MLC interface driver

Generated by Doxygen 1.8.7

Mon Jun 9 2014 14:43:20

Contents

1 MLC driver main page

Author

Tomas Kosan

Version

1.0

Date

2012-2013

Copyright

GNU Public License v3, see http://www.gnu.org/licenses/quick-guide-gplv3.html

1.1 Introduction

This documentation describes developed driver named MLC_drv, which covers all hardware implemented in development platform MLC interface. It allows user of development platform to start it use quickly without knowing low level design of MLC interface. Most of functionality of driver is shown in MLC_test_main.c see this file for reference and example use of driver.

1.2 Change log:

1.2.1 Changes 5.6.2014:

- Added address constants for external FPGA PWM module firmware.

1.2.2 Changes 16.4.2014:

- added ADC init procedure to MLC_init() function.

1.2.3 Changes 21.03.2014:

- added a new structure mlc_adc_result, which is suitable for overlap array of results from ADC. Then it is possible access each value by real index of input as defined in schematics.

1.2.4 Changes 13.11.2013:

- LS3137 now supports full access to CPLD, FPGA and peripherals (ADC, DAC) on MLC interface
- LS3137 can access external SRAM memory
- MLC_ functions parameters and return values definitions was rewritten to fixed widths, i.e. unsigned int is now replaced by uint16_t. This change solves differences between C2000 and R4F platform integer sizes. Using types with exact number of bits is highly recommended.

1.2.5 Changes 14.10.2013:

- added almost full support for TMS320F2812, except DMA transfers (not supported on 2812) and ADC conversion end interrupt.
- implemented basic support for TMS570LS3137
- there are now three projects, each for one platform, these depends on each other it means that all of them should be imported to workspace !

1.2.6 Changes 20.8.2013:

- added functions MLC_DBG_set() and MLC_DBG_clear() see Main macros/functions of driver.
- BTN_xx macros renamed to CHECK_BTNxx()
- removed flags DMACH1 - DMACH3 user must now implicitly enable each interrupt from DMA by function MLC_DMA_enable_isr()
- flag USE_DMA changed to USE_ADC_DMA

1.2.7 Changes 16.8.2013:

- SCI driver has extended functionality + added support for FIFO, see SCI documentation.
- MLC_driver was moved to stand-alone CCS project, this behavior allows us to use it as a library
- also TI sources for 28335 peripherals was moved to project tree of MLC_driver and they are linked to library, therefore you should not use it in project ! See folders includeTI and src, which files are included. However it can be used as part of project, then just copy MLC_drv.c and .h to your project and add also TI sources.

1.2.8 Changes 1.6.2013:

- simple SCI driver is now part of MLC_drv, see SCI documentation.

1.2.9 Changes 21.5.2013:

- DAC driver is now part of MLC_drv, see DAC documentation.
- LCD driver is now part of MLC_drv, see LCD documentation.
- new debug macros, see DEBUG documentation.
- full support for DMA transfers and interrupts from AD converters, see ADC documentation.
- added support for external RAM, it can be size of 128kB (compatible with eZdsp28335) or 256kB, however then button S7 will not work, see documentation.
- added support for global enable signal, useful for CPLD/FPGA to detect DSP reset condition. Requires CPLD firmware at least 0.4. See documentation.

1.3 Using driver in as part of your project

1.3.1 Step 1: Install driver

Driver uses four files, MLC_drv.h, MLC_drv_config.h, platform.h and MLC_drv.c. Add them to your project, include header file MLC_drv.h and you are done. However driver depends on TI's DSP2833x_Device.h and DSP2833x↩ _Examples.h or DSP281x_Device.h and DSP281x_Examples.h or ARM headers and sources, depending on DSP used. Those have to be added to your project as well. File platform.h is used to define type of MCU used in project. It must be set properly, otherwise project will not compile.

1.3.2 Step 2: Minimal startup code

Function MLC_init() must be called first, before any other function of MLC driver. Than we can use rest of functions and macros of driver to configure and access peripherals available on MLC interface board.

1.4 Using driver as library

1.4.1 Step 1: Install driver library - RECOMENDED

Driver is standalone CCS static library project, but it is splitted to three supported platforms: MLC_interface_lib, MLC_interface_lib_2812 and MLC_interface_lib_LS3137. This behavior is needed because each platform must have special settings for compiler etc. The MLC_interface_lib project is basic implementation of driver and other two depends on it.

This project has to be imported to your workspace and should have name MLC_interface_lib. MLC_interface_lib is suitable for TMS320F28335 development and does not have any additional depencies.

Recommended way of using driver libs is: 1/ download sources of demo project and proper library from SVN.

- for TMS320F28335: MLC_test_v2 and MLC_interface_lib
- for TMS320F2812: MLC_test_v2_2812, MLC_interface_lib_2812 and MLC_interface_lib
- for TMS570LS3137: MLC_test_v2_LS3137, MLC_interface_lib_LS3137 and MLC_interface_lib

Then import all projects to workspace, let the CCS to copy them to your workspace, otherwise you risk, that project will be changing during time as a new versions will be commited to SVN. All three demos and libraries are setup to work out of the box after sucessfull import to workspace.

If somethings goes wrong below is shown setup of demo project for TMS320F28335. Project for TMS320F28335 which wants use this library driver must have these settings: Simplest way how to use library driver is to create copy of MLC_test_v2 project under different name and replace its source with your sources.

1.4.2 Configuration of driver

Drivers configuration is held in file MLC_drv_config.h. This file consist of defines which controls driver behavior.

1.4.3 Quick start

Documentation of functions are grouped by purpose in Modules tab, brief documentation can be found here. You can quickly access documentation for:

- Main macros and functions here.
- AD converters here.
- DA converter here.
- LCD display here.
- External addresses here.
- Debug utils here.
- SCI functions are here

1.4.4 TODO:

Improve support for TMS570LS3137, add support for RICE BSL system for Hercules MCU.

2 Module Index

2.1 Modules

Here is a list of all modules:

3 Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

4 File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

5 Module Documentation

5.1 Constants of driver

Macros

- #define MCU_TYPE_TMS320F28335 1
- #define MCU_TYPE_TMS320F2812 2
- #define MCU_TYPE_TMS570LS3137 3
- #define MCU_TYPE MCU_TYPE_TMS320F28335
- #define PCB_REV_1 0x1
- #define PCB_REV_2 0x2
- #define PCB_REV_3 0x3

5.1.1 Detailed Description

5.1.2 Macro Definition Documentation

5.1.2.1 #define MCU_TYPE MCU_TYPE_TMS320F28335

Define which microcontroler is used in project

Definition at line 19 of file platform.h.

5.1.2.2 #define MCU_TYPE_TMS320F2812 2

Constant flag defining TMS320F2812

Definition at line 16 of file platform.h.

5.1.2.3 #define MCU_TYPE_TMS320F28335 1

Constant flag defining TMS320F28335

Definition at line 15 of file platform.h.

5.1.2.4 #define MCU_TYPE_TMS570LS3137 3

Constant flag defining TMS570LS3137

Definition at line 17 of file platform.h.

5.1.2.5 #define PCB_REV_1 0x1

Constant with value of PCB revision. This constant is used to enable/disable GPIO function depending on revision of PCB.

Definition at line 384 of file MLC_drv.h.

Referenced by MLC_ADC_start_conv(), and MLC_ADC_start_one_conv().

5.1.2.6 #define PCB_REV_2 0x2

Constant with value of PCB revision. This constant is used to enable/disable GPIO function depending on revision of PCB.

Definition at line 385 of file MLC_drv.h.

5.1.2.7 #define PCB_REV_3 0x3

Constant with value of PCB revision. This constant is used to enable/disable GPIO function depending on revision of PCB.

Definition at line 386 of file MLC_drv.h.

5.2 Definition of driver's types

Data Structures

- struct mlc_info_t
- struct mlc_adc_result

5.2.1 Detailed Description

5.3 Definition of global variables

Variables

- mlc_info_t * pMLC_info_struct
- mlc_info_t **MLC_info_struct**

5.3.1 Detailed Description

5.3.2 Variable Documentation

5.3.2.1 mlc_info_t* pMLC_info_struct

Pointer to internal MLC_info structure, which holds informations about MLC interface.

Definition at line 126 of file MLC_drv.c.

5.4 Definition of addresses of external peripherals

Macros

- #define BASE_ADDRESS 0x00004000
- #define BASE_ADDRESS 0x60000000u
- #define WRITE_AD_CONF 0x0000
- #define WRITE_PWR_OUT 0x0001
- #define WRITE_SETRES_AD 0x0002
- #define WRITE_CLRRES_AD 0x0003
- #define WRITE_DBGLEDS 0x0004
- #define READ_VER 0x0000
- #define READ_AD1 0x0001
- #define READ_AD2 0x0002
- #define READ_AD3 0x0003
- #define READ_HV_INPUT 0x0004
- #define READ_ARC 0x0005
- #define READ_UIO 0x0009
- #define READ_PCB_REV 0x000F
- #define RW_CS_EXT1 0x0006
- #define RW_CS_EXT2 0x0007
- #define RW_CS_EXT3 0x0008
- #define RW_UIO 0x0009
- #define RW_UIO_CONF 0x000A
- #define READ_FPGA_VER 0x0010
- #define RW_FPGA_LED_REG 0x0011
- #define WRITE_LCD_DISP 0x0012
- #define FPGA_RESERVED_13 0x0013
- #define FPGA_RESERVED_14 0x0014
- #define FPGA_RESERVED_15 0x0015
- #define FPGA_RESERVED_16 0x0016
- #define FPGA_FAULTS_ADDR 0x0017
- #define FPGA_RESERVED_18 0x0018
- #define FPGA_RESERVED_19 0x0019
- #define FPGA_RESERVED_1A 0x001A
- #define FPGA_RESERVED_1B 0x001B
- #define FPGA_RESERVED_1C 0x001C
- #define FPGA_RESERVED_1D 0x001D
- #define FPGA_RESERVED_1E 0x001E
- #define FPGA_RESERVED_1F 0x001F
- #define FPGA_LEDS_SHOW_DATA 0x0000
- #define FPGA_LEDS_SHOW_PWMA 0x0100
- #define FPGA_LEDS_SHOW_PWMB 0x0200
- #define FPGA_LEDS_SHOW_PWMC 0x0300
- #define FPGA_LEDS_SHOW_PWMD 0x0400
- #define FPGA_LEDS_SHOW_PWME 0x0500
- #define FPGA_LEDS_SHOW_PWMF 0x0600
- #define EXT_MOD_OFFSET 0x0100
- #define READ_EXT_VER (EXT_MOD_OFFSET + 0x0010)
- #define RW_EXT_LED_REG (EXT_MOD_OFFSET + 0x0011)
- #define EXT_RESERVED_12 (EXT_MOD_OFFSET + 0x0012)
- #define EXT_RESERVED_13 (EXT_MOD_OFFSET + 0x0013)
- #define EXT_RESERVED_14 (EXT_MOD_OFFSET + 0x0014)
- #define EXT_RESERVED_15 (EXT_MOD_OFFSET + 0x0015)
- #define EXT_RESERVED_16 (EXT_MOD_OFFSET + 0x0016)

- #define EXT_FAULTS_ADDR (EXT_MOD_OFFSET + 0x0017)
- #define EXT_RESERVED_18 (EXT_MOD_OFFSET + 0x0018)
- #define EXT_RESERVED_19 (EXT_MOD_OFFSET + 0x0019)
- #define EXT_RESERVED_1A (EXT_MOD_OFFSET + 0x001A)
- #define EXT_RESERVED_1B (EXT_MOD_OFFSET + 0x001B)
- #define EXT_RESERVED_1C (EXT_MOD_OFFSET + 0x001C)
- #define EXT_RESERVED_1D (EXT_MOD_OFFSET + 0x001D)
- #define EXT_RESERVED_1E (EXT_MOD_OFFSET + 0x001E)
- #define EXT_RESERVED_1F (EXT_MOD_OFFSET + 0x001F)

5.4.1 Detailed Description

5.4.2 Macro Definition Documentation

5.4.2.1 #define BASE_ADDRESS 0x00004000

Base address of ZONE0, address 0x0000 from outer world

Base address of EMIF nCS1, address 0x00000000 from outer world

Definition at line 228 of file MLC_drv.h.

Referenced by MLC_DMA_enable_isr(), MLC_read(), and MLC_write().

5.4.2.2 #define BASE_ADDRESS 0x60000000u

Base address of ZONE0, address 0x0000 from outer world

Base address of EMIF nCS1, address 0x00000000 from outer world

Definition at line 228 of file MLC_drv.h.

5.4.2.3 #define EXT_FAULTS_ADDR (EXT_MOD_OFFSET + 0x0017)

Address of FAULT register.

Definition at line 53 of file MLC_ext_module.h.

5.4.2.4 #define EXT_MOD_OFFSET 0x0100

Basic offset of adresses of external FPGA module

Definition at line 34 of file MLC_ext_module.h.

5.4.2.5 #define EXT_RESERVED_12 (EXT_MOD_OFFSET + 0x0012)

Reserved for future use of basic FPGA design.

Definition at line 47 of file MLC_ext_module.h.

5.4.2.6 #define EXT_RESERVED_13 (EXT_MOD_OFFSET + 0x0013)

Reserved for future use of basic FPGA design.

Definition at line 48 of file MLC_ext_module.h.

5.4.2.7 #define EXT_RESERVED_14 (EXT_MOD_OFFSET + 0x0014)

Reserved for future use of basic FPGA design.

Definition at line 49 of file MLC_ext_module.h.

5.4.2.8 #define EXT_RESERVED_15 (EXT_MOD_OFFSET + 0x0015)

Reserved for future use of basic FPGA design.

Definition at line 50 of file MLC_ext_module.h.

5.4.2.9 #define EXT_RESERVED_16 (EXT_MOD_OFFSET + 0x0016)

Reserved for future use of basic FPGA design.

Definition at line 51 of file MLC_ext_module.h.

5.4.2.10 #define EXT_RESERVED_18 (EXT_MOD_OFFSET + 0x0018)

Reserved for future use of basic FPGA design.

Definition at line 55 of file MLC_ext_module.h.

5.4.2.11 #define EXT_RESERVED_19 (EXT_MOD_OFFSET + 0x0019)

Reserved for future use of basic FPGA design.

Definition at line 56 of file MLC_ext_module.h.

5.4.2.12 #define EXT_RESERVED_1A (EXT_MOD_OFFSET + 0x001A)

Reserved for future use of basic FPGA design.

Definition at line 57 of file MLC_ext_module.h.

5.4.2.13 #define EXT_RESERVED_1B (EXT_MOD_OFFSET + 0x001B)

Reserved for future use of basic FPGA design.

Definition at line 58 of file MLC_ext_module.h.

5.4.2.14 #define EXT_RESERVED_1C (EXT_MOD_OFFSET + 0x001C)

Reserved for future use of basic FPGA design.

Definition at line 59 of file MLC_ext_module.h.

5.4.2.15 #define EXT_RESERVED_1D (EXT_MOD_OFFSET + 0x001D)

Reserved for future use of basic FPGA design.

Definition at line 60 of file MLC_ext_module.h.

5.4.2.16 #define EXT_RESERVED_1E (EXT_MOD_OFFSET + 0x001E)

Reserved for future use of basic FPGA design.

Definition at line 61 of file MLC_ext_module.h.

5.4.2.17 #define EXT_RESERVED_1F (EXT_MOD_OFFSET + 0x001F)

Reserved for future use of basic FPGA design.

Definition at line 62 of file MLC_ext_module.h.

5.4.2.18 #define FPGA_FAULTS_ADDR 0x0017

Address of FAULT register.

Definition at line 285 of file MLC_drv.h.

5.4.2.19 #define FPGA_LEDS_SHOW_DATA 0x0000

Constant for setting LED entity to show data from its own register

Definition at line 297 of file MLC_drv.h.

5.4.2.20 #define FPGA_LEDS_SHOW_PWMA 0x0100

Constant for setting LED entity to show data from PWMA port

Definition at line 298 of file MLC_drv.h.

5.4.2.21 #define FPGA_LEDS_SHOW_PWMB 0x0200

Constant for setting LED entity to show data from PWMB port

Definition at line 299 of file MLC_drv.h.

5.4.2.22 #define FPGA_LEDS_SHOW_PWMC 0x0300

Constant for setting LED entity to show data from PWMC port

Definition at line 300 of file MLC_drv.h.

5.4.2.23 #define FPGA_LEDS_SHOW_PWMD 0x0400

Constant for setting LED entity to show data from PWMD port

Definition at line 301 of file MLC_drv.h.

5.4.2.24 #define FPGA_LEDS_SHOW_PWME 0x0500

Constant for setting LED entity to show data from PWME port

Definition at line 302 of file MLC_drv.h.

5.4.2.25 #define FPGA_LEDS_SHOW_PWMF 0x0600

Constant for setting LED entity to show data from PWMF port

Definition at line 303 of file MLC_drv.h.

5.4.2.26 #define FPGA_RESERVED_13 0x0013

Reserved for future use of basic FPGA design.

Definition at line 280 of file MLC_drv.h.

5.4.2.27 #define FPGA_RESERVED_14 0x0014

Reserved for future use of basic FPGA design.

Definition at line 281 of file MLC_drv.h.

5.4.2.28 #define FPGA_RESERVED_15 0x0015

Reserved for future use of basic FPGA design.

Definition at line 282 of file MLC_drv.h.

5.4.2.29 #define FPGA_RESERVED_16 0x0016

Reserved for future use of basic FPGA design.

Definition at line 283 of file MLC_drv.h.

5.4.2.30 #define FPGA_RESERVED_18 0x0018

Reserved for future use of basic FPGA design.

Definition at line 287 of file MLC_drv.h.

5.4.2.31 #define FPGA_RESERVED_19 0x0019

Reserved for future use of basic FPGA design.

Definition at line 288 of file MLC_drv.h.

5.4.2.32 #define FPGA_RESERVED_1A 0x001A

Reserved for future use of basic FPGA design.

Definition at line 289 of file MLC_drv.h.

5.4.2.33 #define FPGA_RESERVED_1B 0x001B

Reserved for future use of basic FPGA design.

Definition at line 290 of file MLC_drv.h.

5.4.2.34 #define FPGA_RESERVED_1C 0x001C

Reserved for future use of basic FPGA design.

Definition at line 291 of file MLC_drv.h.

5.4.2.35 #define FPGA_RESERVED_1D 0x001D

Reserved for future use of basic FPGA design.

Definition at line 292 of file MLC_drv.h.

5.4.2.36 #define FPGA_RESERVED_1E 0x001E

Reserved for future use of basic FPGA design.

Definition at line 293 of file MLC_drv.h.

5.4.2.37 #define FPGA_RESERVED_1F 0x001F

Reserved for future use of basic FPGA design.

Definition at line 294 of file MLC_drv.h.

5.4.2.38 #define READ_AD1 0x0001

Offset of XINTF location of AD CH1 read channel

Definition at line 243 of file MLC_drv.h.

Referenced by MLC_ADC1_read(), MLC_ADC_read(), and MLC_DMA_enable_isr().

5.4.2.39 #define READ_AD2 0x0002

Offset of XINTF location of AD CH2 read channel

Definition at line 244 of file MLC_drv.h.

Referenced by MLC_ADC2_read(), and MLC_ADC_read().

5.4.2.40 #define READ_AD3 0x0003

Offset of XINTF location of AD CH2 read channel

Definition at line 245 of file MLC_drv.h.

Referenced by MLC_ADC3_read(), and MLC_ADC_read().

5.4.2.41 #define READ_ARC 0x0005

Offset of XINTF location of ARC read register

Definition at line 247 of file MLC_drv.h.

