger PHY initialization in DDR PHY through the VBUSP interface.

 //6.a. Program 0x00000033 to the PHY Initialization Register (address offset 0x004) to re-trigger PLL, ZCAL, and DCAL initialization.

 DDR3A\_PIR = 0x00000033;

 //6.b. Poll for IDONE=1 in the PHY General Status Register 0 (address offset 0x010).

 do {

 read\_val = DDR3A\_PGSR0;

 } while ((read\_val&0x00000001) != 0x00000001);

 //---------------------------------------------------------------------------------------------------------

 // 7. Trigger DDR3 initialization and leveling/training in DDR PHY through the VBUSP interface.

 // a. If using a 16-bit wide DDR interface, program DXEN=0 in the DATX8 2-7 General Configuration Registers (address offsets 0x240, 0x280, 0x2C0, 0x300, 0x340, and 0x380) to disable the leveling/training for the upper byte lanes.

 // b. If using a 32-bit wide DDR interface, program DXEN=0 in the DATX8 4-7 General Configuration Registers (address offsets 0x2C0, 0x300, 0x340, and 0x380) to disable the leveling/training for the upper byte lanes.

 // c. If ECC is not required, program DXEN=0 in the DATX8 8 General Configuration Register (address offset 0x3C0) to disable the leveling/training for the ECC byte lane.

 // NOTE: Setup supports 64-bit by default, ECC enable by default.

 //7.d. Program 0x0000XF81 to the PHY Initialization Register (address offset 0x004) to trigger DDR3 initialization and leveling/training sequences

 DDR3A\_PIR = 0x0000FF81; //WLADJ - ON

 //DDR3A\_PIR = 0x00000781; //WLADJ - OFF

 //---------------------------------------------------------------------------------------------------------

 //7.e. Poll for IDONE=1 in the PHY General Status Register 0 (address offset 0x010).

 do {

 read\_val = DDR3A\_PGSR0;

 } while ((read\_val&0x00000001) != 0x00000001);

 /\* End PHY Configuration \*/

 //---------------------------------------------------------------------------------------------------------

 /\* START EMIF INITIALIZATION

 ++++++++++++++++++SDCFG Register Calculation+++++++++++++++++++

 | 31 - 29 | 28 |27 - 25 | 24 | 23 - 22| 21 - 17 |

 |SDRAM\_TYPE|Rsvd|DDR\_TERM| DDQS | DYN\_ODT| Rsvd |

 | 0x011 | 0 | 0x011 | 0x1 | 0x00 | 0x0 |

 | 16-14 |13 - 12 | 11 - 8 | 7 |6 - 5 | 4 | 3 | 2 | 1 - 0 |

 | CWL | NM | CL | Rsvd |IBANK | Rsvd|EBANK| Rsvd|PAGE\_SIZE|

 | 0x11 | 0x00 | 0x1110 | 0x0 | 0x11 | 0x0 | 0 | 0 | 0x10 |

 SDCFG = 0x0110 0011 0010 0010 0011 0011 1011 0010

 SDCFG = 0x6700486A;//0x63223332

 SDRAM\_TYPE = 3

 DDR\_TERM = 3 (RZQ/4 = 1; RZQ/6=3)

 DDQS = 1

 DYN\_ODT = 0

 CWL = 3 (CWL5=0; CWL6=1; CWL7=2; CWL8=3)

 NM = 0 (64-bit=0, 32-bit=1, 16-bit=2)

 CL = 14 (CL5=2; CL6=4; CL7=6; CL8=8; CL9=10; CL10=12; CL11=14)

 IBANK = 3 (8bank)

 EBANK = 0 (0 - pad\_cs\_o\_n[0] , 1 - pad\_cs\_o\_n[1:0])

 PAGE\_SIZE = 2 (1024page-size=2; 2048page-size=3)

 \*/

 /\* Start DDR3A EMIF Configuration \*/

 //8. Configure the EMIF through the VBUSM interface.

 //8.a. Program all EMIF MMR\92s.

 DDR3A\_SDCFG = 0x62009C62; // 9A62//0x62008C62 ;//0x6600CE62=single rank,0x6600CE6A=dual rank

 DDR3A\_SDTIM1 = 0x125C7C44;

 DDR3A\_SDTIM2 = 0x00001D08;

 DDR3A\_SDTIM3 = 0x31C1FF43;

 DDR3A\_SDTIM4 = 0x543F06AF;

 DDR3A\_ZQCFG = 0x70073200;

 //8.b. Program reg\_initref\_dis=0 in the SDRAM Refresh Control Register (address offset 0x10).

 DDR3A\_SDRFC = 0x00001457;

 GEL\_TextOut("DDR3A initialization complete \n");

 /\* End DDR3A EMIF Configuration \*/

}

/\*--------------------------------------------------

 DDR3A : DDR1600,64bit

---------------------------------------------------\*/

ddr3A\_64bit\_DDR1600\_setup()

{

 unsigned int multiplier = 7;

 unsigned int divider = 0;

 int temp;

 unsigned int OD\_val = 2;

 KICK0 = 0x83E70B13;

 KICK1 = 0x95A4F1E0;

 //1. Poll for IDONE=1 in the PHY General Status Register 0 (address offset 0x010).

 do {

 read\_val = DDR3A\_PGSR0;

 } while ((read\_val&0x00000001) != 0x00000001);

 //4. Clocks are enabled and frequency is stable---------------------------------------

 //DDR3A PLL setup

 GEL\_TextOut ( "DDR3 PLL (PLL2) Setup ... \n");

 //DDR3APLLCTL0 = DDR3APLLCTL0 & 0xFF7FFFFF;

 // Set ENSAT = 1

 DDR3APLLCTL1 |= 0x00000040;

 // Put the PLL in PLL Mode

 DDR3APLLCTL0 |= 0x00800000;

 // In PLL Controller, reset the PLL (bit 13 in DDR3APLLCTL1 register)

 DDR3APLLCTL1 |= 0x00002000;

 // Program the necessary multipliers/dividers and BW adjustments

 // Set the divider values

 DDR3APLLCTL0 &= ~(0x0000003F);

 DDR3APLLCTL0 |= (divider & 0x0000003F);