5.4.2.42 #define READ_EXT_VER (EXT_MOD_OFFSET + 0x0010)

Offset of XINTF location of FPGA firmware version register of external module

Definition at line 36 of file MLC_ext_module.h.

Referenced by MLC_init().

5.4.2.43 #define READ_FPGA_VER 0x0010

Offset of XINTF location of FPGA firmware version read register

Definition at line 263 of file MLC_drv.h.

5.4.2.44 #define READ_HV_INPUT 0x0004

Offset of XINTF location of HV input read register

Definition at line 246 of file MLC_drv.h.

5.4.2.45 #define READ_PCB_REV 0x000F

Offset of XINTF location of PCB version register

Definition at line 249 of file MLC_drv.h.

5.4.2.46 #define READ_UIO 0x0009

Offset of XINTF location of UNI I/O read register

Definition at line 248 of file MLC_drv.h.

5.4.2.47 #define READ_VER 0x0000

Offset of XINTF location of CPLD firmware version register

Definition at line 242 of file MLC_drv.h.

5.4.2.48 #define RW_CS_EXT1 0x0006

Offset of XINTF location of EXT1 access

Definition at line 252 of file MLC_drv.h.

5.4.2.49 #define RW_CS_EXT2 0x0007

Offset of XINTF location of EXT2 access

Definition at line 253 of file MLC_drv.h.

5.4.2.50 #define RW_CS_EXT3 0x0008

Offset of XINTF location of EXT3 access

Definition at line 254 of file MLC_drv.h.

5.4.2.51 #define RW_EXT_LED_REG (EXT_MOD_OFFSET + 0x0011)

External FPGA LED register offset definition. Register is splitted to two parts: bits 15-8 are configuration, 7-0 are data to show. Configuration data says which data will LEDs show. See FPGA_LEDS... constants. Currently showing of data form lower byte of this register and all PWM gates are supported by FPGA firmware 0.1 and higher.

Definition at line 45 of file MLC_ext_module.h.

Referenced by MLC_init().

5.4.2.52 #define RW_FPGA_LED_REG 0x0011

FPGA LED register offset definition. Register is splitted to two parts: bits 15-8 are configuration, 7-0 are data to show. Configuration data says which data will LEDs show. See FPGA_LEDS... constants. Currently showing of data form lower byte of this register and all PWM gates are supported by FPGA demo firmware 3.2 and higher.

Definition at line 271 of file MLC_drv.h.

5.4.2.53 #define RW_UIO 0x0009

Offset of XINTF location of UIO R/W register

Definition at line 255 of file MLC_drv.h.

5.4.2.54 #define RW_UIO_CONF 0x000A

Offset of XINTF location of UIO R/W configuration register

Definition at line 256 of file MLC_drv.h.

5.4.2.55 #define WRITE_AD_CONF 0x0000

Offset of XINTF location of ADC configuration register

Definition at line 235 of file MLC_drv.h.

Referenced by MLC_ADC_setup(), and MLC_ADC_setup_SOC().

5.4.2.56 #define WRITE_CLRRES_AD 0x0003

Offset of XINTF location of ADC reset signal to set to inactive state

Definition at line 238 of file MLC_drv.h.

Referenced by MLC_ADC_reset().

5.4.2.57 #define WRITE_DBGLEDS 0x0004

Offset of XINTF location of debug LEDs/pins output register

Definition at line 239 of file MLC_drv.h.

Referenced by MLC_DBG_clear(), and MLC_DBG_set().

5.4.2.58 #define WRITE_LCD_DISP 0x0012

FPGA LCD register offset definition. Register is splitted to two parts: high byte is position where character passed as low byte will be printed.Offset of XINTF location of LCD write register

Definition at line 278 of file MLC_drv.h.

Referenced by MLC_LCD_clrscr(), MLC_LCD_write_str(), and MLC_LCD_write_to().

5.4.2.59 #define WRITE_PWR_OUT 0x0001

Offset of XINTF location of PWR out output register

Definition at line 236 of file MLC_drv.h.

Referenced by MLC_PWR_off(), and MLC_PWR_on().

5.4.2.60 #define WRITE_SETRES_AD 0x0002

Offset of XINTF location of ADC reset signal to set to active state

Definition at line 237 of file MLC_drv.h.

Referenced by MLC_ADC_reset().

5.5 Configuration settings and helper functions for ADC

Macros

- #define AD_CH1 0x1
- #define AD_CH2 0x2
- #define AD_CH3 0x4
- #define AD_SOCA 0x1
- #define AD_SOCB 0x2
- #define AD_SOCC 0x4
- #define AD_OS0 0x0000
- #define AD_OS1 0x0001
- #define AD_OS2 0x0002
- #define AD_OS3 0x0003
- #define AD_OS4 0x0004
- #define AD_OS5 0x0005
- #define AD_OS6 0x0006
- #define AD_RNG_5V 0x0000
- #define AD_RNG_10V 0x0001
- #define AD_SOC_ALL_DEFAULT 0x0000
- #define AD_SOC_CH3_SOCA 0x1000
- #define AD_SOC_CH3_SOCB 0x2000
- #define AD_SOC_CH3_SOCAB 0x3000
- #define AD_SOC_ALL_SOCA 0x4000
- #define AD_SOC_ALL_SOCB 0x5000
- #define AD_IN0 0
- #define AD_IN1 2
- #define AD_IN2 4
- #define AD_IN3 6
- #define AD_IN4 8
- #define AD_IN5 10
- #define AD_IN6 12
- #define AD_IN7 14
- #define DMA_ADCH1 AD_CH1
- #define DMA_ADCH2 AD_CH2
- #define DMA_ADCH3 AD_CH3

Functions

- void MLC_ADC_reset (void)
- void MLC_ADC_setup (unsigned int os_1, unsigned int os_2, unsigned int os_3, unsigned int rng_1, unsigned int rng_2, unsigned int rng_3)
- void MLC_ADC_setup_SOC (uint16_t adsoc)
- void MLC_ADC_start_conv (void)
- void MLC_ADC_start_one_conv (uint16_t channel)
- volatile int16_t * MLC_ADC1_read (void)
- volatile int16_t * MLC_ADC2_read (void)
- volatile int16_t * MLC_ADC3_read (void)
- volatile int16_t * MLC_ADC1_get_res_ptr (void)
- volatile int16_t * MLC_ADC2_get_res_ptr (void)
- volatile int16_t * MLC_ADC3_get_res_ptr (void)
- volatile mlc_adc_result * MLC_ADC1_get_res_strptr (void)
- volatile mlc_adc_result * MLC_ADC2_get_res_strptr (void)
- volatile mlc_adc_result * MLC_ADC3_get_res_strptr (void)
- volatile int16_t * MLC_ADC_read (volatile int16_t *ad_res, uint16_t ad_addr)

- unsigned int MLC_ADC_wait (void)
- float MLC_ADC_calc_volt (int16_t value, uint16_t range)
- void MLC_ADC_enable_isr (interrupt void *handler)
- void MLC_ADC_disable_isr (void)
- void MLC_DMA_enable_isr (uint16_t channel, interrupt void *handler)
- void MLC_DMA_disable_isr (uint16_t channel)
- uint16_t MLC_DMA_active (void)
- void MLC_DMA_activate (void)
- void MLC_DMA_deactivate (void)

5.5.1 Detailed Description

5.5.2 Macro Definition Documentation

5.5.2.1 #define AD_CH1 0x1

Constant for selecting ADC CH1.

Definition at line 318 of file MLC_drv.h.

Referenced by MLC_ADC_start_one_conv(), and MLC_DMA_enable_isr().

5.5.2.2 #define AD_CH2 0x2

Constant for selecting ADC CH2.

Definition at line 319 of file MLC_drv.h.

Referenced by MLC_ADC_start_one_conv(), and MLC_DMA_enable_isr().

5.5.2.3 #define AD_CH3 0x4

Constant for selecting ADC CH3.

Definition at line 320 of file MLC_drv.h.

Referenced by MLC_ADC_start_one_conv(), and MLC_DMA_enable_isr().

5.5.2.4 #define AD_IN0 0

Index of MSB data from ADC input 0

Definition at line 365 of file MLC_drv.h.

5.5.2.5 #define AD_IN1 2

Index of MSB data from ADC input 1

Definition at line 366 of file MLC_drv.h.

5.5.2.6 #define AD_IN2 4

Index of MSB data from ADC input 2

Definition at line 367 of file MLC_drv.h.

5.5.2.7 #define AD_IN3 6

Index of MSB data from ADC input 3

Definition at line 368 of file MLC_drv.h.

5.5.2.8 #define AD_IN4 8

Index of MSB data from ADC input 4

Definition at line 369 of file MLC_drv.h.

5.5.2.9 #define AD_IN5 10

Index of MSB data from ADC input 5

Definition at line 370 of file MLC_drv.h.

5.5.2.10 #define AD_IN6 12

Index of MSB data from ADC input 6

Definition at line 371 of file MLC_drv.h.

5.5.2.11 #define AD_IN7 14

Index of MSB data from ADC input 7

Definition at line 372 of file MLC_drv.h.

5.5.2.12 #define AD_OS0 0x0000

Constant defining no oversampling for ADC.

Definition at line 330 of file MLC_drv.h.

Referenced by MLC_init().

5.5.2.13 #define AD_OS1 0x0001

Constant defining 2x oversampling for ADC.

Definition at line 332 of file MLC_drv.h.

5.5.2.14 #define AD_OS2 0x0002

Constant defining 4x oversampling for ADC.

Definition at line 334 of file MLC_drv.h.

5.5.2.15 #define AD_OS3 0x0003

Constant defining 8x oversampling for ADC.

Definition at line 336 of file MLC_drv.h.

5.5.2.16 #define AD_OS4 0x0004

Constant defining 16x oversampling for ADC.

Definition at line 338 of file MLC_drv.h.

5.5.2.17 #define AD_OS5 0x0005

Constant defining 32x oversampling for ADC.

Definition at line 340 of file MLC_drv.h.

5.5.2.18 #define AD_OS6 0x0006

Constant defining 64x oversampling for ADC.

Definition at line 342 of file MLC_drv.h.

5.5.2.19 #define AD_RNG_10V 0x0001

Constant defining +/- 10V input range of ADC.

Definition at line 345 of file MLC_drv.h.

5.5.2.20 #define AD_RNG_5V 0x0000

Constant defining +/- 5V input range of ADC.

Definition at line 344 of file MLC_drv.h.

Referenced by MLC_init().

5.5.2.21 #define AD_SOC_ALL_DEFAULT 0x0000

CH3 will start with toggling of ADSOC pin - no HW SOC support

Definition at line 354 of file MLC_drv.h.

Referenced by MLC_init().

5.5.2.22 #define AD_SOC_ALL_SOCA 0x4000

CH1 + CH2 + CH3 will start with ADSOCA

Definition at line 358 of file MLC_drv.h.

5.5.2.23 #define AD_SOC_ALL_SOCB 0x5000

CH1 + CH2 + CH3 will start with ADSOCB

Definition at line 359 of file MLC_drv.h.

5.5.2.24 #define AD_SOC_CH3_SOCA 0x1000

CH1 + CH3 share ADSOCA, CH2 uses default ADSOCB

Definition at line 355 of file MLC_drv.h.

5.5.2.25 #define AD_SOC_CH3_SOCAB 0x3000

CH3 will start with CH1 or CH2 and uses ADSOCA or ADSOCB in the same time

Definition at line 357 of file MLC_drv.h.

5.5.2.26 #define AD_SOC_CH3_SOCB 0x2000

CH1 uses ADSOCA, CH2 + CH3 share default ADSOCB

Definition at line 356 of file MLC_drv.h.

5.5.2.27 #define AD_SOCA 0x1

Constant for selecting ADC CONVST A.

Definition at line 322 of file MLC_drv.h.

5.5.2.28 #define AD_SOCB 0x2

Constant for selecting ADC CONVST B.

Definition at line 323 of file MLC_drv.h.

5.5.2.29 #define AD_SOCC 0x4

Constant for selecting ADC CONVST C.

Definition at line 324 of file MLC_drv.h.

5.5.2.30 #define DMA_ADCH1 AD_CH1

Constant defining DMA CH1

Definition at line 374 of file MLC_drv.h.

Referenced by MLC_DMA_disable_isr().

5.5.2.31 #define DMA_ADCH2 AD_CH2

Constant defining DMA CH2

Definition at line 375 of file MLC_drv.h.

Referenced by MLC_DMA_disable_isr().

5.5.2.32 #define DMA_ADCH3 AD_CH3

Constant defining DMA CH3

Definition at line 376 of file MLC_drv.h.

Referenced by MLC_DMA_disable_isr().

5.5.3 Function Documentation

5.5.3.1 volatile int16_t* MLC_ADC1_get_res_ptr (void)

Returns pointer to array where measured values are stored, this function will NOT actually read a new data from AD1 !

Returns

address of array with results

Definition at line 1354 of file MLC_drv.c.

5.5.3.2 volatile mlc_adc_result* MLC_ADC1_get_res_strptr (void)

Returns pointer to structure where measured values are stored, this function will NOT actually read a new data from AD1 !

Returns

address of structure mlc_adc_result with results

Definition at line 1366 of file MLC_drv.c.

5.5.3.3 volatile int16_t* MLC_ADC1_read (void)

Reads measured values from ADC No.1 this function triggers full read sequence and update values of array of AD results

Returns

address of array with results

Definition at line 1379 of file MLC_drv.c.

References MLC_ADC_read(), and READ_AD1.

Here is the call graph for this function:

5.5.3.4 volatile int16_t* MLC_ADC2_get_res_ptr (void)

Returns pointer to array where measured values are stored, this function will NOT actually read a new data from AD2 !

Returns

address of array with results

Definition at line 1358 of file MLC_drv.c.

5.5.3.5 volatile mlc_adc_result* MLC_ADC2_get_res_strptr (void)

Returns pointer to structure where measured values are stored, this function will NOT actually read a new data from AD2 !

Returns

address of structure mlc_adc_result with results

Definition at line 1370 of file MLC_drv.c.

5.5.3.6 volatile int16_t* MLC_ADC2_read (void)

Reads measured values from ADC No.2 this function triggers full read sequence and update values of array of AD results

Returns

address of array with results

Definition at line 1384 of file MLC_drv.c.

References MLC_ADC_read(), and READ_AD2.

Here is the call graph for this function:

5.5.3.7 volatile int16_t* MLC_ADC3_get_res_ptr (void)

Returns pointer to array where measured values are stored, this function will NOT actually read a new data from AD3 !

Returns

address of array with results

Definition at line 1362 of file MLC_drv.c.

5.5.3.8 volatile mlc_adc_result* MLC_ADC3_get_res_strptr (void)

Returns pointer to structure where measured values are stored, this function will NOT actually read a new data from AD3 !

Returns

address of structure mlc_adc_result with results

Definition at line 1374 of file MLC_drv.c.

5.5.3.9 volatile int16_t* MLC_ADC3_read (void)

Reads measured values from ADC No.3 this function triggers full read sequence and update values of array of AD results

Returns

address of array with results

Definition at line 1389 of file MLC_drv.c.

References MLC_ADC_read(), and READ_AD3.

Here is the call graph for this function:

5.5.3.10 float MLC_ADC_calc_volt (int16_t *value*, uint16_t *range*)

Helper function, it calculate voltage from measured value.

Parameters

value	measured value from ADC, 16 bits
range	use AD_RNG_5V or AD_RNG_10V

Returns

real value measured by ADC

Definition at line 1393 of file MLC_drv.c.

5.5.3.11 void MLC_ADC_disable_isr (void)

Function for disabling AD end of conversion interrupt.

Definition at line 1043 of file MLC_drv.c.

5.5.3.12 void MLC_ADC_enable_isr (interrupt void * *handler*)

Function for setup of handler function called after conversion of AD ends. The handler has to acknowledge interrupt by command PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

TMS320F28335 - it uses XINT1 - i.e. int1.4. This interrupt should not be used in user code.

Parameters

handler	pointer to function to call when conversion end. It MUST be defined with flag interrupt.

Definition at line 1008 of file MLC_drv.c.

Referenced by MLC_ADC_setup().

Here is the caller graph for this function:

5.5.3.13 volatile int16_t* MLC_ADC_read (volatile int16_t * *ad_res*, uint16_t *ad_addr*)

This function can be used to save some computing time. Use as MLC_read_AD(AD_res_1,READ_AD1);

Parameters

ad_res	pointer to array where to store results. The array must have size of 16 words
ad_addr	address of ADC in memory, use READ_ADx constant.

Returns

pointer to array passed as ad_res

Definition at line 1259 of file MLC_drv.c.

References MLC_READ, READ_AD1, READ_AD2, and READ_AD3.

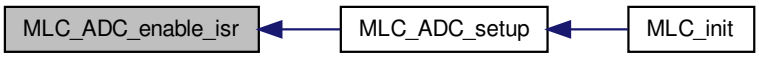
Referenced by MLC_ADC1_read(), MLC_ADC2_read(), and MLC_ADC3_read().

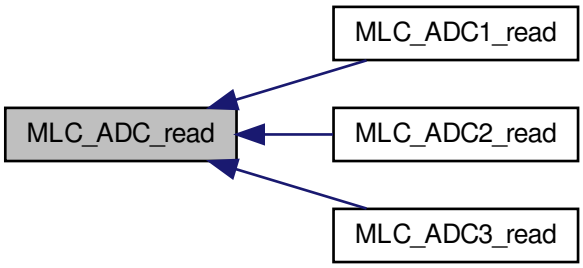
Here is the caller graph for this function:


MLC_ADC1_read
MLC_ADC_read


MLC_ADC2_read
MLC_ADC_read


MLC_ADC3_read
MLC_ADC_read


MLC_ADC_enable_isr
MLC_ADC_setup
MLC_init


MLC_ADC1_read
MLC_ADC_read
MLC_ADC2_read
MLC_ADC3_read

5.5.3.14 void MLC_ADC_reset (void)

This function encapsulate reset process of ADCs.

Definition at line 1061 of file MLC_drv.c.

References DELAY_US(), MLC_WRITE, WRITE_CLRRES_AD, and WRITE_SETRES_AD.

Referenced by MLC_init().

Here is the call graph for this function:

Here is the caller graph for this function:

5.5.3.15 void MLC_ADC_setup (unsigned int *os_1,* unsigned int *os_2,* unsigned int *os_3,* unsigned int *rng_1,* unsigned int *rng_2,* unsigned int *rng_3*)

Function for setting oversampling and input range of all three ADCs.

Parameters

os_1,os_2,os_3	oversampling setup for each channel, use AD_OS0 - AD_OS6. Oversampling will prolong conversion time. I.e. if AD_OS6 is set, then conversion time will be 2^6∗4us = 256us ! All ADC shares one BUSY signal, therefore conversion end is detected after slowest conversion finished.
rng_1,rng_↩ 2,rng_3	set input range, use AD_RNG_5V or AD_RNG_10V

Definition at line 1191 of file MLC_drv.c.

References DELAY_US(), MLC_ADC_enable_isr(), MLC_WRITE, and WRITE_AD_CONF.

Referenced by MLC_init().

Here is the call graph for this function:

Here is the caller graph for this function:

5.5.3.16 void MLC_ADC_setup_SOC (uint16_t *adsoc*)

Function for setting CONVST source pin. Parameter defines which HW pin starts which ADC channel, valid combinations are defined by AD_SOC_∗ constants.

Parameters

adsoc	is one of AD_SOC_∗ constants.

Definition at line 1422 of file MLC_drv.c.

References MLC_WRITE, and WRITE_AD_CONF.

Referenced by MLC_init().