 /\* Step 7: Programming OD[3:0] in the SECCTL register \*/

 DDR3APLLCTL0 &= OUTPUT\_DIVIDE\_MASK; // clear the OD bit field

 DDR3APLLCTL0 |= ~OUTPUT\_DIVIDE\_MASK & (OD\_val - 1) << OUTPUT\_DIVIDE\_OFFSET; // set the OD[3:0] bit field of PLLD to OD\_val

 /\* Set the Multipler values \*/

 DDR3APLLCTL0 &= ~(0x0007FFC0);

 DDR3APLLCTL0 |= ((multiplier << 6) & 0x0007FFC0 );

 temp = ((multiplier + 1) >> 1) - 1;

 DDR3APLLCTL0 &= ~(0xFF000000);

 DDR3APLLCTL0 |= ((temp << 24) & 0xFF000000);

 DDR3APLLCTL1 &= ~(0x0000000F);

 DDR3APLLCTL1 |= ((temp >> 8) & 0x0000000F);

 //In DDR3PLLCTL1, write PLLRST = 0 to bring PLL out of reset

 DDR3APLLCTL1 &= ~(0x00002000);

 // Put the PLL in PLL Mode

 DDR3APLLCTL0 &= ~(0x00800000); // ReSet the Bit 23

 GEL\_TextOut( "DDR3 PLL Setup complete, DDR3A clock now running at 666 MHz.\n" );

 //DDR3A PLL setup complete ---------------------------------------

 /\*------------------------- Start PHY Configuration -------------------------------\*/

 //DDR3A\_PGCR1 = 0x0280C487;

 //5.a Program FRQSEL in the PLL Control Register (address offset 0x018).

 DDR3A\_PLLCR = 0x1C000; //Set FRQSEL=00, for ctl\_clk between 335-533MHz

 //5.b. Program WLSTEP=1, IODDRM=1, and ZCKSEL in the PHY General Configuration Register 1 (address offset 0x00C).

 DDR3A\_PGCR1 |= (1 << 2); //WLSTEP = 1

 DDR3A\_PGCR1 &= ~(IODDRM\_MASK);

 DDR3A\_PGCR1 |= (( 1 << 7) & IODDRM\_MASK);

 DDR3A\_PGCR1 &= ~(ZCKSEL\_MASK);

 DDR3A\_PGCR1 |= (( 1 << 23) & ZCKSEL\_MASK);

 //5.c. Program PHY Timing Parameters Register 0-4 (address offset 0x01C - 0x02C).

 DDR3A\_PTR0 = 0x426213CF;

 DDR3A\_PTR1 = 0xCFC712B3;

 // Maintaining default values of Phy Timing Parameters Register 2 in PUB

 DDR3A\_PTR3 = 0x08861A80;//0x072515C2; //0x0B4515C2;//0x18061A80;

 DDR3A\_PTR4 = 0x0C827100;//0x0AAE7100;

 //5.d. Program PDQ, MPRDQ, and BYTEMASK in the DRAM Configuration Register (address offset 0x044).

 // All other fields must be left at their default values.

 DDR3A\_DCR &= ~(PDQ\_MASK); //PDQ = 0

 DDR3A\_DCR &= ~(MPRDQ\_MASK); //MPRDQ = 0

 DDR3A\_DCR &= ~(BYTEMASK\_MASK);

 DDR3A\_DCR |= (( 1 << 10) & BYTEMASK\_MASK);

 DDR3A\_DCR &= ~(NOSRA\_MASK);

 DDR3A\_DCR |= (( 1 << 27) & NOSRA\_MASK);

 //DDR3A\_DCR &= ~(UDIMM\_MASK);

 //DDR3A\_DCR |= (( 1 << 29) & UDIMM\_MASK);

 //5.e. Program DRAM Timing Parameters Register 0-2 (address offset 0x048 - 0x050).

 DDR3A\_DTPR0 = 0x011CBB66;//0x85589975;//0x8558AA55;

 DDR3A\_DTPR1 = 0x12840300;//0x12835A80;//0x12857280;

 DDR3A\_DTPR2 = 0x5002CE00;

 //5.f. Program BL=0, CL, WR, and PD=1 in the Mode Register 0 (address offset 0x054).

 //All other fields must be left at their default values.

 DDR3A\_MR0 = 0x00001C70; //50

 //5.g. Program DIC, RTT, and TDQS in the Mode Register 1 (address offset 0x058).

 //All other fields must be left at their default values.

 DDR3A\_MR1 = 0x00000006;

 //---------------------------------------------------------------------------------------------------------

 //5.h. Program Mode Register 2 (address offset 0x05C).

 // Maintaining default values of Program Mode Register 2

 DDR3A\_MR2 = 0x00000058;

 //5.i. Program DTMPR=1, DTEXD, DTEXG, RANKEN=1 or 3, and RFSHDT=7 in the Data Training Configuration Register (address offset 0x068).

 //All other fields must be left at their default values.

 DDR3A\_DTCR = 0x710035C7; //0x730035C7;

 //5.j. Program tREFPRD=(5\*tREFI/ddr\_clk\_period) in the PHY General Configuration Register 2 (address offset 0x08C).

 //All other fields must be left at their default values.

 DDR3A\_PGCR2 = 0x00F07A12; //NOBUB = 0, FXDLAT = 0

 //DDR3A\_PGCR2 = 0x00F83D09; //NOBUB = 0, FXDLAT = 1

 //Set Impedence Register

 DDR3A\_ZQ0CR1 = 0x0000007B;

 DDR3A\_ZQ1CR1 = 0x0000007B;

 DDR3A\_ZQ2CR1 = 0x0000007B;

 //DDR3A\_ZQ3CR1 = 0x0000005D;

 //6. Re-trigger PHY initialization in DDR PHY through the VBUSP interface.

 //6.a. Program 0x00000033 to the PHY Initialization Register (address offset 0x004) to re-trigger PLL, ZCAL, and DCAL initialization.

 DDR3A\_PIR = 0x00000033;

 //6.b. Poll for IDONE=1 in the PHY General Status Register 0 (address offset 0x010).

 do {

 read\_val = DDR3A\_PGSR0;

 } while ((read\_val&0x00000001) != 0x00000001);

 //---------------------------------------------------------------------------------------------------------

 // 7. Trigger DDR3 initialization and leveling/training in DDR PHY through the VBUSP interface.

 // a. If using a 16-bit wide DDR interface, program DXEN=0 in the DATX8 2-7 General Configuration Registers (address offsets 0x240, 0x280, 0x2C0, 0x300, 0x340, and 0x380) to disable the leveling/training for the upper byte lanes.

 // b. If using a 32-bit wide DDR interface, program DXEN=0 in the DATX8 4-7 General Configuration Registers (address offsets 0x2C0, 0x300, 0x340, and 0x380) to disable the leveling/training for the upper byte lanes.

 // c. If ECC is not required, program DXEN=0 in the DATX8 8 General Configuration Register (address offset 0x3C0) to disable the leveling/training for the ECC byte lane.

 // NOTE: Setup supports 64-bit by default, ECC enable by default.

 //7.d. Program 0x0000XF81 to the PHY Initialization Register (address offset 0x004) to trigger DDR3 initialization and leveling/training sequences

 DDR3A\_PIR = 0x0000FF81; //WLADJ - ON

 //DDR3A\_PIR = 0x00000781; //WLADJ - OFF

 //---------------------------------------------------------------------------------------------------------

 //7.e. Poll for IDONE=1 in the PHY General Status Register 0 (address offset 0x010).

 do {

 read\_val = DDR3A\_PGSR0;

 } while ((read\_val&0x00000001) != 0x00000001);

 /\* End PHY Configuration \*/

 //---------------------------------------------------------------------------------------------------------

 /\* START EMIF INITIALIZATION

 ++++++++++++++++++SDCFG Register Calculation+++++++++++++++++++

 | 31 - 29 | 28 |27 - 25 | 24 | 23 - 22| 21 - 17 |

 |SDRAM\_TYPE|Rsvd|DDR\_TERM| DDQS | DYN\_ODT| Rsvd |

 | 0x011 | 0 | 0x011 | 0x1 | 0x00 | 0x0 |

 | 16-14 |13 - 12 | 11 - 8 | 7 |6 - 5 | 4 | 3 | 2 | 1 - 0 |

 | CWL | NM | CL | Rsvd |IBANK | Rsvd|EBANK| Rsvd|PAGE\_SIZE|

 | 0x11 | 0x00 | 0x1110 | 0x0 | 0x11 | 0x0 | 0 | 0 | 0x10 |

 SDCFG = 0x0110 0011 0010 0010 0011 0011 1011 0010

 SDCFG = 0x6700486A;//0x63223332

 SDRAM\_TYPE = 3

 DDR\_TERM = 3 (RZQ/4 = 1; RZQ/6=3)

 DDQS = 1

 DYN\_ODT = 0

 CWL = 3 (CWL5=0; CWL6=1; CWL7=2; CWL8=3)

 NM = 0 (64-bit=0, 32-bit=1, 16-bit=2)

 CL = 14 (CL5=2; CL6=4; CL7=6; CL8=8; CL9=10; CL10=12; CL11=14)

 IBANK = 3 (8bank)

 EBANK = 0 (0 - pad\_cs\_o\_n[0] , 1 - pad\_cs\_o\_n[1:0])

 PAGE\_SIZE = 2 (1024page-size=2; 2048page-size=3)

 \*/

 /\* Start DDR3A EMIF Configuration \*/

 //8. Configure the EMIF through the VBUSM interface.

 //8.a. Program all EMIF MMR\92s.

 DDR3A\_SDCFG = 0x6200CE62; // 9A62//0x62008C62 ;//0x6600CE62=single rank,0x6600CE6A=dual rank

 DDR3A\_SDTIM1 = 0x166C9455;

 DDR3A\_SDTIM2 = 0x00001D4A;

 DDR3A\_SDTIM3 = 0x321DFF53;

 DDR3A\_SDTIM4 = 0x543F07FF;

 DDR3A\_ZQCFG = 0x042D3B70;//0x70073200;

 //8.b. Program reg\_initref\_dis=0 in the SDRAM Refresh Control Register (address offset 0x10).

 DDR3A\_SDRFC = 0x00001869;

 GEL\_TextOut("DDR3A initialization complete \n");

 /\* End DDR3A EMIF Configuration \*/

}

/\*--------------------------------------------------

 DDR3A : DDR1600,32bit

---------------------------------------------------\*/

ddr3A\_32bit\_DDR1600\_setup()

{

 unsigned int multiplier = 7;

 unsigned int divider = 0;

 int temp;

 unsigned int OD\_val = 2;

 KICK0 = 0x83E70B13;

 KICK1 = 0x95A4F1E0;

 //1. Poll for IDONE=1 in the PHY General Status Register 0 (address offset 0x010).

 do {

 read\_val = DDR3A\_PGSR0;

 } while ((read\_val&0x00000001) != 0x00000001);

 //4. Clocks are enabled and frequency is stable---------------------------------------

 //DDR3A PLL setup

 GEL\_TextOut ( "DDR3 PLL (PLL2) Setup ... \n");

 //DDR3APLLCTL0 = DDR3APLLCTL0 & 0xFF7FFFFF;

 // Set ENSAT = 1

 DDR3APLLCTL1 |= 0x00000040;

 // Put the PLL in PLL Mode

 DDR3APLLCTL0 |= 0x00800000;

 // In PLL Controller, reset the PLL (bit 13 in DDR3APLLCTL1 register)

 DDR3APLLCTL1 |= 0x00002000;

 // Program the necessary multipliers/dividers and BW adjustments

 // Set the divider values

 DDR3APLLCTL0 &= ~(0x0000003F);

 DDR3APLLCTL0 |= (divider & 0x0000003F);

 /\* Step 7: Programming OD[3:0] in the SECCTL register \*/

 DDR3APLLCTL0 &= OUTPUT\_DIVIDE\_MASK; // clear the OD bit field

 DDR3APLLCTL0 |= ~OUTPUT\_DIVIDE\_MASK & (OD\_val - 1) << OUTPUT\_DIVIDE\_OFFSET; // set the OD[3:0] bit field of PLLD to OD\_val

 /\* Set the Multipler values \*/

 DDR3APLLCTL0 &= ~(0x0007FFC0);

 DDR3APLLCTL0 |= ((multiplier << 6) & 0x0007FFC0 );

 temp = ((multiplier + 1) >> 1) - 1;

 DDR3APLLCTL0 &= ~(0xFF000000);

 DDR3APLLCTL0 |= ((temp << 24) & 0xFF000000);

 DDR3APLLCTL1 &= ~(0x0000000F);

 DDR3APLLCTL1 |= ((temp >> 8) & 0x0000000F);

 //In DDR3PLLCTL1, write PLLRST = 0 to bring PLL out of reset

 DDR3APLLCTL1 &= ~(0x00002000);

 // Put the PLL in PLL Mode

 DDR3APLLCTL0 &= ~(0x00800000); // ReSet the Bit 23

 GEL\_TextOut( "DDR3 PLL Setup complete, DDR3A clock now running at 666 MHz.\n" );

 //DDR3A PLL setup complete ---------------------------------------

 /\*------------------------- Start PHY Configuration -------------------------------\*/

 //DDR3A\_PGCR1 = 0x0280C487;

 //5.a Program FRQSEL in the PLL Control Register (address offset 0x018).