Here is the caller graph for this function:

5.5.3.17 void MLC_ADC_start_conv (void)

Function for manually start conversion of all three channels. In this case settings setted by calling MLC_ADC_↩ setup_SOC() has no meaning. All three channels will be always started.

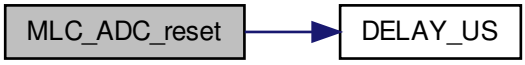
Definition at line 1084 of file MLC_drv.c.

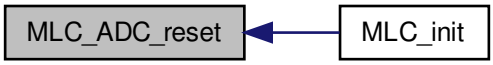
References DELAY_US(), mlc_info_t::pcb_hw_ver, and PCB_REV_1.

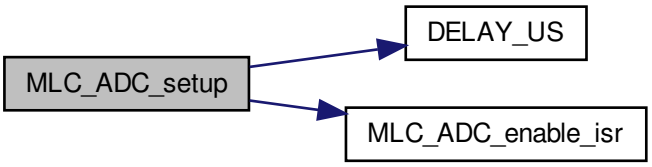
Here is the call graph for this function:

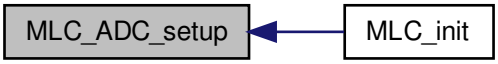
5.5.3.18 void MLC_ADC_start_one_conv (uint16_t *channel*)

Function for manually start conversion of channel. Which ADC(s) will start measure is defined by MLC_ADC_↩ setup_SOC() function.


MLC_ADC_reset
DELAY_US


MLC_ADC_reset
MLC_init


DELAY_US
MLC_ADC_setup
MLC_ADC_enable_isr


MLC_ADC_setup
MLC_init


MLC_ADC_setup_SOC
MLC_init

Parameters

channel	is one of AD_CHx constants.

Definition at line 1122 of file MLC_drv.c.

References AD_CH1, AD_CH2, AD_CH3, DELAY_US(), mlc_info_t::pcb_hw_ver, and PCB_REV_1.

Here is the call graph for this function:

5.5.3.19 unsigned int MLC_ADC_wait (void)

Waits for all AD converters finished conversion and while waiting it blocks.

Returns

number of while cycles spent in blocked state.

Definition at line 1239 of file MLC_drv.c.

5.5.3.20 void MLC_DMA_activate (void)

Activate start of DMA transfer after ADC conversion end.

Definition at line 901 of file MLC_drv.c.

5.5.3.21 uint16_t MLC_DMA_active (void)

Function for checking DMA transfer activity.

Returns

if $>$ 0 then DMA transfer is active

Definition at line 891 of file MLC_drv.c.

5.5.3.22 void MLC_DMA_deactivate (void)

Disable start of DMA transfer after ADC conversion end.

Definition at line 916 of file MLC_drv.c.

5.5.3.23 void MLC_DMA_disable_isr (uint16_t *channel*)

Disables DMA interrupt for specified channel.

Parameters

channel	DMA channel to disable, use one of constant of DMA_ADCHx

Definition at line 972 of file MLC_drv.c.

References DMA_ADCH1, DMA_ADCH2, and DMA_ADCH3.

5.5.3.24 void MLC_DMA_enable_isr (uint16_t *channel*, interrupt void $*$ *handler*)

Enables DMA interrupt and specify user handler. Interrupt handler must acknowledge interrupt at least by command PieCtrlRegs.PIEACK.all = PIEACK_GROUP7;

If DMA transfer is enabled, then it is automatically started after ADC conversion ends.

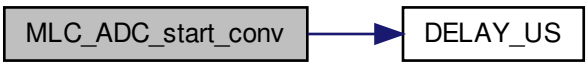
Parameters

channel	DMA channel to enable, use one of constant of DMA_ADCHx
handler	pointer to function to call when data transfer end. It MUST be defined with flag interrupt.

Definition at line 932 of file MLC_drv.c.

References AD_CH1, AD_CH2, AD_CH3, BASE_ADDRESS, dmaConfigCtrlPacket(), and READ_AD1.

Here is the call graph for this function:


MLC_ADC_start_conv
DELAY_US


MLC_ADC_start_one_conv
DELAY_US


MLC_DMA_enable_isr
dmaConfigCtrlPacket

5.6 Helper macros/functions of driver

Macros

- #define CHECK_BTN_S5() !GpioDataRegs.GPADAT.bit.GPIO28
- #define CHECK_BTN_S5() !GpioDataRegs.GPFDAT.bit.GPIOF8
- #define CHECK_BTN_S5() gioGetBit(gioPORTA, 3)
- #define CHECK_BTN_S8() !GpioDataRegs.GPADAT.bit.GPIO29
- #define CHECK_BTN_S8() !GpioDataRegs.GPFDAT.bit.GPIOF9
- #define CHECK_BTN_S8() gioGetBit(gioPORTA, 2)
- #define CHECK_BTN_S6() !GpioDataRegs.GPADAT.bit.GPIO30
- #define CHECK_BTN_S6() !GpioDataRegs.GPFDAT.bit.GPIOF10
- #define CHECK_BTN_S6() gioGetBit(gioPORTA, 4)
- #define CHECK_BTN_S7() 0
- #define CHECK_BTN_S7() !GpioDataRegs.GPFDAT.bit.GPIOF11
- #define CHECK_BTN_S7() gioGetBit(gioPORTA, 0)

Functions

- void DELAY_US (float us)

5.6.1 Detailed Description

5.6.2 Macro Definition Documentation

5.6.2.1 #define CHECK_BTN_S5( ) !GpioDataRegs.GPADAT.bit.GPIO28

This macro returns 1 when button S5 is pressed.

Definition at line 418 of file MLC_drv.h.

5.6.2.2 #define CHECK_BTN_S5( ) !GpioDataRegs.GPFDAT.bit.GPIOF8

This macro returns 1 when button S5 is pressed.

Definition at line 418 of file MLC_drv.h.

5.6.2.3 #define CHECK_BTN_S5( ) gioGetBit(gioPORTA, 3)

This macro returns 1 when button S5 is pressed.

Definition at line 418 of file MLC_drv.h.

5.6.2.4 #define CHECK_BTN_S6( ) !GpioDataRegs.GPADAT.bit.GPIO30

This macro returns 1 when button S6 is pressed.

Definition at line 420 of file MLC_drv.h.

5.6.2.5 #define CHECK_BTN_S6( ) !GpioDataRegs.GPFDAT.bit.GPIOF10

This macro returns 1 when button S6 is pressed.

Definition at line 420 of file MLC_drv.h.

5.6.2.6 #define CHECK_BTN_S6( ) gioGetBit(gioPORTA, 4)

This macro returns 1 when button S6 is pressed.

Definition at line 420 of file MLC_drv.h.

5.6.2.7 #define CHECK_BTN_S7( ) 0

If we want use ext. RAM > 128kB, then it always returns 0.

This macro returns 1 when button S7 is pressed. Collide with ext. RAM > 128kB.

This macro returns 1 when button S7 is pressed.

Definition at line 421 of file MLC_drv.h.

5.6.2.8 #define CHECK_BTN_S7( ) !GpioDataRegs.GPFDAT.bit.GPIOF11

If we want use ext. RAM > 128kB, then it always returns 0.

This macro returns 1 when button S7 is pressed. Collide with ext. RAM > 128kB.

This macro returns 1 when button S7 is pressed.

Definition at line 421 of file MLC_drv.h.

5.6.2.9 #define CHECK_BTN_S7( ) gioGetBit(gioPORTA, 0)

If we want use ext. RAM > 128kB, then it always returns 0.

This macro returns 1 when button S7 is pressed. Collide with ext. RAM > 128kB.

This macro returns 1 when button S7 is pressed.

Definition at line 421 of file MLC_drv.h.

5.6.2.10 #define CHECK_BTN_S8( ) !GpioDataRegs.GPADAT.bit.GPIO29

This macro returns 1 when button S8 is pressed.

Definition at line 419 of file MLC_drv.h.

5.6.2.11 #define CHECK_BTN_S8( ) !GpioDataRegs.GPFDAT.bit.GPIOF9

This macro returns 1 when button S8 is pressed.

Definition at line 419 of file MLC_drv.h.

5.6.2.12 #define CHECK_BTN_S8( ) gioGetBit(gioPORTA, 2)

This macro returns 1 when button S8 is pressed.

Definition at line 419 of file MLC_drv.h.

5.6.3 Function Documentation

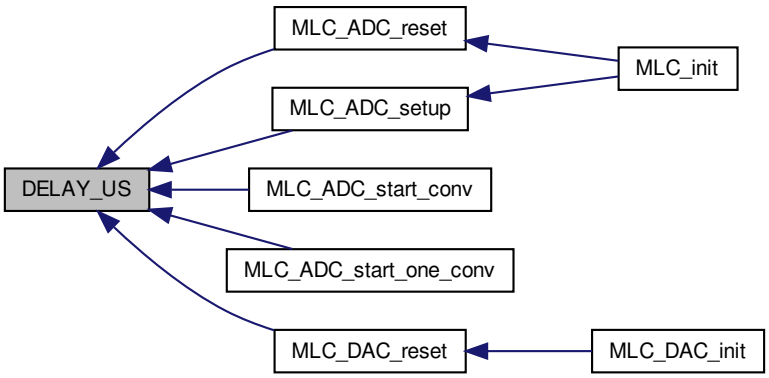
5.6.3.1 void DELAY_US (float *us*)

This function provides simple delay up to 330s with us resolution, however it is not much precise if delay < 10us.

Definition at line 225 of file MLC_drv.c.

Referenced by MLC_ADC_reset(), MLC_ADC_setup(), MLC_ADC_start_conv(), MLC_ADC_start_one_conv(), and MLC_DAC_reset().

Here is the caller graph for this function:


MLC_ADC_reset
MLC_init
MLC_ADC_setup
DELAY_US
MLC_ADC_start_conv
MLC_ADC_start_one_conv
MLC_DAC_reset
MLC_DAC_init

5.7 Main macros/functions of driver

Macros

- #define MLC_WRITE(addr, data) ∗((volatile uint16_t∗)BASE_ADDRESS+addr)=(uint16_t)(data)
- #define MLC_READ(addr) ∗((volatile uint16_t∗)BASE_ADDRESS+addr)
- #define FPGA_check_faults() MLC_READ(FPGA_FAULTS_ADDR)
- #define NO_FAULTS 0x3F

Functions

- void MLC_init ()
- void MLC_global_enable ()
- void MLC_global_disable ()
- void MLC_PWR_on (uint16_t pwr)
- void MLC_PWR_off (uint16_t pwr)
- void MLC_DBG_set (uint16_t dbg)
- void MLC_DBG_clear (uint16_t dbg)
- __inline void MLC_write (uint16_t addr, uint16_t data)
- __inline uint16_t MLC_read (uint16_t addr)

5.7.1 Detailed Description

5.7.2 Macro Definition Documentation

5.7.2.1 #define FPGA_check_faults( ) MLC_READ(FPGA_FAULTS_ADDR)

Macro for reading FAULT inputs from converters.

Definition at line 506 of file MLC_drv.h.

5.7.2.2 #define MLC_READ(*addr*) ∗((volatile uint16_t∗)BASE_ADDRESS+addr)

Macro for reading from XINTF/EMIF connected peripherals.

Parameters

addr	address of external peripheral, use defined constants READ_∗

Returns

read from peripheral

Definition at line 486 of file MLC_drv.h.

Referenced by MLC_ADC_read(), and MLC_init().

5.7.2.3 #define MLC_WRITE(*addr, data*) ∗((volatile uint16_t∗)BASE_ADDRESS+addr)=(uint16_t)(data)

Macro for writing to XINTF/EMIF connected peripherals. CAUTION ! Macro casts it's data parameter to UNSIGNED INT of 16b wide. If design requires writing of SIGNED INT, then data can be UNSIGNED INT without problem, however automatic cast from other data types (i.e. float) will not work. They will be always positive.

Parameters

addr	uint16_t address of external peripheral, use defined constants WRITE_∗
data	uint16_t data to write to peripheral

Definition at line 480 of file MLC_drv.h.

Referenced by MLC_ADC_reset(), MLC_ADC_setup(), MLC_ADC_setup_SOC(), MLC_DBG_clear(), MLC_DBG↩_set(), MLC_init(), MLC_LCD_clrscr(), MLC_LCD_write_str(), MLC_LCD_write_to(), MLC_PWR_off(), and MLC_↩PWR_on().

5.7.2.4 #define NO_FAULTS 0x3F

Constant defining value returned by FPGA_check_faults() when no FAULT is active.

Definition at line 511 of file MLC_drv.h.

5.7.3 Function Documentation

5.7.3.1 void MLC_DBG_clear (uint16_t *dbg*)

Clear debug outputs according to ones in dbg parameter.

Parameters

dbg	defines which debug outputs should be set to 0. I.e. passing 0x01 will turn off debug LED V1.

Definition at line 1791 of file MLC_drv.c.

References MLC_WRITE, and WRITE_DBGLEDS.

5.7.3.2 void MLC_DBG_set (uint16_t *dbg*)

Set debug outputs according to ones in dbg parameter.

Parameters

dbg	defines which debug outputs should be set to 1. I.e. passing 0x01 will turn on debug LED V1.

Definition at line 1785 of file MLC_drv.c.

References MLC_WRITE, and WRITE_DBGLEDS.

5.7.3.3 void MLC_global_disable ( )

Use GPIO60 to disable CPLD and FPGA functionality. Useful for detecting DSP reset via JTAG. Requires CPLD firmware at least 0.4

Definition at line 872 of file MLC_drv.c.

References mlc_info_t::cpld_fw_ver.

5.7.3.4 void MLC_global_enable ( )

Use GPIO60 to enable CPLD and FPGA functionality. Useful for detecting DSP reset via JTAG. Requires CPLD firmware at least 0.4

Definition at line 855 of file MLC_drv.c.

References mlc_info_t::cpld_fw_ver.

5.7.3.5 void MLC_init ( )

This function should be called first, it initializes XINTF ZONE0, ZONE7 (if ext. RAM is used) and fixed purpose GPIOs.

Definition at line 824 of file MLC_drv.c.

References AD_OS0, AD_RNG_5V, AD_SOC_ALL_DEFAULT, DBG_INFO_PUTS, MLC_ADC_reset(), MLC_ADC_setup(), MLC_ADC_setup_SOC(), MLC_READ, MLC_WRITE, mlc_info_t::mod_fw_ver, READ_EXT_VER, and RW_EXT_LED_REG.

Here is the call graph for this function:

5.7.3.6 void MLC_PWR_off (uint16_t *pwr*)

Turn off PWR outputs according to ones in pwr parameter.

Parameters

pwr	defines which outputs should be turn off, in fact this means, that output will be floating, because those power outputs are open collector so turn off means disconnect from ground.

Definition at line 1779 of file MLC_drv.c.

References MLC_WRITE, and WRITE_PWR_OUT.

5.7.3.7 void MLC_PWR_on (uint16_t *pwr*)

Turn on PWR outputs according to ones in pwr parameter.

Parameters

pwr	defines which outputs should be turn on, in fact this means, that output will be zero level. But those power outputs are open collector so turn on means connect to ground.

Definition at line 1773 of file MLC_drv.c.

References MLC_WRITE, and WRITE_PWR_OUT.

5.7.3.8 __inline uint16_t MLC_read (uint16_t *addr*)

Function for reading from XINTF connected peripherals. However using macro MLC_READ() is recommended due higher performance.

Parameters

addr	address of external peripheral, use defined constants READ_∗

Returns

value from periheral

Definition at line 501 of file MLC_drv.h.

References BASE_ADDRESS.

5.7.3.9 __inline void MLC_write (uint16_t *addr*, uint16_t *data*)

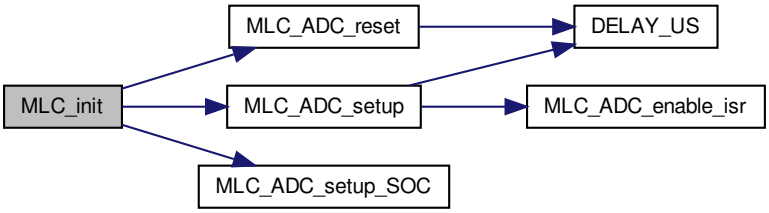
Function for writing to XINTF connected peripherals. However using macro MLC_WRITE() is recommended due higher performance.

Parameters

addr	address of external peripheral, use defined constants WRITE_∗
data	data to write to periheral

Definition at line 495 of file MLC_drv.h.

References BASE_ADDRESS.


MLC_ADC_reset
DELAY_US
MLC_init
MLC_ADC_setup
MLC_ADC_enable_isr
MLC_ADC_setup_SOC

5.8 Utils for LCD display text output.

Functions

- void MLC_LCD_write_to (uint16_t pos, char chr)
- void MLC_LCD_write_str (char *pStr, uint16_t pos)
- void MLC_LCD_clrscr (void)

5.8.1 Detailed Description

5.8.2 Function Documentation

5.8.2.1 void MLC_LCD_clrscr (void)

Clear display.

Definition at line 1415 of file MLC_drv.c.

References MLC_WRITE, and WRITE_LCD_DISP.

5.8.2.2 void MLC_LCD_write_str (char * *pStr,* uint16_t *pos*)

Write string from exact position of LCD display, display entity uses linear addressing.

Parameters

pos	position in range <0,31> where to start
pStr	string to be written

Definition at line 1404 of file MLC_drv.c.

References MLC_WRITE, and WRITE_LCD_DISP.

5.8.2.3 void MLC_LCD_write_to (uint16_t *pos,* char *chr*)

Write one char to exact position of LCD display.

Parameters

pos	position in range <0,31> to write to, display entity uses linear addressing.
chr	character which will be written to position

Definition at line 1397 of file MLC_drv.c.

References MLC_WRITE, and WRITE_LCD_DISP.

5.9 Utils for DAC real time output.

Macros

- #define DAC_CH_A 0
- #define DAC_CH_B 1
- #define DAC_CH_C 2
- #define DAC_CH_D 3
- #define DAC_CH_E 4
- #define DAC_CH_F 5
- #define DAC_CH_G 6
- #define DAC_CH_H 7
- #define dac_reset() MLC_DAC_reset()
- #define dac_control MLC_DAC_control
- #define dac_write MLC_DAC_write
- #define write_dac MLC_DAC_write
- #define dac_init() MLC_DAC_init()
- #define dac_sel_channel dac_sel_ch
- #define dac_variables dac_values

Functions

- void MLC_DAC_reset (void)
- void MLC_DAC_control (uint16_t cmd)
- void MLC_DAC_write (uint16_t count)
- void MLC_DAC_init (void)

Variables

- uint16_t dac_values [8]
- uint16_t dac_sel_ch [8]

5.9.1 Detailed Description

5.9.2 Macro Definition Documentation

5.9.2.1 #define DAC_CH_A 0

Constant for access channel A of DAC.

Definition at line 722 of file MLC_drv.h.

Referenced by MLC_DAC_init().

5.9.2.2 #define DAC_CH_B 1

Constant for access channel B of DAC.

Definition at line 723 of file MLC_drv.h.

Referenced by MLC_DAC_init().

5.9.2.3 #define DAC_CH_C 2

Constant for access channel C of DAC.

Definition at line 724 of file MLC_drv.h.

Referenced by MLC_DAC_init().

5.9.2.4 #define DAC_CH_D 3

Constant for access channel D of DAC.

Definition at line 725 of file MLC_drv.h.

Referenced by MLC_DAC_init().

5.9.2.5 #define DAC_CH_E 4

Constant for access channel E of DAC.

Definition at line 726 of file MLC_drv.h.

Referenced by MLC_DAC_init().

5.9.2.6 #define DAC_CH_F 5

Constant for access channel F of DAC.

Definition at line 727 of file MLC_drv.h.

Referenced by MLC_DAC_init().