 DDR3A\_PLLCR = 0x1C000; //Set FRQSEL=00, for ctl\_clk between 335-533MHz

 //5.b. Program WLSTEP=1, IODDRM=1, and ZCKSEL in the PHY General Configuration Register 1 (address offset 0x00C).

 DDR3A\_PGCR1 |= (1 << 2); //WLSTEP = 1

 DDR3A\_PGCR1 &= ~(IODDRM\_MASK);

 DDR3A\_PGCR1 |= (( 1 << 7) & IODDRM\_MASK);

 DDR3A\_PGCR1 &= ~(ZCKSEL\_MASK);

 DDR3A\_PGCR1 |= (( 1 << 23) & ZCKSEL\_MASK);

 //5.c. Program PHY Timing Parameters Register 0-4 (address offset 0x01C - 0x02C).

 DDR3A\_PTR0 = 0x426213CF;

 DDR3A\_PTR1 = 0xCFC712B3;

 // Maintaining default values of Phy Timing Parameters Register 2 in PUB

 DDR3A\_PTR3 = 0x08861A80;//0x072515C2; //0x0B4515C2;//0x18061A80;

 DDR3A\_PTR4 = 0x0C827100;//0x0AAE7100;

 //5.d. Program PDQ, MPRDQ, and BYTEMASK in the DRAM Configuration Register (address offset 0x044).

 // All other fields must be left at their default values.

 DDR3A\_DCR &= ~(PDQ\_MASK); //PDQ = 0

 DDR3A\_DCR &= ~(MPRDQ\_MASK); //MPRDQ = 0

 DDR3A\_DCR &= ~(BYTEMASK\_MASK);

 DDR3A\_DCR |= (( 1 << 10) & BYTEMASK\_MASK);

 DDR3A\_DCR &= ~(NOSRA\_MASK);

 DDR3A\_DCR |= (( 1 << 27) & NOSRA\_MASK);

 //DDR3A\_DCR &= ~(UDIMM\_MASK);

 //DDR3A\_DCR |= (( 1 << 29) & UDIMM\_MASK);

 //5.e. Program DRAM Timing Parameters Register 0-2 (address offset 0x048 - 0x050).

 DDR3A\_DTPR0 = 0x011CBB66;//0x85589975;//0x8558AA55;

 DDR3A\_DTPR1 = 0x12840300;//0x12835A80;//0x12857280;

 DDR3A\_DTPR2 = 0x5002CE00;

 //5.f. Program BL=0, CL, WR, and PD=1 in the Mode Register 0 (address offset 0x054).

 //All other fields must be left at their default values.

 DDR3A\_MR0 = 0x00001C70; //50

 //5.g. Program DIC, RTT, and TDQS in the Mode Register 1 (address offset 0x058).

 //All other fields must be left at their default values.

 DDR3A\_MR1 = 0x00000006;

 //---------------------------------------------------------------------------------------------------------

 //5.h. Program Mode Register 2 (address offset 0x05C).

 // Maintaining default values of Program Mode Register 2

 DDR3A\_MR2 = 0x00000058;

 //5.i. Program DTMPR=1, DTEXD, DTEXG, RANKEN=1 or 3, and RFSHDT=7 in the Data Training Configuration Register (address offset 0x068).

 //All other fields must be left at their default values.

 DDR3A\_DTCR = 0x710035C7; //0x730035C7;

 //5.j. Program tREFPRD=(5\*tREFI/ddr\_clk\_period) in the PHY General Configuration Register 2 (address offset 0x08C).

 //All other fields must be left at their default values.

 DDR3A\_PGCR2 = 0x00F07A12; //NOBUB = 0, FXDLAT = 0

 //DDR3A\_PGCR2 = 0x00F83D09; //NOBUB = 0, FXDLAT = 1

 //Set Impedence Register

 DDR3A\_ZQ0CR1 = 0x0000007B;

 DDR3A\_ZQ1CR1 = 0x0000007B;

 DDR3A\_ZQ2CR1 = 0x0000007B;

 //DDR3A\_ZQ3CR1 = 0x0000005D;

 //6. Re-trigger PHY initialization in DDR PHY through the VBUSP interface.

 //6.a. Program 0x00000033 to the PHY Initialization Register (address offset 0x004) to re-trigger PLL, ZCAL, and DCAL initialization.

 DDR3A\_PIR = 0x00000033;

 //6.b. Poll for IDONE=1 in the PHY General Status Register 0 (address offset 0x010).

 do {

 read\_val = DDR3A\_PGSR0;

 } while ((read\_val&0x00000001) != 0x00000001);

 //---------------------------------------------------------------------------------------------------------

 // 7. Trigger DDR3 initialization and leveling/training in DDR PHY through the VBUSP interface.

 // a. If using a 16-bit wide DDR interface, program DXEN=0 in the DATX8 2-7 General Configuration Registers (address offsets 0x240, 0x280, 0x2C0, 0x300, 0x340, and 0x380) to disable the leveling/training for the upper byte lanes.

 // b. If using a 32-bit wide DDR interface, program DXEN=0 in the DATX8 4-7 General Configuration Registers (address offsets 0x2C0, 0x300, 0x340, and 0x380) to disable the leveling/training for the upper byte lanes.

 // c. If ECC is not required, program DXEN=0 in the DATX8 8 General Configuration Register (address offset 0x3C0) to disable the leveling/training for the ECC byte lane.

 // NOTE: Setup supports 64-bit by default, ECC enable by default.

 //7.d. Program 0x0000XF81 to the PHY Initialization Register (address offset 0x004) to trigger DDR3 initialization and leveling/training sequences

 DDR3A\_PIR = 0x0000FF81; //WLADJ - ON

 //DDR3A\_PIR = 0x00000781; //WLADJ - OFF

 //---------------------------------------------------------------------------------------------------------

 //7.e. Poll for IDONE=1 in the PHY General Status Register 0 (address offset 0x010).

 do {

 read\_val = DDR3A\_PGSR0;

 } while ((read\_val&0x00000001) != 0x00000001);

 /\* End PHY Configuration \*/

 //---------------------------------------------------------------------------------------------------------

 /\* START EMIF INITIALIZATION

 ++++++++++++++++++SDCFG Register Calculation+++++++++++++++++++

 | 31 - 29 | 28 |27 - 25 | 24 | 23 - 22| 21 - 17 |

 |SDRAM\_TYPE|Rsvd|DDR\_TERM| DDQS | DYN\_ODT| Rsvd |

 | 0x011 | 0 | 0x011 | 0x1 | 0x00 | 0x0 |

 | 16-14 |13 - 12 | 11 - 8 | 7 |6 - 5 | 4 | 3 | 2 | 1 - 0 |

 | CWL | NM | CL | Rsvd |IBANK | Rsvd|EBANK| Rsvd|PAGE\_SIZE|

 | 0x11 | 0x00 | 0x1110 | 0x0 | 0x11 | 0x0 | 0 | 0 | 0x10 |

 SDCFG = 0x0110 0011 0010 0010 0011 0011 1011 0010

 SDCFG = 0x6700486A;//0x63223332

 SDRAM\_TYPE = 3

 DDR\_TERM = 3 (RZQ/4 = 1; RZQ/6=3)

 DDQS = 1

 DYN\_ODT = 0

 CWL = 3 (CWL5=0; CWL6=1; CWL7=2; CWL8=3)

 NM = 0 (64-bit=0, 32-bit=1, 16-bit=2)

 CL = 14 (CL5=2; CL6=4; CL7=6; CL8=8; CL9=10; CL10=12; CL11=14)

 IBANK = 3 (8bank)

 EBANK = 0 (0 - pad\_cs\_o\_n[0] , 1 - pad\_cs\_o\_n[1:0])

 PAGE\_SIZE = 2 (1024page-size=2; 2048page-size=3)

 \*/

 /\* Start DDR3A EMIF Configuration \*/

 //8. Configure the EMIF through the VBUSM interface.

 //8.a. Program all EMIF MMR\92s.

 DDR3A\_SDCFG = 0x6200DE62; // 9A62//0x62008C62 ;//0x6600CE62=single rank,0x6600CE6A=dual rank

 DDR3A\_SDTIM1 = 0x166C9455;

 DDR3A\_SDTIM2 = 0x00001D4A;

 DDR3A\_SDTIM3 = 0x321DFF53;

 DDR3A\_SDTIM4 = 0x543F07FF;

 DDR3A\_ZQCFG = 0x70073200;

 //8.b. Program reg\_initref\_dis=0 in the SDRAM Refresh Control Register (address offset 0x10).

 DDR3A\_SDRFC = 0x00001869;

 GEL\_TextOut("DDR3A initialization complete \n");

 /\* End DDR3A EMIF Configuration \*/

}

/\*--------------------------------------------------------------\*/

/\* TCI66x MENU \*/

/\*--------------------------------------------------------------\*/

menuitem "TCI66x Functions";

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \*

 \* NAME

 \* Global\_Default\_Setup

 \*

 \* PURPOSE:

 \* Setup almost everything ready for a new debug session:

 \* DSP modules and EVM board modules.

 \*

 \* USAGE

 \* This routine can be called as:

 \*

 \* Global\_Default\_Setup()

 \*

 \* RETURN VALUE

 \* NONE

 \*

 \* REFERENCE

 \*

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

hotmenu Global\_Default\_Setup()

{

 GEL\_TextOut( "Global Default Setup...\n" );

 Global\_Default\_Setup\_Silent();

 GEL\_TextOut( "Global Default Setup... Done.\n" );

}

hotmenu Reset()

{

 GEL\_Reset();

}

hotmenu InitXMC()

{

 xmc\_setup();

}

hotmenu CORE\_PLL\_INIT\_100MHZ\_to\_614\_28MHz()

{

 Set\_Pll1(1); // call Set\_Pll1 with index = 1 -> 100 MHz to 614.28 MHz operation

}

hotmenu CORE\_PLL\_INIT\_100MHZ\_to\_737\_5MHz()

{

 Set\_Pll1(2); // call Set\_Pll1 with index = 2 -> 100 MHz to 737.5 MHz operation

}

hotmenu CORE\_PLL\_INIT\_100MHZ\_to\_1GHz()

{

 Set\_Pll1(3); // call Set\_Pll1 with index = 3 -> 100 MHz to 1 GHz operation

}

hotmenu CORE\_PLL\_INIT\_100MHZ\_to\_1\_2GHz()

{

 Set\_Pll1(4); // call Set\_Pll1 with index = 4 -> 100 MHz to 1.2 GHz operation

}

hotmenu CORE\_PLL\_INIT\_100MHZ\_to\_1\_35Gz()

{

 Set\_Pll1(5); // call Set\_Pll1 with index = 5 -> 100 MHz to 1.35 GHz operation

}

hotmenu TETRIS\_POWERUP\_AND\_PLL\_INIT\_100MHZ\_to\_1000MHz()

{

 Set\_Tetris\_Pll(1); // 100 MHz to 1.0 GHz operation

}

hotmenu TETRIS\_POWERUP\_AND\_PLL\_INIT\_100MHZ\_to\_1400MHz()

{

 Set\_Tetris\_Pll(2); // 100 MHz to 1.4 GHz operation

}

hotmenu TETRIS\_POWERUP\_AND\_PLL\_INIT\_175MHZ\_to\_1400MHz()

{

 Set\_Tetris\_Pll(3); // 175 MHz to 1.4 GHz operation

}

hotmenu PA\_PLL\_COnfig()

{

 PaPllConfig();

}

hotmenu InitDDR3A\_32bit\_DDR800()

{

 ddr3A\_32bit\_DDR800\_setup();

}

hotmenu InitDDR3A\_32bit\_DDR1066()

{

 ddr3A\_32bit\_DDR1066\_setup();

}

hotmenu InitDDR3A\_32bit\_DDR1333()

{

 ddr3A\_32bit\_DDR1333\_setup();

}

hotmenu InitDDR3A\_32bit\_DDR1600()