5.9.2.7 #define DAC_CH_G 6

Constant for access channel G of DAC.

Definition at line 728 of file MLC_drv.h.

Referenced by MLC_DAC_init().

5.9.2.8 #define DAC_CH_H 7

Constant for access channel H of DAC.

Definition at line 729 of file MLC_drv.h.

Referenced by MLC_DAC_init().

5.9.2.9 #define dac_control **MLC_DAC_control**

Backward compatibility macro

Definition at line 754 of file MLC_drv.h.

5.9.2.10 #define dac_init( ) **MLC_DAC_init()**

Backward compatibility macro

Definition at line 757 of file MLC_drv.h.

5.9.2.11 #define dac_reset( ) **MLC_DAC_reset()**

Backward compatibility macro

Definition at line 753 of file MLC_drv.h.

5.9.2.12 #define dac_sel_channel **dac_sel_ch**

Backward compatibility macro

Definition at line 758 of file MLC_drv.h.

5.9.2.13 #define dac_variables **dac_values**

Backward compatibility macro

Definition at line 759 of file MLC_drv.h.

5.9.2.14 #define dac_write MLC_DAC_write

Backward compatibility macro

Definition at line 755 of file MLC_drv.h.

5.9.2.15 #define write_dac MLC_DAC_write

Backward compatibility macro

Definition at line 756 of file MLC_drv.h.

5.9.3 Function Documentation

5.9.3.1 void MLC_DAC_control (uint16_t *cmd*)

Sends control word to DAC, according to datasheet of AD5328.

Parameters

cmd	word to send as configuration.

Definition at line 1494 of file MLC_drv.c.

5.9.3.2 void MLC_DAC_init (void)

Initializes DAC, setup it according to flags DAC_USE_x. It also set array of channels to default state, where value with index 0 will be send to DAC_CH_A etc. This means, that we do not need to write dac_sel_ch array.

Definition at line 1428 of file MLC_drv.c.

References DAC_CH_A, DAC_CH_B, DAC_CH_C, DAC_CH_D, DAC_CH_E, DAC_CH_F, DAC_CH_G, DAC_↩CH_H, dac_sel_ch, and MLC_DAC_reset().

Here is the call graph for this function:

5.9.3.3 void MLC_DAC_reset (void)

Reset DAC, this must be called before any access to DAC.

Definition at line 1506 of file MLC_drv.c.

References DELAY_US().

Referenced by MLC_DAC_init().

Here is the call graph for this function:

Here is the caller graph for this function:

5.9.3.4 void MLC_DAC_write (uint16_t *count*)

Function to send data from buffer to DAC. It can be defined how many values we need to write out.

Parameters

count	number in range <0,7>. It then send values from dac_values to channel defined in dac_↩sel_ch from 0 to count. I.e. count = 4 sends values from 0 to 4.

Definition at line 1551 of file MLC_drv.c.

References dac_sel_ch, and dac_values.

5.9.4 Variable Documentation

5.9.4.1 uint16_t dac_sel_ch[8]

Array which maps values from dac_values array to DAC outputs

Definition at line 115 of file MLC_drv.c.

Referenced by MLC_DAC_init(), and MLC_DAC_write().

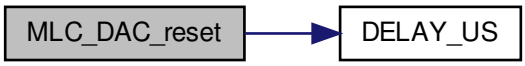
5.9.4.2 uint16_t dac_values[8]

Array which stores values to send to channel defined in dac_sel_ch

Definition at line 114 of file MLC_drv.c.

Referenced by MLC_DAC_write().


MLC_DAC_init
MLC_DAC_reset
DELAY_US


MLC_DAC_reset
DELAY_US


MLC_DAC_reset
MLC_DAC_init

5.10 Utils for simple debug messaging to CIO of CCS.

Macros

- #define DEBUG_LEVEL_NONE 0
- #define DEBUG_LEVEL_ERR 1
- #define DEBUG_LEVEL_WRN 2
- #define DEBUG_LEVEL_INFO 3
- #define DBG_ERR_PUTS(str) puts(str)
- #define DBG_WRN_PUTS(str) ;
- #define DBG_INFO_PUTS(str) ;
- #define DEBUG_LEVEL DEBUG_LEVEL_ERR

5.10.1 Detailed Description

They will output string passed as parameter to CCSs console. These macros depends on <stdio.h> and they have a BIG overhead together with big memory footprint ! They also work only in debug mode and uses one breakpoint. Use at your own risk.

5.10.2 Macro Definition Documentation

5.10.2.1 #define DBG_ERR_PUTS(*str*) puts(str)

Send error str to CCS CIO. Requires <stdio.h> to be included.

Definition at line 791 of file MLC_drv.h.

5.10.2.2 #define DBG_INFO_PUTS(*str*) ;

Send info str to CCS CIO. Requires <stdio.h> to be included.

Definition at line 793 of file MLC_drv.h.

Referenced by MLC_init().

5.10.2.3 #define DBG_WRN_PUTS(*str*) ;

Send warning str to CCS CIO. Requires <stdio.h> to be included.

Definition at line 792 of file MLC_drv.h.

5.10.2.4 #define DEBUG_LEVEL DEBUG_LEVEL_ERR

Sets desired debug level.

Definition at line 119 of file MLC_drv_config.h.

5.10.2.5 #define DEBUG_LEVEL_ERR 1

Constant defining that debug output will print error messages only.

Definition at line 774 of file MLC_drv.h.

5.10.2.6 #define DEBUG_LEVEL_INFO 3

Constant defining that debug output will print all messages.

Definition at line 776 of file MLC_drv.h.

5.10.2.7 #define DEBUG_LEVEL_NONE 0

Constant defining that debug output will not print any messages.

Definition at line 773 of file MLC_drv.h.

5.10.2.8 #define DEBUG_LEVEL_WRN 2

Constant defining that debug output will print error and warning messages.

Definition at line 775 of file MLC_drv.h.

5.11 Utils for USB to SCI

Functions

- void MLC_SCI_init (uint32_t baudrate)
- void MLC_SCI_send_char (char c)
- void MLC_SCI_send_str (char ∗str)
- char MLC_SCI_recv_char (void)
- char MLC_SCI_char_avail (void)
- char MLC_SCI_wait_send (void)
- void MLC_SCI_send_char_wait (char c)

5.11.1 Detailed Description

5.11.2 Function Documentation

5.11.2.1 char MLC_SCI_char_avail (void)

Check if char is available in RX buffer of SCI.

Returns

0 if not, >0 if char is in buffer and can be read. If FIFO is enabled it actually returns number of waiting bytes.

Definition at line 1657 of file MLC_drv.c.

5.11.2.2 void MLC_SCI_init (uint32_t *baudrate*)

It initializes GPIO and baudrate generator.

Parameters

baudrate	required baudrate, for example 115200

5.11.2.3 char MLC_SCI_recv_char (void)

Waits for one char to be received. Blocks until char is received.

Returns

received data byte/char

Definition at line 1694 of file MLC_drv.c.

5.11.2.4 void MLC_SCI_send_char (char *c*)

Send one char/byte via SCI -> USB -> PC. This is blocking function, if TX buffer is not empty, then it waits for TX buffer to be empty. If FIFO enabled, then it can feed upto 16 bytes to FIFO before it blocks another write to FIFO and waits until at least one level is empty.

Parameters

c	char to send

Definition at line 1722 of file MLC_drv.c.

Referenced by MLC_SCI_send_char_wait(), and MLC_SCI_send_str().

Here is the caller graph for this function:

5.11.2.5 void MLC_SCI_send_char_wait (char *c*)

Send one char and waits till it is sent.

Parameters

c	char to send

Definition at line 1747 of file MLC_drv.c.

References MLC_SCI_send_char().

Here is the call graph for this function:

5.11.2.6 void MLC_SCI_send_str (char * *str*)

Sends string via SCI -> USB- > PC. This function is blocking and returns after data is sent.

Parameters

str	pointer to string to send

Definition at line 1764 of file MLC_drv.c.

References MLC_SCI_send_char().

Here is the call graph for this function:

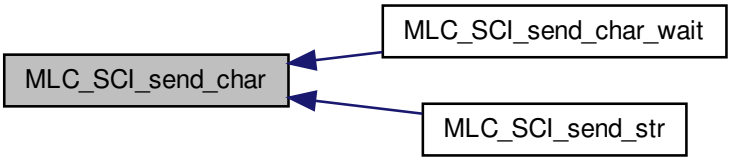
5.11.2.7 char MLC_SCI_wait_send (void)

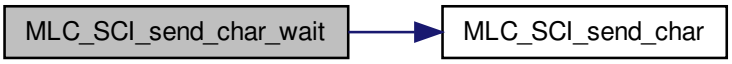
Waits for char to be send.

Returns

1 if char was sent.

Definition at line 1680 of file MLC_drv.c.


MLC_SCI_send_char_wait
MLC_SCI_send_char
MLC_SCI_send_str


MLC_SCI_send_char_wait
MLC_SCI_send_char


MLC_SCI_send_str
MLC_SCI_send_char

5.12 Configuration settings of driver

Macros

- #define USE_ADC_DMA 1
- #define USE_FRSTDATA 0
- #define QUICK_READ 1
- #define ALLOW_XINTF_SPEEDUP 0
- #define FORCE_PCB_REV 0x0
- #define USE_EXT_RAM 1
- #define USE_256kB_RAM 1
- #define DAC_USE_VCC_AS_REF 0
- #define DAC_USE_REF_BUFFER 1
- #define DAC_USE_GAINx2 1
- #define USE_SCI_TX_FIFO 1
- #define USE_SCI_RX_FIFO 1

5.12.1 Detailed Description

5.12.2 Macro Definition Documentation

5.12.2.1 #define ALLOW_XINTF_SPEEDUP 0

Set to 1 to allow lower XINTF timings, i.e. quicker reads and writes. Allow only when CPLD has a new version of FW, at least version 4. MLC_init() however checks version of CPLD firmware and automatically enables this speedup when version is correct. If there is some issues with reading data from ADC (usually swapped low and high word of result), then disabling this feature is recommended.

Definition at line 58 of file MLC_drv_config.h.

5.12.2.2 #define DAC_USE_GAINx2 1

Set output stage gain of DAC to 2x when 1, else 1x. This is useful when Vref is 2,5V and we demand output swing of DAC in range 0 to 5V. When option DAC_USE_VCC_AS_REF is set to 1, then this option has no meaning and gain will be always 1x.

Definition at line 92 of file MLC_drv_config.h.

5.12.2.3 #define DAC_USE_REF_BUFFER 1

Turn on(1)/off(0) internal reference voltage buffers of DAC.

Definition at line 84 of file MLC_drv_config.h.

5.12.2.4 #define DAC_USE_VCC_AS_REF 0

If set to 1 then Vcc = 5V is applied as ref. signal for DAC. Allowing this flag automatically disables DAC_USE_GA↩ INx2 option.

Definition at line 79 of file MLC_drv_config.h.

5.12.2.5 #define FORCE_PCB_REV 0x0

If $>$ 0 then PCB revision is forced to be value of FORCE_REV. Useful only for debug.

Definition at line 63 of file MLC_drv_config.h.

5.12.2.6 #define QUICK_READ 1

Set to 1 to enable quick read, reading of 16 values takes $\sim$1,7us. Set to 0 to save some memory, but prolong reading to $\sim$2,5us. This flag has no meaning when DMA transfers are used.

Definition at line 51 of file MLC_drv_config.h.

5.12.2.7 #define USE_256kB_RAM 1

If set to 1 then it disable BTN_S7, but extend ext. memory size to 256kB, this flag is applicable to eZdsp 28335 kit replacement by T.Kosan only.

Definition at line 74 of file MLC_drv_config.h.

5.12.2.8 #define USE_ADC_DMA 1

Allow DMA for ADC conversion results transfer. This flag ensures proper configuration of DMA transfers. ADC uses channels CH1 - CH3. Interrupt after DMA transfer end has to be individualy enabled for each DMA transfer channel by function MLC_enable_DMA_isr().

Definition at line 39 of file MLC_drv_config.h.

5.12.2.9 #define USE_EXT_RAM 1

Allow using of external RAM, set to 1 to configure XINTF to work with ext. RAM.

Definition at line 68 of file MLC_drv_config.h.

5.12.2.10 #define USE_FRSTDATA 0

Set to 1 to check first data of ADC, usefull when we do not read all data from A/D but it can be tricky, it is better always read all data. This flag has no meaning when DMA transfers are used.

Definition at line 45 of file MLC_drv_config.h.

5.12.2.11 #define USE_SCI_RX_FIFO 1

This option allows using of RX FIFO of SCI. FIFO is 16 bytes deep and can help avoid loosing of data received.

Definition at line 104 of file MLC_drv_config.h.

5.12.2.12 #define USE_SCI_TX_FIFO 1

This option allows using of TX FIFO of SCI. FIFO is 16 bytes deep and can significantly decrease CPU overhead.

Definition at line 98 of file MLC_drv_config.h.

6 Data Structure Documentation

6.1 mlc_adc_result Struct Reference

`#include <MLC_drv.h>`

Data Fields

- int16_t in0
- int16_t in0_low
- int16_t in1
- int16_t in1_low
- int16_t in2
- int16_t in2_low
- int16_t in3
- int16_t in3_low
- int16_t in4
- int16_t in4_low
- int16_t in5
- int16_t in6_low
- int16_t in7
- int16_t in7_low
- int16_t in8
- int16_t in8_low

6.1.1 Detailed Description

Structure which maps array of results to user friendlier structure. It stores 16x16bit wide values, which have 18bit of valid data. High word of data (most significant) is stored in part inX, where X is input number. Last two bits are stored in part inX_low. In most cases inX_low values are ignored by user.

Definition at line 184 of file MLC_drv.h.

6.1.2 Field Documentation

6.1.2.1 int16_t in0

ADC input 0 high word of result.

Definition at line 185 of file MLC_drv.h.

6.1.2.2 int16_t in0_low

ADC input 0 low word of result.

Definition at line 186 of file MLC_drv.h.

6.1.2.3 int16_t in1

ADC input 1 high word of result.

Definition at line 187 of file MLC_drv.h.

6.1.2.4 int16_t in1_low

ADC input 1 low word of result.

Definition at line 188 of file MLC_drv.h.

6.1.2.5 int16_t in2

ADC input 2 high word of result.

Definition at line 189 of file MLC_drv.h.

6.1.2.6 int16_t in2_low

ADC input 2 low word of result.

Definition at line 190 of file MLC_drv.h.

6.1.2.7 int16_t in3

ADC input 3 high word of result.

Definition at line 191 of file MLC_drv.h.

6.1.2.8 int16_t in3_low

ADC input 3 low word of result.

Definition at line 192 of file MLC_drv.h.

6.1.2.9 int16_t in4

ADC input 4 high word of result.

Definition at line 193 of file MLC_drv.h.

6.1.2.10 int16_t in4_low

ADC input 4 low word of result.

Definition at line 194 of file MLC_drv.h.

6.1.2.11 int16_t in5

ADC input 5 high word of result.

Definition at line 195 of file MLC_drv.h.

6.1.2.12 int16_t in6_low

ADC input 5 low word of result.

Definition at line 196 of file MLC_drv.h.

6.1.2.13 int16_t in7

ADC input 6 high word of result.

Definition at line 197 of file MLC_drv.h.

6.1.2.14 int16_t in7_low

ADC input 6 low word of result.

Definition at line 198 of file MLC_drv.h.

6.1.2.15 int16_t in8

ADC input 7 high word of result.

Definition at line 199 of file MLC_drv.h.

6.1.2.16 int16_t in8_low

ADC input 7 low word of result.

Definition at line 200 of file MLC_drv.h.

The documentation for this struct was generated from the following file:

- include/MLC_drv.h

6.2 mlc_info_t Struct Reference

`#include <MLC_drv.h>`

Data Fields

- uint16_t cpld_fw_ver
- uint16_t fpga_fw_ver
- uint16_t pcb_hw_ver
- uint16_t adc_conf
- uint16_t mod_fw_ver

6.2.1 Detailed Description

Structure which holds internal data of MLC driver.

Definition at line 169 of file MLC_drv.h.

6.2.2 Field Documentation

6.2.2.1 uint16_t adc_conf

ADC configuration flags, it is updated each time MLC_ADC_setup() is called.

Definition at line 173 of file MLC_drv.h.

6.2.2.2 uint16_t cpld_fw_ver

CPLD firmware version, it is loaded from CPLD by MLC_init() routine.

Definition at line 170 of file MLC_drv.h.

Referenced by MLC_global_disable(), and MLC_global_enable().

6.2.2.3 uint16_t fpga_fw_ver

FPGA firmware version, it is loaded from FPGA by MLC_init() routine.

Definition at line 171 of file MLC_drv.h.

6.2.2.4 uint16_t mod_fw_ver

Firmware version of FPGA PWM module.

Definition at line 174 of file MLC_drv.h.

Referenced by MLC_init().

6.2.2.5 uint16_t pcb_hw_ver

PCB version, it is loaded from CPLD by MLC_init() routine.

Definition at line 172 of file MLC_drv.h.

Referenced by MLC_ADC_start_conv(), and MLC_ADC_start_one_conv().

The documentation for this struct was generated from the following file:

- include/MLC_drv.h

7 File Documentation

7.1 include/MLC_drv.h File Reference

Header file for MLC driver.

```
#include "stdint.h"
#include "platform.h"
#include "MLC_drv_config.h"
#include "MLC_ext_module.h"
```

Include dependency graph for MLC_drv.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct mlc_info_t
- struct mlc_adc_result

Macros

- #define BASE_ADDRESS 0x00004000
- #define BASE_ADDRESS 0x60000000u
- #define WRITE_AD_CONF 0x0000
- #define WRITE_PWR_OUT 0x0001
- #define WRITE_SETRES_AD 0x0002
- #define WRITE_CLRRES_AD 0x0003
- #define WRITE_DBGLEDS 0x0004
- #define READ_VER 0x0000
- #define READ_AD1 0x0001
- #define READ_AD2 0x0002
- #define READ_AD3 0x0003
- #define READ_HV_INPUT 0x0004
- #define READ_ARC 0x0005
- #define READ_UIO 0x0009
- #define READ_PCB_REV 0x000F
- #define RW_CS_EXT1 0x0006
- #define RW_CS_EXT2 0x0007
- #define RW_CS_EXT3 0x0008
- #define RW_UIO 0x0009
- #define RW_UIO_CONF 0x000A
- #define READ_FPGA_VER 0x0010
- #define RW_FPGA_LED_REG 0x0011
- #define WRITE_LCD_DISP 0x0012
- #define FPGA_RESERVED_13 0x0013

- #define FPGA_RESERVED_14 0x0014
- #define FPGA_RESERVED_15 0x0015
- #define FPGA_RESERVED_16 0x0016
- #define FPGA_FAULTS_ADDR 0x0017
- #define FPGA_RESERVED_18 0x0018
- #define FPGA_RESERVED_19 0x0019
- #define FPGA_RESERVED_1A 0x001A
- #define FPGA_RESERVED_1B 0x001B
- #define FPGA_RESERVED_1C 0x001C
- #define FPGA_RESERVED_1D 0x001D
- #define FPGA_RESERVED_1E 0x001E
- #define FPGA_RESERVED_1F 0x001F
- #define FPGA_LEDS_SHOW_DATA 0x0000
- #define FPGA_LEDS_SHOW_PWMA 0x0100
- #define FPGA_LEDS_SHOW_PWMB 0x0200
- #define FPGA_LEDS_SHOW_PWMC 0x0300
- #define FPGA_LEDS_SHOW_PWMD 0x0400
- #define FPGA_LEDS_SHOW_PWME 0x0500
- #define FPGA_LEDS_SHOW_PWMF 0x0600
- #define AD_CH1 0x1
- #define AD_CH2 0x2
- #define AD_CH3 0x4
- #define AD_SOCA 0x1
- #define AD_SOCB 0x2
- #define AD_SOCC 0x4
- #define AD_OS0 0x0000
- #define AD_OS1 0x0001
- #define AD_OS2 0x0002
- #define AD_OS3 0x0003
- #define AD_OS4 0x0004
- #define AD_OS5 0x0005
- #define AD_OS6 0x0006
- #define AD_RNG_5V 0x0000
- #define AD_RNG_10V 0x0001
- #define AD_SOC_ALL_DEFAULT 0x0000
- #define AD_SOC_CH3_SOCA 0x1000
- #define AD_SOC_CH3_SOCB 0x2000
- #define AD_SOC_CH3_SOCAB 0x3000
- #define AD_SOC_ALL_SOCA 0x4000
- #define AD_SOC_ALL_SOCB 0x5000
- #define AD_IN0 0
- #define AD_IN1 2
- #define AD_IN2 4
- #define AD_IN3 6
- #define AD_IN4 8
- #define AD_IN5 10
- #define AD_IN6 12
- #define AD_IN7 14
- #define DMA_ADCH1 AD_CH1
- #define DMA_ADCH2 AD_CH2
- #define DMA_ADCH3 AD_CH3
- #define PCB_REV_1 0x1
- #define PCB_REV_2 0x2
- #define PCB_REV_3 0x3
- #define CHECK_BTN_S5() !GpioDataRegs.GPADAT.bit.GPIO28

- #define CHECK_BTN_S8() !GpioDataRegs.GPADAT.bit.GPIO29
- #define CHECK_BTN_S6() !GpioDataRegs.GPADAT.bit.GPIO30
- #define CHECK_BTN_S7() 0
- #define CHECK_BTN_S5() !GpioDataRegs.GPFDAT.bit.GPIOF8
- #define CHECK_BTN_S8() !GpioDataRegs.GPFDAT.bit.GPIOF9
- #define CHECK_BTN_S6() !GpioDataRegs.GPFDAT.bit.GPIOF10
- #define CHECK_BTN_S7() !GpioDataRegs.GPFDAT.bit.GPIOF11
- #define CHECK_BTN_S5() gioGetBit(gioPORTA, 3)
- #define CHECK_BTN_S8() gioGetBit(gioPORTA, 2)
- #define CHECK_BTN_S6() gioGetBit(gioPORTA, 4)
- #define CHECK_BTN_S7() gioGetBit(gioPORTA, 0)
- #define MLC_WRITE(addr, data) *((volatile uint16_t*)BASE_ADDRESS+addr)=(uint16_t)(data)
- #define MLC_READ(addr) *((volatile uint16_t*)BASE_ADDRESS+addr)
- #define FPGA_check_faults() MLC_READ(FPGA_FAULTS_ADDR)
- #define NO_FAULTS 0x3F
- #define DAC_CH_A 0
- #define DAC_CH_B 1
- #define DAC_CH_C 2
- #define DAC_CH_D 3
- #define DAC_CH_E 4
- #define DAC_CH_F 5
- #define DAC_CH_G 6
- #define DAC_CH_H 7
- #define dac_reset() MLC_DAC_reset()
- #define dac_control MLC_DAC_control
- #define dac_write MLC_DAC_write
- #define write_dac MLC_DAC_write
- #define dac_init() MLC_DAC_init()
- #define dac_sel_channel dac_sel_ch
- #define dac_variables dac_values
- #define DEBUG_LEVEL_NONE 0
- #define DEBUG_LEVEL_ERR 1
- #define DEBUG_LEVEL_WRN 2
- #define DEBUG_LEVEL_INFO 3
- #define DBG_ERR_PUTS(str) puts(str)
- #define DBG_WRN_PUTS(str) ;
- #define DBG_INFO_PUTS(str) ;

Functions

- void DELAY_US (float us)
- void MLC_init ()
- void MLC_global_enable ()
- void MLC_global_disable ()
- void MLC_PWR_on (uint16_t pwr)
- void MLC_PWR_off (uint16_t pwr)
- void MLC_DBG_set (uint16_t dbg)
- void MLC_DBG_clear (uint16_t dbg)
- __inline void MLC_write (uint16_t addr, uint16_t data)
- __inline uint16_t MLC_read (uint16_t addr)
- void MLC_ADC_reset (void)
- void MLC_ADC_setup (unsigned int os_1, unsigned int os_2, unsigned int os_3, unsigned int rng_1, unsigned int rng_2, unsigned int rng_3)
- void MLC_ADC_setup_SOC (uint16_t adsoc)

- void MLC_ADC_start_conv (void)
- void MLC_ADC_start_one_conv (uint16_t channel)
- volatile int16_t * MLC_ADC1_read (void)
- volatile int16_t * MLC_ADC2_read (void)
- volatile int16_t * MLC_ADC3_read (void)
- volatile int16_t * MLC_ADC1_get_res_ptr (void)
- volatile int16_t * MLC_ADC2_get_res_ptr (void)
- volatile int16_t * MLC_ADC3_get_res_ptr (void)
- volatile mlc_adc_result * MLC_ADC1_get_res_strptr (void)
- volatile mlc_adc_result * MLC_ADC2_get_res_strptr (void)
- volatile mlc_adc_result * MLC_ADC3_get_res_strptr (void)
- volatile int16_t * MLC_ADC_read (volatile int16_t *ad_res, uint16_t ad_addr)
- unsigned int MLC_ADC_wait (void)
- float MLC_ADC_calc_volt (int16_t value, uint16_t range)
- void MLC_ADC_enable_isr (interrupt void *handler)
- void MLC_ADC_disable_isr (void)
- void MLC_DMA_enable_isr (uint16_t channel, interrupt void *handler)
- void MLC_DMA_disable_isr (uint16_t channel)
- uint16_t MLC_DMA_active (void)
- void MLC_DMA_activate (void)
- void MLC_DMA_deactivate (void)
- void MLC_LCD_write_to (uint16_t pos, char chr)
- void MLC_LCD_write_str (char *pStr, uint16_t pos)
- void MLC_LCD_clrscr (void)
- void MLC_DAC_reset (void)
- void MLC_DAC_control (uint16_t cmd)
- void MLC_DAC_write (uint16_t count)
- void MLC_DAC_init (void)
- void MLC_SCI_init (uint32_t baudrate)
- void MLC_SCI_send_char (char c)
- void MLC_SCI_send_str (char *str)
- char MLC_SCI_recv_char (void)
- char MLC_SCI_char_avail (void)
- char MLC_SCI_wait_send (void)
- void MLC_SCI_send_char_wait (char c)

Variables

- mlc_info_t * pMLC_info_struct
- mlc_info_t **MLC_info_struct**
- uint16_t dac_values [8]
- uint16_t dac_sel_ch [8]

7.1.1 Detailed Description

Header file for MLC driver.

The MLC driver is used to control hardware platform named MLC (Multi Level Converter) interface.

It provides all necessary logically named constants to provide easy using of designed hardware.

The driver also provides functions to configure and use all hardware onboard.

Author

Tomas Kosan

Version

0.9

Date

2012-2013

Copyright

GNU Public License v3, see http://www.gnu.org/licenses/quick-guide-gplv3.html

Definition in file MLC_drv.h.

7.2 MLC_drv.h

```
00001
00151 #ifndef MLC_DRV_H_
00152 #define MLC_DRV_H_
00153
00154 #include "stdint.h"
00155 #include "platform.h"
00156 #include "MLC_drv_config.h"
00157 #include "MLC_ext_module.h"
00158
00169 typedef struct{
00170         uint16_t cpld_fw_ver;
00171         uint16_t fpga_fw_ver;
00172         uint16_t pcb_hw_ver;
00173         uint16_t adc_conf;
00174         uint16_t mod_fw_ver;
00175 } mlc_info_t;
00184 typedef struct{
00185   int16_t in0;
00186   int16_t in0_low;
00187   int16_t in1;
00188   int16_t in1_low;
00189   int16_t in2;
00190   int16_t in2_low;
00191   int16_t in3;
00192   int16_t in3_low;
00193   int16_t in4;
00194   int16_t in4_low;
00195   int16_t in5;
00196   int16_t in6_low;
00197   int16_t in7;
00198   int16_t in7_low;
00199   int16_t in8;
00200   int16_t in8_low;
00201 } mlc_adc_result;
00212 extern mlc_info_t* pMLC_info_struct;
00213 extern mlc_info_t MLC_info_struct; /*< Internal MLC_info structure, which holds informations
       about MLC interface. */
00214
00222 #if MCU_TYPE == MCU_TYPE_TMS320F2812 || MCU_TYPE == MCU_TYPE_TMS320F28335
00223 // ZONE0, address 0x0000
00224 #define BASE_ADDRESS 0x00004000
00225 #endif
00226
00227 #if MCU_TYPE == MCU_TYPE_TMS570LS3137
00228 #define BASE_ADDRESS 0x60000000u
00229 #endif
00230
00231 /*--LATEX-CPLD-BLOCK-START--*/
00232 /*****************************************************************************************/
00233 // CPLD has address offset range 0xF
00234 // write addresses constants
00235 #define WRITE_AD_CONF   0x0000
00236 #define WRITE_PWR_OUT   0x0001
00237 #define WRITE_SETRES_AD 0x0002
00238 #define WRITE_CLRRES_AD 0x0003
00239 #define WRITE_DBGLEDS   0x0004
00241 // read addresses constants
00242 #define READ_VER                0x0000
00243 #define READ_AD1                0x0001
00244 #define READ_AD2                0x0002
00245 #define READ_AD3                0x0003
00246 #define READ_HV_INPUT   0x0004
```

```
00247 #define READ_ARC                0x0005
00248 #define READ_UIO                0x0009
00249 #define READ_PCB_REV    0x000F
00251 // read+write addresses constants
00252 #define RW_CS_EXT1              0x0006
00253 #define RW_CS_EXT2              0x0007
00254 #define RW_CS_EXT3              0x0008
00255 #define RW_UIO                  0x0009
00256 #define RW_UIO_CONF             0x000A
00257 /*****************************************************************************************/
00258 /*--LATEX-CPLD-BLOCK-END--*/
00259
00260 /*--LATEX-FPGA-BLOCK-START--*/
00261 /*****************************************************************************************/
00262 // FPGA uses addresses higher than 0xF
00263 #define READ_FPGA_VER   0x0010
00271 #define RW_FPGA_LED_REG 0x0011
00272
00278 #define WRITE_LCD_DISP  0x0012
00280 #define FPGA_RESERVED_13   0x0013
00281 #define FPGA_RESERVED_14   0x0014
00282 #define FPGA_RESERVED_15   0x0015
00283 #define FPGA_RESERVED_16   0x0016
00285 #define FPGA_FAULTS_ADDR   0x0017
00287 #define FPGA_RESERVED_18   0x0018
00288 #define FPGA_RESERVED_19   0x0019
00289 #define FPGA_RESERVED_1A   0x001A
00290 #define FPGA_RESERVED_1B   0x001B
00291 #define FPGA_RESERVED_1C   0x001C
00292 #define FPGA_RESERVED_1D   0x001D
00293 #define FPGA_RESERVED_1E   0x001E
00294 #define FPGA_RESERVED_1F   0x001F
00297 #define FPGA_LEDS_SHOW_DATA             0x0000
00298 #define FPGA_LEDS_SHOW_PWMA             0x0100
00299 #define FPGA_LEDS_SHOW_PWMB             0x0200
00300 #define FPGA_LEDS_SHOW_PWMC             0x0300
00301 #define FPGA_LEDS_SHOW_PWMD             0x0400
00302 #define FPGA_LEDS_SHOW_PWME             0x0500
00303 #define FPGA_LEDS_SHOW_PWMF             0x0600
00305 /*****************************************************************************************/
00306 /*--LATEX-FPGA-BLOCK-END--*/
00307
00317 // AD numbering
00318 #define AD_CH1                  0x1
00319 #define AD_CH2                  0x2
00320 #define AD_CH3                  0x4
00322 #define AD_SOCA                 0x1
00323 #define AD_SOCB                 0x2
00324 #define AD_SOCC                 0x4
00327 // AD config values
00328 // oversampling setup values
00329 // no OS
00330 #define AD_OS0                  0x0000
00331 // 2xOS
00332 #define AD_OS1                  0x0001
00333 // 4xOS
00334 #define AD_OS2                  0x0002
00335 // 8xOS
00336 #define AD_OS3                  0x0003
00337 // 16x OS
00338 #define AD_OS4                  0x0004
00339 // 32xOS
00340 #define AD_OS5                  0x0005
00341 // 64xOS
00342 #define AD_OS6                  0x0006
00343 // range select constants
00344 #define AD_RNG_5V               0x0000
00345 #define AD_RNG_10V              0x0001
00347 // Parameters for MLC_setup_AD_SOC() function
00348 // mapping of convstart pins to ADs
00349 // each AD has it own convstart pin
00350 // CH1 - GPIO32/ADSOCA
00351 // CH2 - GPIO33/ADSOCB
00352 // CH3 - GPIO61 - no ADSOC from EV manager
00353
00354 #define AD_SOC_ALL_DEFAULT              0x0000
00355 #define AD_SOC_CH3_SOCA                 0x1000
00356 #define AD_SOC_CH3_SOCB                 0x2000
00357 #define AD_SOC_CH3_SOCAB                0x3000
00358 #define AD_SOC_ALL_SOCA                 0x4000
00359 #define AD_SOC_ALL_SOCB                 0x5000
00361 // This flag turns on use of EOC interrupt
00362 // It will use XINT1
00363 //#define USE_AD_INTERRUPT      1       /*!< Allow ADC end of conversion interrupt */
00364
00365 #define AD_IN0          0
00366 #define AD_IN1          2
```

```
00367 #define AD_IN2          4
00368 #define AD_IN3          6
00369 #define AD_IN4          8
00370 #define AD_IN5          10
00371 #define AD_IN6          12
00372 #define AD_IN7          14
00374 #define DMA_ADCH1       AD_CH1
00375 #define DMA_ADCH2       AD_CH2
00376 #define DMA_ADCH3       AD_CH3
00384 #define PCB_REV_1               0x1
00385 #define PCB_REV_2               0x2
00386 #define PCB_REV_3               0x3
00395 // define buttons, they are active at 0, data is inverted
00396 // i.e. pushed button means that BTN_Sx returns '1'
00397 #if MCU_TYPE == MCU_TYPE_TMS320F28335
00398         #define CHECK_BTN_S5()          !GpioDataRegs.GPADAT.bit.GPIO28
00399         #define CHECK_BTN_S8()          !GpioDataRegs.GPADAT.bit.GPIO29
00400         #define CHECK_BTN_S6()          !GpioDataRegs.GPADAT.bit.GPIO30
00402         #if USE_256kB_RAM == 1
00403                 #define CHECK_BTN_S7()          0
00404         #else
00405                 #define CHECK_BTN_S7()          !GpioDataRegs.GPADAT.bit.GPIO31
00406         #endif
00407 #endif
00408
00409
00410 #if MCU_TYPE == MCU_TYPE_TMS320F2812
00411 #define CHECK_BTN_S5()          !GpioDataRegs.GPFDAT.bit.GPIOF8
00412 #define CHECK_BTN_S8()          !GpioDataRegs.GPFDAT.bit.GPIOF9
00413 #define CHECK_BTN_S6()          !GpioDataRegs.GPFDAT.bit.GPIOF10
00414 #define CHECK_BTN_S7()          !GpioDataRegs.GPFDAT.bit.GPIOF11
00415 #endif
00416
00417 #if MCU_TYPE == MCU_TYPE_TMS570LS3137
00418 #define CHECK_BTN_S5()          gioGetBit(gioPORTA, 3)
00419 #define CHECK_BTN_S8()          gioGetBit(gioPORTA, 2)
00420 #define CHECK_BTN_S6()          gioGetBit(gioPORTA, 4)
00421 #define CHECK_BTN_S7()          gioGetBit(gioPORTA, 0)
00423 void DELAY_US(float us);
00425 #endif
00426
00427
00440 void MLC_init();
00441
00442 void MLC_global_enable();
00443 void MLC_global_disable();
00450 void MLC_PWR_on(uint16_t pwr);
00451
00457 void MLC_PWR_off(uint16_t pwr);
00458
00463 void MLC_DBG_set(uint16_t dbg);
00464
00469 void MLC_DBG_clear(uint16_t dbg);
00470
00471 // read/write macros - subtitute of inline functions, which do not work properly
00472 // these macros allows to comunicate at rate about 20MB/s
00480 #define MLC_WRITE(addr, data) *((volatile uint16_t*)BASE_ADDRESS+addr)=(uint16_t)(data)
00481
00486 #define MLC_READ(addr) *((volatile uint16_t*)BASE_ADDRESS+addr)
00487
00488 // ordinary functions with same behaviour as macros above
00489 // there are slower (~4MB/s) but, saves some memory
00495 __inline void MLC_write(uint16_t addr, uint16_t data) {*((volatile uint16_t*)(
      BASE_ADDRESS+addr))=(uint16_t)data;}
00501 __inline uint16_t MLC_read(uint16_t addr) {return *((volatile uint16_t*)(
      BASE_ADDRESS+addr));}
00502
00506 #define FPGA_check_faults() MLC_READ(FPGA_FAULTS_ADDR)
00507
00511 #define NO_FAULTS 0x3F
00512
00523 void MLC_ADC_reset(void);
00524
00533 void MLC_ADC_setup(unsigned int os_1, unsigned int os_2, unsigned int os_3, unsigned int rng_1
      , unsigned int rng_2, unsigned int rng_3);
00534
00540 void MLC_ADC_setup_SOC(uint16_t adsoc);
00541
00546 void MLC_ADC_start_conv(void);
00547
00552 void MLC_ADC_start_one_conv(uint16_t channel);
00553
00559 volatile int16_t* MLC_ADC1_read(void);
00560
00566 volatile int16_t* MLC_ADC2_read(void);
00567
00573 volatile int16_t* MLC_ADC3_read(void);
```

```
00574
00579 volatile int16_t* MLC_ADC1_get_res_ptr(void);
00580
00585 volatile int16_t* MLC_ADC2_get_res_ptr(void);
00586
00591 volatile int16_t* MLC_ADC3_get_res_ptr(void);
00592
00597 volatile mlc_adc_result* MLC_ADC1_get_res_strptr(void);
00598
00603 volatile mlc_adc_result* MLC_ADC2_get_res_strptr(void);
00604
00609 volatile mlc_adc_result* MLC_ADC3_get_res_strptr(void);
00610
00617 volatile int16_t* MLC_ADC_read(volatile int16_t* ad_res, uint16_t ad_addr);
00618
00622 unsigned int MLC_ADC_wait(void);
00623
00624 // helper function, it calculate voltage from measured value
00631 float MLC_ADC_calc_volt(int16_t value, uint16_t range);
00632
00639 void MLC_ADC_enable_isr(interrupt void* handler);
00640
00644 void MLC_ADC_disable_isr(void);
00645
00646
00647 // F2812 lacks DMA support
00648 #if MCU_TYPE == MCU_TYPE_TMS320F28335  || MCU_TYPE == MCU_TYPE_TMS570LS3137
00649
00656 void MLC_DMA_enable_isr(uint16_t channel, interrupt void* handler);
00657
00662 void MLC_DMA_disable_isr(uint16_t channel);
00663
00668 uint16_t MLC_DMA_active(void);
00669
00674 void MLC_DMA_activate(void);
00675
00680 void MLC_DMA_deactivate(void);
00681
00682
00685 #endif
00686
00692 // functions for LCD display, FPGA must have basic design to those functions work
00698 void MLC_LCD_write_to(uint16_t pos, char chr);
00704 void MLC_LCD_write_str(char* pStr, uint16_t pos);
00705
00709 void MLC_LCD_clrscr(void);
00710
00720 // constants corresponding
00721 // with PCB marking of ADC outputs
00722 #define DAC_CH_A        0
00723 #define DAC_CH_B        1
00724 #define DAC_CH_C        2
00725 #define DAC_CH_D        3
00726 #define DAC_CH_E        4
00727 #define DAC_CH_F        5
00728 #define DAC_CH_G        6
00729 #define DAC_CH_H        7
00731 extern uint16_t dac_values[8];
00732 extern uint16_t dac_sel_ch[8];
00734 void MLC_DAC_reset(void);
00739 void MLC_DAC_control(uint16_t cmd);
00740
00745 void MLC_DAC_write(uint16_t count);
00751 void MLC_DAC_init(void);
00752
00753 #define dac_reset()                             MLC_DAC_reset()
00754 #define dac_control                                     MLC_DAC_control
00755 #define dac_write                                       MLC_DAC_write
00756 #define write_dac                                       MLC_DAC_write
00757 #define dac_init()                                      MLC_DAC_init()
00758 #define dac_sel_channel                         dac_sel_ch
00759 #define dac_variables                           dac_values
00773 #define DEBUG_LEVEL_NONE        0
00774 #define DEBUG_LEVEL_ERR         1
00775 #define DEBUG_LEVEL_WRN         2
00776 #define DEBUG_LEVEL_INFO        3
00778 #if DEBUG_LEVEL == DEBUG_LEVEL_INFO
00779         #define DBG_ERR_PUTS(str)       puts(str)
00780         #define DBG_WRN_PUTS(str)       puts(str)
00781         #define DBG_INFO_PUTS(str)      puts(str)
00782 #endif
00783
00784 #if DEBUG_LEVEL == DEBUG_LEVEL_WRN
00785         #define DBG_ERR_PUTS(str)       puts(str)
00786         #define DBG_WRN_PUTS(str)       puts(str)
00787         #define DBG_INFO_PUTS(str)      ;
00788 #endif
```

```
00789
00790 #if DEBUG_LEVEL == DEBUG_LEVEL_ERR
00791         #define DBG_ERR_PUTS(str)       puts(str)
00792         #define DBG_WRN_PUTS(str)       ;
00793         #define DBG_INFO_PUTS(str)      ;
00794 #endif
00795
00796 #if DEBUG_LEVEL == DEBUG_LEVEL_NONE
00797         #define DBG_ERR_PUTS(str)       ;
00798         #define DBG_WRN_PUTS(str)       ;
00799         #define DBG_INFO_PUTS(str)      ;
00800 #endif
00801
00813 void MLC_SCI_init(uint32_t baudrate);
00820 void MLC_SCI_send_char(char c);
00826 void MLC_SCI_send_str(char* str);
00831 char MLC_SCI_recv_char(void);
00832
00837 char MLC_SCI_char_avail(void);
00838
00843 char MLC_SCI_wait_send(void);
00844
00849 void MLC_SCI_send_char_wait(char c);
00850
00853 #endif /* end of MLC_drv */
00854
```