{

 ddr3A\_32bit\_DDR1600\_setup();

}

hotmenu InitDDR3A\_64bit\_DDR1600()

{

 ddr3A\_64bit\_DDR1600\_setup();

}

///\* Function to enable CORE PLL observation clock for PLL output \*///

hotmenu ENABLE\_CORE\_PLL\_OBSCLK()

{

 /\* Unlock Chip Level Registers \*/

 KICK0 = KICK0\_UNLOCK;

 KICK1 = KICK1\_UNLOCK;

 /\* set bit 1 to enable power to the CORE PLL observation clock, clear bit 0 to view the CORE PLL observation (output) clock \*/

 OBSCLKCTL |= (1 << 1); /\* set bit 1 to enable power to the observation clock \*/

 OBSCLKCTL &= ~(1 << 0); /\* clear bit 0 to view the CORE PLL clock \*/

 /\* Lock Chip Level Registers \*/

 KICK0 = 0x00000000;

 KICK1 = 0x00000000;

 GEL\_TextOut("CORE PLL observation clock enabled and configured to show CORE PLL output\n");

}

/\* Function to enable DDR PLL observation clock for PLL output \*/

hotmenu ENABLE\_DDR\_PLL\_OBSCLK ()

{

 /\* Unlock Chip Level Registers \*/

 KICK0 = KICK0\_UNLOCK;

 KICK1 = KICK1\_UNLOCK;

 /\* set bit 1 to enable power to the CORE PLL observation clock, clear bit 0 to view the CORE PLL observation (output) clock \*/

 OBSCLKCTL |= (1 << 3); /\* set bit 3 to enable power to the observation clock \*/

 OBSCLKCTL |= (1 << 2); /\* set bit 2 to view the DDR PLL clock \*/

 /\* Lock Chip Level Registers \*/

 //KICK0 = 0x00000000;

 // KICK1 = 0x00000000;

 GEL\_TextOut("DDR PLL observation clock enabled and configured to show DDR PLL output\n");

}

hotmenu ENABLE\_ARM\_PLL\_OBSCLK ()

{

 /\* Unlock Chip Level Registers \*/

 KICK0 = KICK0\_UNLOCK;

 KICK1 = KICK1\_UNLOCK;

 /\* set bit 1 to enable power to the CORE PLL observation clock, clear bit 0 to view the CORE PLL observation (output) clock \*/

 //OBSCLKCTL |= (1 << 3); /\* set bit 3 to enable power to the observation clock \*/

 OBSCLKCTL |= (1 << 6); /\* set bit 2 to view the DDR PLL clock \*/

 /\* Lock Chip Level Registers \*/

 KICK0 = 0x00000000;

 KICK1 = 0x00000000;

 GEL\_TextOut("DDR PLL observation clock enabled and configured to show DDR PLL output\n");

}

hotmenu ENABLE\_PA\_PLL\_OBSCLK ()

{

 /\* Unlock Chip Level Registers \*/

 KICK0 = KICK0\_UNLOCK;

 KICK1 = KICK1\_UNLOCK;

 /\* set bit 1 to enable power to the CORE PLL observation clock, clear bit 0 to view the CORE PLL observation (output) clock \*/

 OBSCLKCTL |= (1 << 4); /\* set bit 3 to enable power to the observation clock \*/

 OBSCLKCTL |= (1 << 5); /\* set bit 2 to view the DDR PLL clock \*/

 /\* Lock Chip Level Registers \*/

 KICK0 = 0x00000000;

 KICK1 = 0x00000000;

 GEL\_TextOut("DDR PLL observation clock enabled and configured to show DDR PLL output\n");

}

hotmenu ddr3A\_write\_read\_test()

{

 //int data\_set[4];

 //= {0xAAAAAAAA, 0x55555555, 0xFFFFFFFF, 0x00000000};

 unsigned int write\_data = 0xAAAAAAAA;

 unsigned int read\_data = 0x0;

 unsigned int errors = 0;

 int dw;

 unsigned int i, mem\_start, mem\_size, mem\_location;

 mem\_start = DDR3A\_BASE\_ADDRESS + (DNUM \* 0x01000000);

 mem\_size = 0x100;

 for(dw=0;dw<4;dw++)

 {

 if (dw == 0) write\_data = 0xAAAAAAAA;

 if (dw == 1) write\_data = 0x55555555;

 if (dw == 2) write\_data = 0xFFFFFFFF;

 if (dw == 3) write\_data = 0x00000000;

 mem\_location = mem\_start;

 GEL\_TextOut( "Memory Test Write Core: %d, Mem Start: 0x%x, Mem Size: 0x%x, value: 0x%x ...\n",,2,,,DNUM,mem\_start,mem\_size,write\_data);

 for(i=0;i<mem\_size;i++)

 {

 \*( unsigned int\* )(mem\_location) = write\_data;

 mem\_location += 4;

 }

 mem\_location = mem\_start;

 GEL\_TextOut( "Memory Test Read Core: %d, Mem Start: 0x%x, Mem Size: 0x%x ...\n",,2,,,DNUM,mem\_start,mem\_size);

 for (i=0;i<mem\_size;i++)

 {

 read\_data = \*( unsigned int\* )(mem\_location);

 if (read\_data != write\_data)

 {

 GEL\_TextOut("DDR3 Data Error: DSP Core: %d, Mem Addr: 0x%x, read: 0x%x, expected: 0x%x \n",,2,,,DNUM,(DDR3\_BASE\_ADDRESS + (i \* 4)),read\_data,write\_data);

 errors++;

 }

 mem\_location += 4;

 }

 if (errors == 0)

 {

 GEL\_TextOut( "Memory Test Done, no errors found.\n" );

 }

 else

 {

 GEL\_TextOut("Memory Test Done, %d errors were encounterd. \n",,2,,,errors);

 }

 }

 GEL\_TextOut( "All Memory Test Completed on core: %d with %d errors.\n",,2,,,DNUM,errors);

}

menuitem "DSP CLOCK Estimation";

#define TIMER\_TSC (1) // The timer used for polling TSCH/TSCL

#define TIMER\_TSC\_POLL\_PERIOD (10) // Every 10 seconds

unsigned int gPollPeriod = TIMER\_TISC\_POLL\_PERIOD;

unsigned int gTSCL = 0;

unsigned int gTSCH = 0; // Global var for holding previous read of TSCL/H

unsigned int gNumberPoll=0; // Number of pulling \*/

unsigned int gLoopCount=0;

hotmenu dspEnableTsc()