7.3 include/MLC_drv_config.h File Reference

Header configuration file for MLC driver.

This graph shows which files directly or indirectly include this file:

Macros

- #define USE_ADC_DMA 1
- #define USE_FRSTDATA 0
- #define QUICK_READ 1
- #define ALLOW_XINTF_SPEEDUP 0
- #define FORCE_PCB_REV 0x0
- #define USE_EXT_RAM 1
- #define USE_256kB_RAM 1
- #define DAC_USE_VCC_AS_REF 0
- #define DAC_USE_REF_BUFFER 1
- #define DAC_USE_GAINx2 1
- #define USE_SCI_TX_FIFO 1
- #define USE_SCI_RX_FIFO 1
- #define DEBUG_LEVEL DEBUG_LEVEL_ERR

7.3.1 Detailed Description

Header configuration file for MLC driver.

The MLC driver is used to control hardware platform named MLC (Multi Level Converter) interface.

It provides all necessary logically named constants to provide easy using of designed hardware.

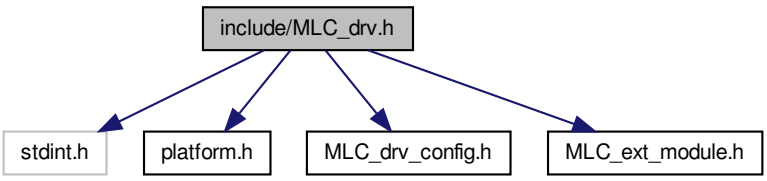
The driver also provides functions to configure and use all hardware onboard.

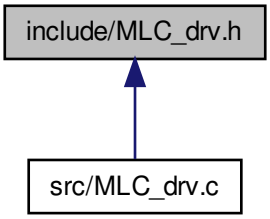
This file carries all configuration switches of MLC_drv. It means that this file is intended to

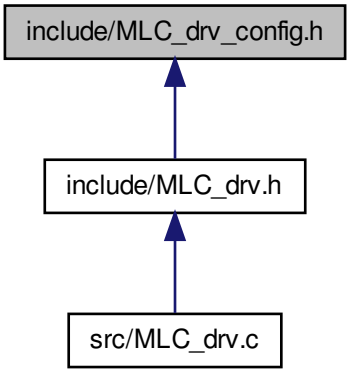
be used by user to fit MLC_drv to specific project.

Author

Tomas Kosan


include/MLC_drv.h
stdint.h
platform.h
MLC_drv_config.h
MLC_ext_module.h


include/MLC_drv.h
src/MLC_drv.c


include/MLC_drv_config.h
include/MLC_drv.h
src/MLC_drv.c

Version

0.1

Date

2012-2013

Copyright

GNU Public License v3, see http://www.gnu.org/licenses/quick-guide-gplv3.html

Definition in file MLC_drv_config.h.

7.4 MLC_drv_config.h

```
00001 /*
00002  * MLC_drv_config.h
00003  *
00004  *  Created on: 19.8.2013
00005  *      Author: Tomas Kosan
00006  */
00007
00027 #ifndef MLC_DRV_CONFIG_H_
00028 #define MLC_DRV_CONFIG_H_
00029
00039 #define USE_ADC_DMA                     1
00040
00045 #define USE_FRSTDATA            0
00046
00051 #define QUICK_READ                      1
00052
00058 #define ALLOW_XINTF_SPEEDUP 0
00059
00063 #define FORCE_PCB_REV           0x0
00064
00068 #define USE_EXT_RAM             1
00069
00074 #define USE_256kB_RAM           1
00075
00079 #define DAC_USE_VCC_AS_REF      0
00080
00084 #define DAC_USE_REF_BUFFER      1
00085
00092 #define DAC_USE_GAINx2          1
00093
00098 #define USE_SCI_TX_FIFO         1
00099
00104 #define USE_SCI_RX_FIFO         1
00105
00119 #define DEBUG_LEVEL                     DEBUG_LEVEL_ERR
00123 #endif /* MLC_DRV_CONFIG_H_ */
```

7.5 include/MLC_ext_module.h File Reference

Header file which adding support for external module.

This graph shows which files directly or indirectly include this file:

Macros

- #define EXT_MOD_OFFSET 0x0100
- #define READ_EXT_VER (EXT_MOD_OFFSET + 0x0010)
- #define RW_EXT_LED_REG (EXT_MOD_OFFSET + 0x0011)
- #define EXT_RESERVED_12 (EXT_MOD_OFFSET + 0x0012)
- #define EXT_RESERVED_13 (EXT_MOD_OFFSET + 0x0013)
- #define EXT_RESERVED_14 (EXT_MOD_OFFSET + 0x0014)
- #define EXT_RESERVED_15 (EXT_MOD_OFFSET + 0x0015)

- #define EXT_RESERVED_16 (EXT_MOD_OFFSET + 0x0016)
- #define EXT_FAULTS_ADDR (EXT_MOD_OFFSET + 0x0017)
- #define EXT_RESERVED_18 (EXT_MOD_OFFSET + 0x0018)
- #define EXT_RESERVED_19 (EXT_MOD_OFFSET + 0x0019)
- #define EXT_RESERVED_1A (EXT_MOD_OFFSET + 0x001A)
- #define EXT_RESERVED_1B (EXT_MOD_OFFSET + 0x001B)
- #define EXT_RESERVED_1C (EXT_MOD_OFFSET + 0x001C)
- #define EXT_RESERVED_1D (EXT_MOD_OFFSET + 0x001D)
- #define EXT_RESERVED_1E (EXT_MOD_OFFSET + 0x001E)
- #define EXT_RESERVED_1F (EXT_MOD_OFFSET + 0x001F)

7.5.1 Detailed Description

Header file which adding support for external module.

The MLC driver is used to control hardware platform named MLC (Multi Level Converter) interface.

It provides all necessary logically named constants to provide easy using of designed hardware.

The driver also provides functions to configure and use all hardware onboard.

This file carries all constants used by basic firmware of external FPGA module.

Author

Tomas Kosan

Version

0.1

Date

2013-2014

Copyright

GNU Public License v3, see http://www.gnu.org/licenses/quick-guide-gplv3.html

Definition in file MLC_ext_module.h.

7.6 MLC_ext_module.h

```
00001 /*
00002  * MLC_ext_module.h
00003  *
00004  *  Created on: 5.6.2014
00005  *      Author: Tomas Kosan
00006  */
00007
00026 #ifndef MLC_EXT_MODULE_H_
00027 #define MLC_EXT_MODULE_H_
00028
00034 #define EXT_MOD_OFFSET   0x0100
00036 #define READ_EXT_VER       (EXT_MOD_OFFSET + 0x0010)
00045 #define RW_EXT_LED_REG    (EXT_MOD_OFFSET + 0x0011)
00046
00047 #define EXT_RESERVED_12   (EXT_MOD_OFFSET + 0x0012)
00048 #define EXT_RESERVED_13   (EXT_MOD_OFFSET + 0x0013)
00049 #define EXT_RESERVED_14   (EXT_MOD_OFFSET + 0x0014)
00050 #define EXT_RESERVED_15   (EXT_MOD_OFFSET + 0x0015)
00051 #define EXT_RESERVED_16   (EXT_MOD_OFFSET + 0x0016)
00053 #define EXT_FAULTS_ADDR   (EXT_MOD_OFFSET + 0x0017)
00055 #define EXT_RESERVED_18   (EXT_MOD_OFFSET + 0x0018)
00056 #define EXT_RESERVED_19   (EXT_MOD_OFFSET + 0x0019)
```

```
00057 #define EXT_RESERVED_1A   (EXT_MOD_OFFSET + 0x001A)
00058 #define EXT_RESERVED_1B   (EXT_MOD_OFFSET + 0x001B)
00059 #define EXT_RESERVED_1C   (EXT_MOD_OFFSET + 0x001C)
00060 #define EXT_RESERVED_1D   (EXT_MOD_OFFSET + 0x001D)
00061 #define EXT_RESERVED_1E   (EXT_MOD_OFFSET + 0x001E)
00062 #define EXT_RESERVED_1F   (EXT_MOD_OFFSET + 0x001F)
00066 #endif /* MLC_EXT_MODULE_H_ */
```

7.7 src/MLC_drv.c File Reference

Source code of MLC driver.

```
#include "platform.h"
#include "MLC_drv.h"
#include <stdint.h>
#include <stdio.h>
#include "DSP2833x_Device.h"
#include "DSP2833x_Examples.h"
#include "DSP281x_Device.h"
#include "DSP281x_Examples.h"
#include "emif.h"
#include "sys_common.h"
#include "system.h"
#include "het.h"
#include "gio.h"
#include "spi.h"
#include "sys_vim.h"
#include "sys_dma.h"
```

Include dependency graph for MLC_drv.c:

Macros

- #define **PIN_STARTCONVA** (uint32)(1U << 21U)
- #define **PIN_STARTCONVB** (uint32)(1U << 20U)
- #define **PIN_STARTCONVC** (uint32)(1U << 22U)
- #define **READ_ADC_ALL** 0xFF
- #define **MLC_READ_32**(addr) *((volatile uint32_t*)BASE_ADDRESS+addr)
- #define **AD_LEN** 16
- #define **TMS570LS3137_CLK** 180000000
- #define **DELAY_1US** 13

Functions

- interrupt void **isr_conv_end** (void)
- void dmaConfigCtrlPacket (uint32 sadd, uint32 dadd, uint32 dsize)
- void **gio_isr_handler** (void)
- interrupt void **dma_transfer_end** (void)
- void DELAY_US (float us)
- void **MLC_init_F28335** (void)
- void **MLC_init_F2812** (void)
- void **MLC_init_LS3137** (void)
- void MLC_init (void)
- void MLC_global_enable ()
- void MLC_global_disable ()
- uint16_t MLC_DMA_active ()
- void MLC_DMA_activate ()
- void MLC_DMA_deactivate ()

- void MLC_DMA_enable_isr (uint16_t channel, interrupt void *handler)
- void MLC_DMA_disable_isr (uint16_t channel)
- void MLC_ADC_enable_isr (interrupt void *handler)
- void MLC_ADC_disable_isr (void)
- void MLC_ADC_reset (void)
- void MLC_ADC_start_conv (void)
- void MLC_ADC_start_one_conv (uint16_t channel)
- void MLC_ADC_setup (unsigned int os_1, unsigned int os_2, unsigned int os_3, unsigned int rng_1, unsigned int rng_2, unsigned int rng_3)
- unsigned int MLC_ADC_wait (void)
- volatile int16_t * MLC_ADC_read (volatile int16_t *ad_res, uint16_t ad_addr)
- volatile int16_t * MLC_ADC1_get_res_ptr (void)
- volatile int16_t * MLC_ADC2_get_res_ptr (void)
- volatile int16_t * MLC_ADC3_get_res_ptr (void)
- volatile mlc_adc_result * MLC_ADC1_get_res_strptr (void)
- volatile mlc_adc_result * MLC_ADC2_get_res_strptr (void)
- volatile mlc_adc_result * MLC_ADC3_get_res_strptr (void)
- volatile int16_t * MLC_ADC1_read (void)
- volatile int16_t * MLC_ADC2_read (void)
- volatile int16_t * MLC_ADC3_read (void)
- float MLC_ADC_calc_volt (int16_t value, uint16_t range)
- void MLC_LCD_write_to (uint16_t pos, char chr)
- void MLC_LCD_write_str (char *pStr, uint16_t pos)
- void MLC_LCD_clrscr (void)
- void MLC_ADC_setup_SOC (uint16_t adsoc)
- void MLC_DAC_init (void)
- void MLC_DAC_control (uint16_t cmd)
- void MLC_DAC_reset (void)
- void MLC_DAC_write (uint16_t count)
- void **MLC_SCI_init** (unsigned long int baudrate)
- char MLC_SCI_char_avail (void)
- char MLC_SCI_wait_send (void)
- char MLC_SCI_recv_char (void)
- void MLC_SCI_send_char (char c)
- void MLC_SCI_send_char_wait (char c)
- void MLC_SCI_send_str (char *str)
- void MLC_PWR_on (uint16_t pwr)
- void MLC_PWR_off (uint16_t pwr)
- void MLC_DBG_set (uint16_t dbg)
- void MLC_DBG_clear (uint16_t dbg)

Variables

- volatile int16_t **AD_dma_res_1** [16]
- volatile int16_t **AD_dma_res_2** [16]
- volatile int16_t **AD_dma_res_3** [16]
- t_isrFuncPTR **adc_convend_handler**
- g_dmaCTRL **g_dmaCTRLPKT**
- spiDAT1_t **spi2_conf**
- volatile int16_t * **AD_res_1** = AD_dma_res_1
- volatile int16_t * **AD_res_2** = AD_dma_res_2
- volatile int16_t * **AD_res_3** = AD_dma_res_3
- volatile mlc_adc_result * **AD_res_1_struct** = (mlc_adc_result*) AD_dma_res_1
- volatile mlc_adc_result * **AD_res_2_struct** = (mlc_adc_result*) AD_dma_res_2

- volatile mlc_adc_result * **AD_res_3_struct** = (mlc_adc_result*) AD_dma_res_3
- uint16_t dac_values [8]
- uint16_t dac_sel_ch [8]
- volatile uint16_t **pwr_out_state** = 0
- volatile uint16_t **dbg_state** = 0
- volatile int temp
- mlc_info_t **MLC_info_struct**
- mlc_info_t * pMLC_info_struct = &MLC_info_struct
- uint16_t **ad_conf** = 0
- uint16_t **ad_soc_conf** = 0

7.7.1 Detailed Description

Source code of MLC driver.

This file implements driver for memory mapped peripherals on MLC interface.

Definition in file MLC_drv.c.

7.7.2 Function Documentation

7.7.2.1 void dmaConfigCtrlPacket (uint32 *sadd,* uint32 *dadd,* uint32 *dsize*)

void dmaConfigCtrlPacket(uint32 sadd,uint32 dadd,uint32 dsize)
configuring dma control packet stack

```
sadd  > source address
dadd  > destination  address
dsize > data size
```

@ note : after configuring the stack the control packet needs to be set by calling dmaSetCtrlPacket()

Definition at line 162 of file MLC_drv.c.

Referenced by MLC_DMA_enable_isr().

Here is the caller graph for this function:

7.7.3 Variable Documentation

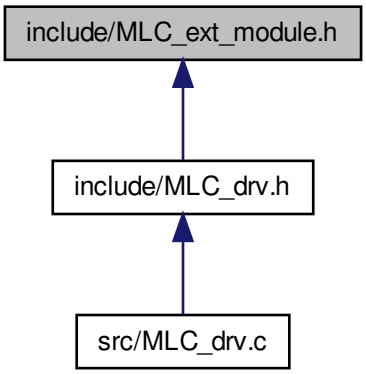
7.7.3.1 volatile int temp

temp int

Definition at line 123 of file MLC_drv.c.

7.8 MLC_drv.c

```
00001
00011 /*
00012  * Platform independent include section
00013  */
00014
00015 #include "platform.h"
00016 #include "MLC_drv.h"
00017 #include <stdint.h>
00018 #include <stdio.h>
00019
00020 /*
00021  * Platform dependent include section
00022  */
00023
00024 #if MCU_TYPE == MCU_TYPE_TMS320F28335
```


include/MLC_ext_module.h
include/MLC_drv.h
src/MLC_drv.c


src/MLC_drv.c
MLC_drv.h
stdio.h
DSP2833x_Device.h
DSP2833x_Examples.h
DSP281x_Device.h
DSP281x_Examples.h
emif.h
sys_common.h
system.h
het.h
gio.h
spi.h
sys_vim.h
sys_dma.h
platform.h
stdint.h
MLC_drv_config.h
MLC_ext_module.h


dmaConfigCtrlPacket
MLC_DMA_enable_isr

```
00025 #include "DSP2833x_Device.h"
00026 #include "DSP2833x_Examples.h"
00027 #endif
00028
00029 #if MCU_TYPE == MCU_TYPE_TMS320F2812
00030 #include "DSP281x_Device.h"
00031 #include "DSP281x_Examples.h"
00032 #endif
00033
00034 #if MCU_TYPE == MCU_TYPE_TMS570LS3137
00035 #include "emif.h"
00036 #include "sys_common.h"
00037 #include "system.h"
00038 #include "het.h"
00039 #include "gio.h"
00040 #include "spi.h"
00041 #include "sys_vim.h"
00042 #include "sys_dma.h"
00043 #endif
00044
00045 /*
00046  * Platform dependent #define section
00047  */
00048
00049 #if MCU_TYPE == MCU_TYPE_TMS570LS3137
00050 #define PIN_STARTCONVA (uint32)(1U << 21U)
00051 #define PIN_STARTCONVB (uint32)(1U << 20U)
00052 #define PIN_STARTCONVC (uint32)(1U << 22U)
00053
00054 #define READ_ADC_ALL    0xFF
00055
00056 #define MLC_READ_32(addr) *((volatile uint32_t*)BASE_ADDRESS+addr)
00057
00058 #endif
00059
00060 /*
00061  * Plaform independent define section
00062  */
00063
00064 // arrays for storing results
00065 #define AD_LEN 16
00066
00067 /*
00068  * Platform dependent global variables section
00069  */
00070
00071 // place ad results to RAM which can be DMA accessed
00072 #if MCU_TYPE == MCU_TYPE_TMS320F28335
00073 #pragma DATA_SECTION(AD_dma_res_1, "DMARAML6")
00074 volatile int16_t AD_dma_res_1[16];
00075 #pragma DATA_SECTION(AD_dma_res_2, "DMARAML6")
00076 volatile int16_t AD_dma_res_2[16];
00077 #pragma DATA_SECTION(AD_dma_res_3, "DMARAML6")
00078 volatile int16_t AD_dma_res_3[16];
00079 #endif
00080
00081 #if MCU_TYPE == MCU_TYPE_TMS320F2812
00082 volatile int16_t AD_dma_res_1[16];
00083 volatile int16_t AD_dma_res_2[16];
00084 volatile int16_t AD_dma_res_3[16];
00085 #endif
00086
00087 #if MCU_TYPE == MCU_TYPE_TMS570LS3137
00088 volatile int16_t AD_dma_res_1[16];
00089 volatile int16_t AD_dma_res_2[16];
00090 volatile int16_t AD_dma_res_3[16];
00091
00092 t_isrFuncPTR    adc_convend_handler;
00093
00094 g_dmaCTRL g_dmaCTRLPKT;    /* dma control packet configuration stack */
00095
00096 spiDAT1_t spi2_conf;
00097
00098 #endif
00099
00100 /*
00101  * Platform independent global variables
00102  */
00103
00104 // create pointers to ADC buffers
00105 volatile int16_t* AD_res_1 = AD_dma_res_1;
00106 volatile int16_t* AD_res_2 = AD_dma_res_2;
00107 volatile int16_t* AD_res_3 = AD_dma_res_3;
00108
00109 volatile mlc_adc_result* AD_res_1_struct = (mlc_adc_result*) AD_dma_res_1;
00110 volatile mlc_adc_result* AD_res_2_struct = (mlc_adc_result*) AD_dma_res_2;
00111 volatile mlc_adc_result* AD_res_3_struct = (mlc_adc_result*) AD_dma_res_3;
```

```
00112
00113 // dac
00114 uint16_t dac_values[8];
00115 uint16_t dac_sel_ch[8];
00116
00117 // pwr outputs buffer
00118 volatile uint16_t pwr_out_state = 0;
00119 // debug LEDs state
00120 volatile uint16_t dbg_state = 0;
00121
00123 volatile int temp;
00124
00125 mlc_info_t MLC_info_struct;
00126 mlc_info_t* pMLC_info_struct = &MLC_info_struct;
00127 uint16_t ad_conf = 0;
00128 uint16_t ad_soc_conf = 0;
00129
00130 /*
00131  * Interrupt handlers
00132  */
00133
00134 //#if USE_AD_INTERRUPT == 1
00135 #if MCU_TYPE == MCU_TYPE_TMS320F2812 || MCU_TYPE == MCU_TYPE_TMS320F28335
00136 // TMS320Fxxxx needs this while TMS570 not
00137 interrupt
00138 #endif
00139 void isr_conv_end(void){
00140         // unimplemented ... consider implementing your own handler.
00141         //ESTOP0;
00142 #if MCU_TYPE == MCU_TYPE_TMS320F28335
00143         PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
00144 #endif
00145
00146 #if MCU_TYPE == MCU_TYPE_TMS570LS3137
00147         asm(" MOV R0, R0"); // NOP
00148 #endif
00149 }
00150
00151 #if MCU_TYPE == MCU_TYPE_TMS570LS3137 //TODO: Implement ISR of ADC conv end
00152
00162 void dmaConfigCtrlPacket(uint32 sadd,uint32 dadd,uint32 dsize)
00163 {
00164   g_dmaCTRLPKT.SADD      = sadd;                          /* source address             */
00165   g_dmaCTRLPKT.DADD      = dadd;                          /* destination  address       */
00166   g_dmaCTRLPKT.CHCTRL    = 0;                /* channel control            */
00167   g_dmaCTRLPKT.FRCNT     = 1;                /* frame count                */
00168   g_dmaCTRLPKT.ELCNT     = dsize;            /* element count              */
00169   g_dmaCTRLPKT.ELDOFFSET = 4;                /* element destination offset */
00170   g_dmaCTRLPKT.ELSOFFSET = 0;                    /* element destination offset */
00171   g_dmaCTRLPKT.FRDOFFSET = 0;                    /* frame destination offset   */
00172   g_dmaCTRLPKT.FRSOFFSET = 0;                /* frame destination offset   */
00173   g_dmaCTRLPKT.PORTASGN  = 4;                /* port b                     */
00174   g_dmaCTRLPKT.RDSIZE    = ACCESS_16_BIT;        /* read size                      */
00175   g_dmaCTRLPKT.WRSIZE    = ACCESS_16_BIT;        /* write size                     */
00176   g_dmaCTRLPKT.TTYPE     = FRAME_TRANSFER ;  /* transfer type              */
00177   g_dmaCTRLPKT.ADDMODERD = ADDR_OFFSET;      /* address mode read          */
00178   g_dmaCTRLPKT.ADDMODEWR = ADDR_INC1;        /* address mode write         */
00179   g_dmaCTRLPKT.AUTOINIT  = AUTOINIT_ON;      /* autoinit                   */
00180 }
00181
00182 #pragma CODE_STATE(gio_isr_handler, 32)
00183 #pragma INTERRUPT(gio_isr_handler, IRQ)
00184
00185 void gio_isr_handler(void){
00186         // check if it is GIOA5 and autoclear int flag
00187         int32_t offset = gioREG->OFF1 - 1U;
00188     // adc conversion end isr
00189         if (offset == 5){
00190         // some code
00191                 adc_convend_handler();
00192     }
00193 }
00194 #endif
00195
00196 //#endif
00197 #if MCU_TYPE == MCU_TYPE_TMS320F2812 || MCU_TYPE == MCU_TYPE_TMS320F28335
00198 interrupt
00199 #endif
00200 void dma_transfer_end(void){
00201         // unimplemented ... consider implementing your own handler.
00202         //ESTOP0;
00203 #if MCU_TYPE == MCU_TYPE_TMS570LS3137 //TODO: Implement DMA transfer end
00204         asm(" MOV R0, R0"); // NOP
00205 #endif
00206
00207 #if MCU_TYPE == MCU_TYPE_TMS320F28335
00208         PieCtrlRegs.PIEACK.all = PIEACK_GROUP7;
```

```
00209 #endif
00210 }
00211
00212 /*
00213  * Support functions
00214  */
00215 #ifndef TMS570LS3137_CLK
00216         #define TMS570LS3137_CLK 180000000
00217 #endif
00218
00219 #define DELAY_1US               13
00220
00221 #if MCU_TYPE == MCU_TYPE_TMS570LS3137
00222         /*
00223          *  Simple delay function with 1us resolution, 330s is maximal delay
00224          */
00225         void DELAY_US(float us){
00226                 register uint32_t i;
00227                 for (i=0;i<(uint32_t)(us*DELAY_1US);i++){
00228                 }
00229         }
00230 #endif
00231
00232 /*
00233  * Init of GPIO, 28335 needs to switch pins to XINTF
00234 */
00235
00236 void MLC_init_F28335(void){
00237
00238 #if MCU_TYPE == MCU_TYPE_TMS320F28335
00239         EALLOW;
00240         // XZCS0
00241         GpioCtrlRegs.GPBMUX1.bit.GPIO36 = 0x3;
00242 #if USE_EXT_RAM == 1
00243         // XZCS7
00244         GpioCtrlRegs.GPBMUX1.bit.GPIO37 = 0x3;
00245 #endif
00246         // XWE0
00247         GpioCtrlRegs.GPBMUX1.bit.GPIO38 = 0x3;
00248         // XA0 - XA7
00249         GpioCtrlRegs.GPBMUX1.bit.GPIO40 = 0x3;
00250         GpioCtrlRegs.GPBMUX1.bit.GPIO41 = 0x3;
00251         GpioCtrlRegs.GPBMUX1.bit.GPIO42 = 0x3;
00252         GpioCtrlRegs.GPBMUX1.bit.GPIO43 = 0x3;
00253         GpioCtrlRegs.GPBMUX1.bit.GPIO44 = 0x3;
00254         GpioCtrlRegs.GPBMUX1.bit.GPIO45 = 0x3;
00255         GpioCtrlRegs.GPBMUX1.bit.GPIO46 = 0x3;
00256         GpioCtrlRegs.GPBMUX1.bit.GPIO47 = 0x3;
00257
00258         // XD15 - XD0
00259         GpioCtrlRegs.GPCMUX1.bit.GPIO64 = 0x3;
00260         GpioCtrlRegs.GPCMUX1.bit.GPIO65 = 0x3;
00261         GpioCtrlRegs.GPCMUX1.bit.GPIO66 = 0x3;
00262         GpioCtrlRegs.GPCMUX1.bit.GPIO67 = 0x3;
00263         GpioCtrlRegs.GPCMUX1.bit.GPIO68 = 0x3;
00264         GpioCtrlRegs.GPCMUX1.bit.GPIO69 = 0x3;
00265         GpioCtrlRegs.GPCMUX1.bit.GPIO70 = 0x3;
00266         GpioCtrlRegs.GPCMUX1.bit.GPIO71 = 0x3;
00267         GpioCtrlRegs.GPCMUX1.bit.GPIO72 = 0x3;
00268         GpioCtrlRegs.GPCMUX1.bit.GPIO73 = 0x3;
00269         GpioCtrlRegs.GPCMUX1.bit.GPIO74 = 0x3;
00270         GpioCtrlRegs.GPCMUX1.bit.GPIO75 = 0x3;
00271         GpioCtrlRegs.GPCMUX1.bit.GPIO76 = 0x3;
00272         GpioCtrlRegs.GPCMUX1.bit.GPIO77 = 0x3;
00273         GpioCtrlRegs.GPCMUX1.bit.GPIO78 = 0x3;
00274         GpioCtrlRegs.GPCMUX1.bit.GPIO79 = 0x3;
00275
00276         // TODO: reduce XA only to used ones
00277         // XA8 - XA15
00278         GpioCtrlRegs.GPCMUX2.bit.GPIO80 = 0x3;
00279         GpioCtrlRegs.GPCMUX2.bit.GPIO81 = 0x3;
00280         GpioCtrlRegs.GPCMUX2.bit.GPIO82 = 0x3;
00281         GpioCtrlRegs.GPCMUX2.bit.GPIO83 = 0x3;
00282         GpioCtrlRegs.GPCMUX2.bit.GPIO84 = 0x3;
00283         GpioCtrlRegs.GPCMUX2.bit.GPIO85 = 0x3;
00284         GpioCtrlRegs.GPCMUX2.bit.GPIO86 = 0x3;
00285         GpioCtrlRegs.GPCMUX2.bit.GPIO87 = 0x3;
00286         // XA - 16
00287         GpioCtrlRegs.GPBMUX1.bit.GPIO39 = 0x3;
00288
00289 #if USE_256kB_RAM == 1
00290         // enable XA17 as XINTF -> allow addresing of 256kB
00291         GpioCtrlRegs.GPAMUX2.bit.GPIO31 = 0x3;
00292 #endif
00293
00294 /*
00295  *      Initialize GPIO according to MLC interface hardware
```

```
00296  */
00297         // configure pins for AD converter - outputs
00298         // CONVSTA
00299         GpioCtrlRegs.GPBMUX1.bit.GPIO32 = 0x0;
00300         GpioCtrlRegs.GPBDIR.bit.GPIO32 = 0x1;
00301         // CONVSTB
00302         GpioCtrlRegs.GPBMUX1.bit.GPIO33 = 0x0;
00303         GpioCtrlRegs.GPBDIR.bit.GPIO33 = 0x1;
00304
00305         // prepare first conversion - goto idle state
00306         GpioDataRegs.GPBSET.bit.GPIO32 = 0x1;
00307         GpioDataRegs.GPBSET.bit.GPIO33 = 0x1;
00308
00309         // AD busy - input
00310         GpioCtrlRegs.GPAMUX2.bit.GPIO25 = 0x0;
00311         GpioCtrlRegs.GPADIR.bit.GPIO25 = 0x0;
00312
00313         // FRST data - input
00314         GpioCtrlRegs.GPBMUX1.bit.GPIO35 = 0x0;
00315         GpioCtrlRegs.GPBDIR.bit.GPIO35  = 0x0;
00316
00317         // Buttons
00318         GpioCtrlRegs.GPAMUX2.bit.GPIO29 = 0x0;
00319         GpioCtrlRegs.GPADIR.bit.GPIO29  = 0x0;
00320         GpioCtrlRegs.GPAMUX2.bit.GPIO30 = 0x0;
00321         GpioCtrlRegs.GPADIR.bit.GPIO30  = 0x0;
00322 #if USE_256kB_RAM != 1
00323         // use XA17 as button input
00324         GpioCtrlRegs.GPAMUX2.bit.GPIO31 = 0x0;
00325         GpioCtrlRegs.GPADIR.bit.GPIO31  = 0x0;
00326 #endif
00327         GpioCtrlRegs.GPAMUX2.bit.GPIO28 = 0x0;
00328         GpioCtrlRegs.GPADIR.bit.GPIO28  = 0x0;
00329
00330
00331         // IRC input 50,51,53 EQEP1
00332         GpioCtrlRegs.GPBMUX2.bit.GPIO50 = 0x1;
00333         GpioCtrlRegs.GPBMUX2.bit.GPIO51 = 0x1;
00334         GpioCtrlRegs.GPBMUX2.bit.GPIO53 = 0x1;
00335
00336         GpioCtrlRegs.GPBQSEL2.bit.GPIO50 = 0x1;
00337         GpioCtrlRegs.GPBQSEL2.bit.GPIO51 = 0x1;
00338         GpioCtrlRegs.GPBQSEL2.bit.GPIO53 = 0x1;
00339         GpioCtrlRegs.GPBCTRL.bit.QUALPRD2 = 0x0F;
00340
00341
00342         GpioCtrlRegs.GPBQSEL2.bit.GPIO50 = 0; // Synch to SYSCLKOUT
00343         GpioCtrlRegs.GPBQSEL2.bit.GPIO51 = 0; // Synch to SYSCLKOUT
00344         GpioCtrlRegs.GPBQSEL2.bit.GPIO53 = 0; // Synch to SYSCLKOUT
00345
00346         EDIS;
00347
00348 /*
00349  * Configure XINTF timings
00350  */
00351
00352 /*
00353         ---------|___________|----------
00354
00355         <XRDLEAD> <XRDACTIVE> <XRDTRAIL>
00356
00357         <           adresa             >
00358         ------------------<data>--------
00359 */
00360         // 26 ns cycle granularity
00361         EALLOW;
00362         // 16-bit data bus wide interface
00363         XintfRegs.XTIMING0.bit.XSIZE = 0x3;
00364
00365         // do not use READY
00366         XintfRegs.XTIMING0.bit.USEREADY = 0x0;
00367         XintfRegs.XTIMING0.bit.READYMODE = 0x0;
00368
00369         // XINTF read timing
00370         XintfRegs.XTIMING0.bit.XRDLEAD   = 0x1;//1 //26ns
00371         XintfRegs.XTIMING0.bit.XRDACTIVE = 0x3;//3 //78ns
00372         XintfRegs.XTIMING0.bit.XRDTRAIL  = 0x0;//0 //0ns -- not valid ?
00373
00374         // XINTF write timing
00375         XintfRegs.XTIMING0.bit.XWRLEAD   = 0x1;//1 //26ns
00376         XintfRegs.XTIMING0.bit.XWRACTIVE = 0x1;//1 //26ns
00377         XintfRegs.XTIMING0.bit.XWRTRAIL  = 0x2;//2 //52ns
00378
00379         // scale 1:2
00380         XintfRegs.XTIMING0.bit.X2TIMING = 0x1;
00381         EDIS;
00382
```

```
00383
00384 #if USE_EXT_RAM == 1
00385         EALLOW;
00386         // setup ZONE7
00387         // 16-bit wide interface
00388         XintfRegs.XTIMING7.bit.XSIZE = 0x3;
00389
00390         // do not use READY
00391         XintfRegs.XTIMING7.bit.USEREADY = 0x0;
00392         XintfRegs.XTIMING7.bit.READYMODE = 0x0;
00393
00394         // XINTF read timing
00395         XintfRegs.XTIMING7.bit.XRDLEAD   = 0x1;
00396         XintfRegs.XTIMING7.bit.XRDACTIVE = 0x1;
00397         XintfRegs.XTIMING7.bit.XRDTRAIL  = 0x1;
00398
00399         // XINTF write timing
00400         XintfRegs.XTIMING7.bit.XWRLEAD   = 0x1;
00401         XintfRegs.XTIMING7.bit.XWRACTIVE = 0x1;
00402         XintfRegs.XTIMING7.bit.XWRTRAIL  = 0x1;
00403
00404         // scale 1:1
00405         XintfRegs.XTIMING7.bit.X2TIMING = 0x0;
00406         EDIS;
00407 #endif
00408         EALLOW;
00409         // XTIMCLK=SYSCLKOUT =  when 0 or 75MHz when 1
00410         XintfRegs.XINTCNF2.bit.XTIMCLK = 0x1; // 75MHz
00411         // tune output clock to be:
00412         // 1/2 of XTIMCLK when 1 or 1/1 when 0
00413         // 26.6ns is one unit
00414         XintfRegs.XINTCNF2.bit.CLKMODE = 0x1; // 37.5MHz
00415
00416         EDIS;
00417         // wait until all configuration passes pipeline
00418         asm(" RPT #7 || NOP");
00419
00420 #if FORCE_PCB_REV != 0
00421         mlc_pcb_rev = FORCE_PCB_REV;
00422 #else
00423         // read PCB revision (stored in CPLD firmware)
00424         MLC_info_struct.pcb_hw_ver = MLC_READ(READ_PCB_REV);
00425 #endif
00426
00427         // read firmware revision of CPLD
00428         MLC_info_struct.cpld_fw_ver = MLC_READ(READ_VER);
00429         // and FPGA
00430         MLC_info_struct.fpga_fw_ver = MLC_READ(READ_FPGA_VER);
00431
00432 #if ALLOW_XINTF_SPEEDUP == 1
00433
00434         // check if we can enable speedup, depends on PCB revision and firmware
00435         if ((MLC_info_struct.pcb_hw_ver > PCB_REV_1) && (MLC_info_struct.
      cpld_fw_ver > 3)){
00436                 DBG_INFO_PUTS("XINTF is set to higher speed.");
00437                 // 13 ns cycle granularity
00438                 EALLOW;
00439                 // XINTF read timing
00440                 XintfRegs.XTIMING0.bit.XRDLEAD   = 0x1; //13ns
00441                 XintfRegs.XTIMING0.bit.XRDACTIVE = 0x5; //65ns
00442                 XintfRegs.XTIMING0.bit.XRDTRAIL  = 0x0; //0ns -- not valid ?
00443
00444                 // XINTF write timing
00445                 XintfRegs.XTIMING0.bit.XWRLEAD   = 0x1; //13ns
00446                 XintfRegs.XTIMING0.bit.XWRACTIVE = 0x2; //26ns
00447                 XintfRegs.XTIMING0.bit.XWRTRAIL  = 0x3; //39ns
00448
00449                 // scale 1:1