{

 //GEL\_TextOut( "dspEnableTsc - write a value to TSCL to enable it\n" );

 if( GEL\_IsHalted() ) {

 TSCL = 0; // A write to TSCL will enable TSC (timestamp counter)

 GEL\_Run();

 } else {

 GEL\_Halt();

 TSCL = 0;

 GEL\_Run();

 }

}

hotmenu dspDumpTsc()

{

 unsigned int tscl, tsch;

 tscl = TSCL; /\* note: need to read TSCL first \*/

 tsch = TSCH;

 GEL\_TextOut( "dspEnableTsc - TSCH=%x, TSCL=%x\n",,,,, tscl, tsch );

}

dspPollTsc()

{

 unsigned int tscl, tsch;

 unsigned long long tsc1, tsc2;

 if( gLoopCount <= gNumberPoll) {

 //GEL\_EnableRealtime();

 GEL\_Halt();

 tscl = TSCL; /\* The read time can be considered as variations \*/

 tsch = TSCH; /\* The read won't cause variation \*/

 //GEL\_DisableRealtime();

 GEL\_Run();

 tsc2 = (((unsigned long long) tsch)<<32) + tscl;

 tsc1 = (((unsigned long long)gTSCH)<<32) + gTSCL;

 gTSCL = tscl;

 gTSCH = tsch;

 //tsc1 = (tsc2-tsc1)/TIMER\_TSC\_POLL\_PERIOD;

 tsc1 = (tsc2-tsc1)/gPollPeriod;

 GEL\_TextOut( "dspPollTsc - [TSCH,TSCL] = [%x, %x], freq=%dhz, i=%d\n",,,,,

 gTSCH, gTSCL, (tsc1), gLoopCount);

 }

 if( gLoopCount>=gNumberPoll ) {

 dspCancelTscTimer();

 } else {

 gLoopCount++;

 }

}

//

// To cancel the Timer - TIMER\_TSC, after using it. Otherwise, it will continue running.

//

hotmenu dspCancelTscTimer()

{

 GEL\_TextOut( "dspCancelTscTimer\n");

 GEL\_CancelTimer( TIMER\_TSC );

}

//

// To poll the DSP clock.

//

dialog dspPollDSPClockFreq(

 pollPeriod "Polling period (sec) - the longer, the more accurate!",

 numberOfPoll "Number of Polls" )

{

 gPollPeriod = pollPeriod;

 GEL\_TextOut( "dspPollDSPClockFreq with - pollPeriod=%dsec, numberOfPoll=%d\n"

 ,,,,, gPollPeriod, numberOfPoll);

 gNumberPoll = numberOfPoll-1;

 gLoopCount = 0;

 dspEnableTsc();

 // Get the initial value of TSC

 //GEL\_EnableRealtime();

 GEL\_Halt();

 gTSCL = TSCL; /\* The read time can be considered as variations \*/

 gTSCH = TSCH; /\* The read won't cause variation \*/

 //GEL\_DisableRealtime();

 GEL\_Run();

 GEL\_SetTimer( gPollPeriod\*1000, TIMER\_TSC, "dspPollTsc()");

}

#define MDIO\_VERSION\_REG (\*(unsigned int \*) (0x24200F00 + 0x00))

/\* mdio\_user\_access\_reg \*/

#define MDIO\_USER\_ACCESS\_REG (\*(unsigned int \*) (0x24200F00 + 0x80))

#define MDIO\_USER\_CONTROL\_REG (\*(unsigned int \*) (0x24200F00 + 0x04))

#define MDIO\_USER\_INT\_MASK\_REG (\*(unsigned int \*) (0x24200F00 + 0x28))

/\* shift and mask for MDIO\_USER\_ACCESS\_REG \*/

#define CSL\_MDIO\_USER\_ACCESS\_REG\_DATA\_MASK (0x0000FFFFu)

#define CSL\_MDIO\_USER\_ACCESS\_REG\_DATA\_SHIFT (0x00000000u)

#define CSL\_MDIO\_USER\_ACCESS\_REG\_DATA\_RESETVAL (0x00000000u)

#define CSL\_MDIO\_USER\_ACCESS\_REG\_PHYADR\_MASK (0x001F0000u)

#define CSL\_MDIO\_USER\_ACCESS\_REG\_PHYADR\_SHIFT (0x00000010u)

#define CSL\_MDIO\_USER\_ACCESS\_REG\_PHYADR\_RESETVAL (0x00000000u)

#define CSL\_MDIO\_USER\_ACCESS\_REG\_REGADR\_MASK (0x03E00000u)

#define CSL\_MDIO\_USER\_ACCESS\_REG\_REGADR\_SHIFT (0x00000015u)

#define CSL\_MDIO\_USER\_ACCESS\_REG\_REGADR\_RESETVAL (0x00000000u)

#define CSL\_MDIO\_USER\_ACCESS\_REG\_ACK\_MASK (0x20000000u)

#define CSL\_MDIO\_USER\_ACCESS\_REG\_ACK\_SHIFT (0x0000001Du)

#define CSL\_MDIO\_USER\_ACCESS\_REG\_ACK\_RESETVAL (0x00000000u)

#define CSL\_MDIO\_USER\_ACCESS\_REG\_WRITE\_MASK (0x40000000u)

#define CSL\_MDIO\_USER\_ACCESS\_REG\_WRITE\_SHIFT (0x0000001Eu)

#define CSL\_MDIO\_USER\_ACCESS\_REG\_WRITE\_RESETVAL (0x00000000u)

#define CSL\_MDIO\_USER\_ACCESS\_REG\_GO\_MASK (0x80000000u)

#define CSL\_MDIO\_USER\_ACCESS\_REG\_GO\_SHIFT (0x0000001Fu)

#define CSL\_MDIO\_USER\_ACCESS\_REG\_GO\_RESETVAL (0x00000000u)

/\*

#define CSL\_FMK(PER\_REG\_FIELD, val) \

 (((val) << CSL\_##PER\_REG\_FIELD##\_SHIFT) & CSL\_##PER\_REG\_FIELD##\_MASK)

#define CSL\_FEXT(reg, PER\_REG\_FIELD) \

 (((reg) & CSL\_##PER\_REG\_FIELD##\_MASK) >> CSL\_##PER\_REG\_FIELD##\_SHIFT)

#endif

 \*/

/\* the Field MaKe macro \*/

#define CSL\_FMK(SHIFT, MASK, val) (((val) << SHIFT) & MASK)

/\* the Field EXTract macro \*/

#define CSL\_FEXT(reg, SHIFT, MASK) (((reg) & MASK) >> SHIFT)

#define PHY\_REG\_PAGE\_ADDR 22

phy\_hwwrite(unsigned short phy\_addr, unsigned short addr, unsigned short val)

{

 unsigned int retVal=1;

 unsigned int i, delay=2000;

 MDIO\_USER\_ACCESS\_REG = CSL\_FMK (CSL\_MDIO\_USER\_ACCESS\_REG\_GO\_SHIFT, CSL\_MDIO\_USER\_ACCESS\_REG\_GO\_MASK, 1u) |

 CSL\_FMK (CSL\_MDIO\_USER\_ACCESS\_REG\_WRITE\_SHIFT, CSL\_MDIO\_USER\_ACCESS\_REG\_WRITE\_MASK, 1) |

 CSL\_FMK (CSL\_MDIO\_USER\_ACCESS\_REG\_REGADR\_SHIFT, CSL\_MDIO\_USER\_ACCESS\_REG\_REGADR\_MASK, addr) |

 CSL\_FMK (CSL\_MDIO\_USER\_ACCESS\_REG\_PHYADR\_SHIFT, CSL\_MDIO\_USER\_ACCESS\_REG\_PHYADR\_MASK, phy\_addr) |

 CSL\_FMK (CSL\_MDIO\_USER\_ACCESS\_REG\_DATA\_SHIFT, CSL\_MDIO\_USER\_ACCESS\_REG\_DATA\_MASK, val);

 while(1)

 {

 retVal=CSL\_FEXT(MDIO\_USER\_ACCESS\_REG, CSL\_MDIO\_USER\_ACCESS\_REG\_GO\_SHIFT, CSL\_MDIO\_USER\_ACCESS\_REG\_GO\_MASK);

 if(retVal == 0)

 break;

 for(i = 0; i < delay; i++); // this delay is much more than required

 }

}

phy\_hwread(unsigned short phy\_addr, unsigned short addr, unsigned short \*val)

{

 MDIO\_USER\_ACCESS\_REG = CSL\_FMK (CSL\_MDIO\_USER\_ACCESS\_REG\_GO\_SHIFT, CSL\_MDIO\_USER\_ACCESS\_REG\_GO\_MASK, 1u) |

 CSL\_FMK (CSL\_MDIO\_USER\_ACCESS\_REG\_REGADR\_SHIFT, CSL\_MDIO\_USER\_ACCESS\_REG\_REGADR\_MASK, addr) |

 CSL\_FMK (CSL\_MDIO\_USER\_ACCESS\_REG\_PHYADR\_SHIFT, CSL\_MDIO\_USER\_ACCESS\_REG\_PHYADR\_MASK, phy\_addr);

 PHYREG0\_waitResultsAck(\*val, ack);

 if(!ack)

 GEL\_TextOut("Reg read error for PHY: %d\n",,,,, phy\_addr);

}

#define PHYREG0\_waitResultsAck( results, ack ) { \

 results = CSL\_FEXT( MDIO\_USER\_ACCESS\_REG, CSL\_MDIO\_USER\_ACCESS\_REG\_DATA\_SHIFT, CSL\_MDIO\_USER\_ACCESS\_REG\_DATA\_MASK); \

 ack = CSL\_FEXT( MDIO\_USER\_ACCESS\_REG, CSL\_MDIO\_USER\_ACCESS\_REG\_ACK\_SHIFT, CSL\_MDIO\_USER\_ACCESS\_REG\_ACK\_MASK); }

phy\_miiread (unsigned short phyaddr, unsigned short page, unsigned int addr, unsigned short \*val)

{

 phy\_hwwrite(phyaddr, PHY\_REG\_PAGE\_ADDR, page);

 phy\_hwread(phyaddr, addr, val);

}

phy\_miiwrite(unsigned short phyaddr, unsigned short page, unsigned short addr, unsigned short val)

{

 phy\_hwwrite(phyaddr, PHY\_REG\_PAGE\_ADDR, page);

 phy\_hwwrite(phyaddr, addr, val);

}

\_setupPhy(unsigned short phyaddr)

{

 unsigned short val;

 GEL\_TextOut("Staring Setup for PHY: %d\n",,,,, phyaddr);

 /\* Register settings as per RN section 3.1 \*/

 phy\_hwwrite(phyaddr, 22, 0x00ff);

 phy\_hwwrite(phyaddr, 17, 0x214B);

 phy\_hwwrite(phyaddr, 16, 0x2144);

 phy\_hwwrite(phyaddr, 17, 0x0C28);

 phy\_hwwrite(phyaddr, 16, 0x2146);

 phy\_hwwrite(phyaddr, 17, 0xB233);

 phy\_hwwrite(phyaddr, 16, 0x214D);

 phy\_hwwrite(phyaddr, 17, 0xCC0C);

 phy\_hwwrite(phyaddr, 16, 0x2159);

 phy\_hwwrite(phyaddr, 22, 0x0000);

 phy\_hwwrite(phyaddr, 9, 0x1800); //Set master

 GEL\_TextOut("Misc register done for PHY: %d\n",,,,, phyaddr);

 /\* Set SGMII to Copper mode Page 18, Reg 20.2:0 = 1 \*/

 phy\_hwwrite(phyaddr, 22, 18);

 phy\_hwwrite(phyaddr, 20, 0x1);

 /\* PHY Reset Page 18, Reg 20.15 = 1 \*/

 phy\_hwwrite(phyaddr, 20, 0x8201);

 phy\_hwwrite(phyaddr, 22, 00);

 GEL\_TextOut("Waiting for copper link up for PHY: %d\n",,,,, phyaddr);

}

hotmenu setupPhy()

{

 GEL\_TextOut("Reading MDIO\_VERSION\_REG.\n");

 GEL\_TextOut("MDIO\_VERSION\_REG: %x\n",,,,, MDIO\_VERSION\_REG);

 MDIO\_USER\_CONTROL\_REG = (1 << 30) | 0xFA;

 MDIO\_USER\_INT\_MASK\_REG = 0x00;

 \_setupPhy(0);

 //\_setupPhy(1);

}