00450                 XintfRegs.XTIMING0.bit.X2TIMING = 0x0;
00451                 EDIS;
00452                 // wait untill all config pass pipeline
00453                 asm(" RPT #7 || NOP");
00454         } else {
00455                 //DBG_INFO_PUTS("Can not setup XINTF to higher speed, FW of CPLD does not allow this.
       Upgrade to version 4 or higher.");
00456         }
00457 #endif
00458
00459
00460         // CONVSTC
00461         // check PCB revison
00462         if (MLC_info_struct.pcb_hw_ver > PCB_REV_1){
00463                 EALLOW;
00464                 GpioCtrlRegs.GPBMUX2.bit.GPIO61= 0x0;
00465                 GpioCtrlRegs.GPBDIR.bit.GPIO61 = 0x1;
00466                 GpioDataRegs.GPBSET.bit.GPIO61 = 0x1;
00467                 EDIS;
```

```
00468         }
00469
00470         // revision 3 of FW has global enable
00471         if (MLC_info_struct.cpld_fw_ver > 2){
00472                 EALLOW;
00473                 GpioCtrlRegs.GPBMUX2.bit.GPIO60 = 0x0;
00474                 GpioCtrlRegs.GPBDIR.bit.GPIO60 = 0x1;
00475                 GpioDataRegs.GPBSET.bit.GPIO60 = 0x1;
00476                 EDIS;
00477         }
00478
00479 //#if USE_DMA == 1
00480
00481         // run even if debug request halt
00482         //DmaRegs.DEBUGCTRL.bit.FREE = 1;
00483
00484
00485         EALLOW;
00486         // disable DMA for CH1
00487         DmaRegs.CH1.CONTROL.bit.RUN = 0;
00488         // disable DMA for CH2
00489         DmaRegs.CH2.CONTROL.bit.RUN = 0;
00490         // disable DMA for CH3
00491         DmaRegs.CH3.CONTROL.bit.RUN = 0;
00492         EDIS;
00493
00494 #if USE_ADC_DMA == 1
00495         EALLOW;
00496         // allow DMA
00497         DmaRegs.CH1.CONTROL.bit.RUN = 1;
00498         // allow interrupt
00499         DmaRegs.CH1.MODE.bit.CHINTE = 1;
00500         // data size 16b
00501         DmaRegs.CH1.MODE.bit.DATASIZE = 0;
00502         // set to continuous mode - after transfer DMA is reset and wait
00503         // for another interrupt event
00504         DmaRegs.CH1.MODE.bit.CONTINUOUS = 1;
00505         // generate interrupt when transfer done
00506         DmaRegs.CH1.MODE.bit.CHINTMODE = 1;
00507         // enable start of DMA from interrupt
00508         //DmaRegs.CH1.MODE.bit.PERINTE = 1;
00509         // select proper interrupt source for start of transfer
00510         // in our case it is XINT1 which is connected to BUSY of ADC
00511         DmaRegs.CH1.MODE.bit.PERINTSEL = 3;
00512         // burst setup - 16 words
00513         DmaRegs.CH1.BURST_SIZE.bit.BURSTSIZE = 15;
00514         // burst step source - no address change
00515         DmaRegs.CH1.SRC_BURST_STEP = 0;
00516         // burst step destination - add 1 to address
00517         DmaRegs.CH1.DST_BURST_STEP = 1;
00518         // transfer size - one burst
00519         DmaRegs.CH1.TRANSFER_SIZE = 0;
00520         // begin address
00521         DmaRegs.CH1.SRC_ADDR_SHADOW = BASE_ADDRESS + READ_AD1;
00522         // dest address
00523         DmaRegs.CH1.DST_ADDR_SHADOW = (Uint32)AD_dma_res_1;
00524         // setup isr handler
00525         //MLC_enable_DMA_isr(AD_CH1, dma_transfer_end);
00526         // clear all flags
00527         DmaRegs.CH1.CONTROL.bit.ERRCLR = 1;
00528         DmaRegs.CH1.CONTROL.bit.SYNCCLR = 1;
00529         DmaRegs.CH1.CONTROL.bit.PERINTCLR = 1;
00530
00531         // allow DMA
00532         DmaRegs.CH2.CONTROL.bit.RUN = 1;
00533         // allow interrupt
00534         DmaRegs.CH2.MODE.bit.CHINTE = 1;
00535         // data size 16b
00536         DmaRegs.CH2.MODE.bit.DATASIZE = 0;
00537         // set to continuous mode - after transfer DMA is reset and wait
00538         // for another interrupt event
00539         DmaRegs.CH2.MODE.bit.CONTINUOUS = 1;
00540     // allow interrupt when done only when other channels are disabled
00541         // generate interrupt when transfer done
00542         DmaRegs.CH2.MODE.bit.CHINTMODE = 1;
00543         // enable start of DMA from interrupt
00544         //DmaRegs.CH2.MODE.bit.PERINTE = 1;
00545         // select proper interrupt source for start of transfer
00546         // in our case it is XINT1
00547         DmaRegs.CH2.MODE.bit.PERINTSEL = 3;
00548         // burst setup - 16 words
00549         DmaRegs.CH2.BURST_SIZE.bit.BURSTSIZE = 15;
00550         // burst step source - no address change
00551         DmaRegs.CH2.SRC_BURST_STEP = 0;
00552         // burst step destination - add 1 to address
00553         DmaRegs.CH2.DST_BURST_STEP = 1;
00554         // transfer size - one burst
```

```
00555         DmaRegs.CH2.TRANSFER_SIZE = 0;
00556         // begin address
00557         DmaRegs.CH2.SRC_ADDR_SHADOW = BASE_ADDRESS + READ_AD2;
00558         // dest address
00559         DmaRegs.CH2.DST_ADDR_SHADOW = (Uint32)AD_dma_res_2;
00560         // setup isr handler
00561         //MLC_enable_DMA_isr(AD_CH2, dma_transfer_end);
00562 
00563         // clear all flags
00564         DmaRegs.CH2.CONTROL.bit.ERRCLR = 1;
00565         DmaRegs.CH2.CONTROL.bit.SYNCCLR = 1;
00566         DmaRegs.CH2.CONTROL.bit.PERINTCLR = 1;
00567 
00568         // allow DMA
00569         DmaRegs.CH3.CONTROL.bit.RUN = 1;
00570         // allow interrupt
00571         DmaRegs.CH3.MODE.bit.CHINTE = 1;
00572         // data size 16b
00573         DmaRegs.CH3.MODE.bit.DATASIZE = 0;
00574         // set to continuous mode - after transfer DMA is reset and wait
00575         // for another interrupt event
00576         DmaRegs.CH3.MODE.bit.CONTINUOUS = 1;
00577         // generate interrupt when transfer done
00578         DmaRegs.CH3.MODE.bit.CHINTMODE = 1;
00579         // enable start of DMA from interrupt
00580         //DmaRegs.CH3.MODE.bit.PERINTE = 1;
00581         // select proper interrupt source for start of transfer
00582         // in our case it is XINT1
00583         DmaRegs.CH3.MODE.bit.PERINTSEL = 3;
00584         // burst setup - 16 words
00585         DmaRegs.CH3.BURST_SIZE.bit.BURSTSIZE = 15;
00586         // burst step source - no address change
00587         DmaRegs.CH3.SRC_BURST_STEP = 0;
00588         // burst step destination - add 1 to address
00589         DmaRegs.CH3.DST_BURST_STEP = 1;
00590         // transfer size - one burst
00591         DmaRegs.CH3.TRANSFER_SIZE = 0;
00592         // begin address
00593         DmaRegs.CH3.SRC_ADDR_SHADOW = BASE_ADDRESS + READ_AD3;
00594         // dest address
00595         DmaRegs.CH3.DST_ADDR_SHADOW = (Uint32)AD_dma_res_3;
00596         // setup default isr handler
00597         //MLC_enable_DMA_isr(AD_CH3, dma_transfer_end);
00598         // clear all flags
00599         DmaRegs.CH3.CONTROL.bit.ERRCLR = 1;
00600         DmaRegs.CH3.CONTROL.bit.SYNCCLR = 1;
00601         DmaRegs.CH3.CONTROL.bit.PERINTCLR = 1;
00602         EDIS;
00603 #endif
00604 #endif
00605 }
00606 
00607 /*
00608  * F2812 init routine
00609  */
00610 
00611 void MLC_init_F2812(void){
00612 #if MCU_TYPE == MCU_TYPE_TMS320F2812
00613         EALLOW;
00614         // configure pins for AD converter - outputs
00615         // CONVSTA
00616         GpioMuxRegs.GPDMUX.bit.T2CTRIP_SOCA_GPIOD1 = 0x0;
00617         GpioMuxRegs.GPDDIR.bit.GPIOD1 = 0x1;
00618         // CONVSTB
00619         GpioMuxRegs.GPDMUX.bit.T4CTRIP_SOCB_GPIOD6 = 0x0;
00620         GpioMuxRegs.GPDDIR.bit.GPIOD6 = 0x1;
00621 
00622         // prepare first conversion - goto idle state
00623         GpioDataRegs.GPDSET.bit.GPIOD1 = 0x1;
00624         GpioDataRegs.GPDSET.bit.GPIOD6 = 0x1;
00625 
00626         // AD busy - input
00627         GpioMuxRegs.GPEMUX.bit.XNMI_XINT13_GPIOE2 = 0x0;
00628         GpioMuxRegs.GPEDIR.bit.GPIOE2 = 0x0;
00629 
00630         // FRST data - input --UNSUPORTED ! in rev. 1
00631         GpioMuxRegs.GPFMUX.bit.MDXA_GPIOF12 = 0x0;
00632         GpioMuxRegs.GPFDIR.bit.GPIOF12  = 0x0;
00633 
00634         // Buttons
00635         GpioMuxRegs.GPFMUX.bit.MCLKXA_GPIOF8 = 0x0;
00636         GpioMuxRegs.GPFDIR.bit.GPIOF8  = 0x0;
00637         GpioMuxRegs.GPFMUX.bit.MCLKRA_GPIOF9 = 0x0;
00638         GpioMuxRegs.GPFDIR.bit.GPIOF9  = 0x0;
00639         GpioMuxRegs.GPFMUX.bit.MFSRA_GPIOF11 = 0x0;
00640         GpioMuxRegs.GPFDIR.bit.GPIOF11  = 0x0;
00641         GpioMuxRegs.GPFMUX.bit.MFSXA_GPIOF10 = 0x0;
```

```
00642         GpioMuxRegs.GPFDIR.bit.GPIOF10  = 0x0;
00643
00644
00645
00646         // IRC input 50,51,53 EQEP1
00647         GpioMuxRegs.GPAMUX.bit.CAP1Q1_GPIOA8 = 0x1;
00648         GpioMuxRegs.GPAMUX.bit.CAP2Q2_GPIOA9 = 0x1;
00649         GpioMuxRegs.GPAMUX.bit.CAP3QI1_GPIOA10 = 0x1;
00650
00651         EDIS;
00652         /*
00653                 ---------|___________|----------
00654
00655                 <XRDLEAD> <XRDACTIVE> <XRDTRAIL>
00656
00657                 <           adresa             >
00658                 ------------------<data>--------
00659         */
00660                 EALLOW;
00661                 // 16-bit data bus wide interface
00662                 XintfRegs.XTIMING1.bit.XSIZE = 0x3;
00663
00664                 // do not use READY
00665                 XintfRegs.XTIMING1.bit.USEREADY = 0x0;
00666                 XintfRegs.XTIMING0.bit.READYMODE = 0x0;
00667
00668                 // XINTF read timing
00669                 XintfRegs.XTIMING1.bit.XRDLEAD   = 0x1;//1
00670                 XintfRegs.XTIMING1.bit.XRDACTIVE = 0x3;//3
00671                 XintfRegs.XTIMING1.bit.XRDTRAIL  = 0x0;//0
00672
00673                 // XINTF write timing
00674                 XintfRegs.XTIMING1.bit.XWRLEAD   = 0x1;//1
00675                 XintfRegs.XTIMING1.bit.XWRACTIVE = 0x2;//1
00676                 XintfRegs.XTIMING1.bit.XWRTRAIL  = 0x1;//2
00677
00678                 // scale 1:2
00679                 XintfRegs.XTIMING1.bit.X2TIMING = 0x1;
00680                 EDIS;
00681
00682
00683         #if USE_EXT_RAM == 1
00684                 EALLOW;
00685                 // setup ZONE7
00686                 // 16-bit wide interface
00687                 XintfRegs.XTIMING7.bit.XSIZE = 0x3;
00688
00689                 // do not use READY
00690                 XintfRegs.XTIMING7.bit.USEREADY = 0x0;
00691                 XintfRegs.XTIMING7.bit.READYMODE = 0x0;
00692
00693                 // XINTF read timing
00694                 XintfRegs.XTIMING7.bit.XRDLEAD   = 0x1;
00695                 XintfRegs.XTIMING7.bit.XRDACTIVE = 0x1;
00696                 XintfRegs.XTIMING7.bit.XRDTRAIL  = 0x1;
00697
00698                 // XINTF write timing
00699                 XintfRegs.XTIMING7.bit.XWRLEAD   = 0x1;
00700                 XintfRegs.XTIMING7.bit.XWRACTIVE = 0x1;
00701                 XintfRegs.XTIMING7.bit.XWRTRAIL  = 0x1;
00702
00703                 // scale 1:1
00704                 XintfRegs.XTIMING7.bit.X2TIMING = 0x0;
00705                 EDIS;
00706         #endif
00707                 EALLOW;
00708                 // XTIMCLK=SYSCLKOUT =  when 0 or 75MHz when 1
00709                 XintfRegs.XINTCNF2.bit.XTIMCLK = 0x1; // 75MHz
00710                 // tune output clock to be:
00711                 // 1/2 of XTIMCLK when 1 or 1/1 when 0
00712                 XintfRegs.XINTCNF2.bit.CLKMODE = 0x1; // 37.5MHz
00713
00714                 EDIS;
00715                 // wait until all configuration passes pipeline
00716                 asm(" RPT #7 || NOP");
00717
00718         #if FORCE_PCB_REV != 0
00719                 mlc_pcb_rev = FORCE_PCB_REV;
00720         #else
00721                 // read PCB revision (stored in CPLD firmware)
00722                 MLC_info_struct.pcb_hw_ver = MLC_READ(
    READ_PCB_REV);
00723         #endif
00724
00725                 // read firmware revision of CPLD
00726                 MLC_info_struct.cpld_fw_ver = MLC_READ(
    READ_VER);
```

```
00727                 // and FPGA
00728                 MLC_info_struct.fpga_fw_ver = MLC_READ(
    READ_FPGA_VER);
00729
00730         #if ALLOW_XINTF_SPEEDUP == 1
00731
00732                 // check if we can enable speedup, depends on PCB revision and firmware
00733                 if ((mlc_pcb_rev > PCB_REV_1) && (mlc_fw_rev > 3)){
00734                         DBG_INFO_PUTS("XINTF is set to higher speed.");
00735                         EALLOW;
00736                         // XINTF read timing
00737                         XintfRegs.XTIMING1.bit.XRDLEAD   = 0x1;
00738                         XintfRegs.XTIMING1.bit.XRDACTIVE = 0x5;
00739                         XintfRegs.XTIMING1.bit.XRDTRAIL  = 0x0;
00740
00741                         // XINTF write timing
00742                         XintfRegs.XTIMING1.bit.XWRLEAD   = 0x1;
00743                         XintfRegs.XTIMING1.bit.XWRACTIVE = 0x2;
00744                         XintfRegs.XTIMING1.bit.XWRTRAIL  = 0x3;
00745
00746                         // scale 1:1
00747                         XintfRegs.XTIMING1.bit.X2TIMING = 0x0;
00748                         EDIS;
00749                         // wait untill all config pass pipeline
00750                         asm(" RPT #7 || NOP");
00751                 } else {
00752                         DBG_INFO_PUTS("Can not setup XINTF to higher speed, FW of CPLD does
     not allow this. Upgrade to version 4 or higher.");
00753                 }
00754         #endif
00755
00756         // CONVSTC
00757         // check PCB revison
00758         if (MLC_info_struct.pcb_hw_ver > PCB_REV_1){
00759                 EALLOW;
00760                 GpioMuxRegs.GPAMUX.bit.TCLKINA_GPIOA12 = 0x0;
00761                 GpioMuxRegs.GPADIR.bit.GPIOA12 = 0x1;
00762                 GpioDataRegs.GPASET.bit.GPIOA12 = 0x1;
00763                 EDIS;
00764         }
00765
00766         // revision 3 of FW has global enable
00767         if (MLC_info_struct.cpld_fw_ver > 2){
00768                 EALLOW;
00769                 GpioMuxRegs.GPAMUX.bit.TDIRA_GPIOA11 = 0x0;
00770                 GpioMuxRegs.GPADIR.bit.GPIOA11 = 0x1;
00771                 GpioDataRegs.GPASET.bit.GPIOA11 = 0x1;
00772                 EDIS;
00773         }
00774 #endif
00775 }
00776
00777 /*
00778  * LS3137 init routine
00779  */
00780
00781 void MLC_init_LS3137(void){
00782 #if MCU_TYPE == MCU_TYPE_TMS570LS3137
00783         systemInit(); // init system, including pinmux, PLL etc.
00784         systemREG1->GPREG1 |= 0x80000000; // this must be there for proper function of EMIF, see SPNU499B
     p. 158
00785         emif_ASYNC1Init(); // setup peripherals EMIF
00786         emif_ASYNC2Init(); // setup ext. RAM EMIF
00787
00788         // read firmware revision of CPLD
00789         MLC_info_struct.cpld_fw_ver = MLC_READ(READ_VER);
00790         // and FPGA
00791         MLC_info_struct.fpga_fw_ver = MLC_READ(READ_FPGA_VER);
00792
00793
00794         gioInit(); //setup GIOA/GIOB
00795
00796         // init default GPIOA5 handler - conv end
00797         adc_convend_handler = isr_conv_end;
00798         // setup VIM
00799         vimInit();
00800         // GIO will trigger gio_isr_handler
00801         vimChannelMap(9,9, &gio_isr_handler);
00802
00803         // DMA setup
00804         dmaEnable();            // enable DMA controller
00805         dmaReqAssign(0,1); //channel 0
00806         dmaReqAssign(1,2); //channel 1
00807         dmaReqAssign(2,3); //channel 2
00808
00809         // setup CH0 - ADC1
00810         dmaConfigCtrlPacket((uint32_t)((uint16_t*)
```

```
      BASE_ADDRESS+READ_AD1), (uint32_t)AD_res_1, 16);
00811         dmaSetCtrlPacket(DMA_CH0, g_dmaCTRLPKT);
00812
00813         // setup CH1 - ADC2
00814         dmaConfigCtrlPacket((uint32_t)((uint16_t*)
      BASE_ADDRESS+READ_AD2), (uint32_t)AD_res_2, 16);
00815         dmaSetCtrlPacket(DMA_CH1, g_dmaCTRLPKT);
00816
00817         // setup CH2 - ADC3
00818         dmaConfigCtrlPacket((uint32_t)((uint16_t*)
      BASE_ADDRESS+READ_AD3), (uint32_t)AD_res_3, 16);
00819         dmaSetCtrlPacket(DMA_CH2, g_dmaCTRLPKT);
00820 #endif
00821 }
00822
00823 // basic multiplatform init
00824 void MLC_init(void)
00825 {
00826 #if MCU_TYPE == MCU_TYPE_TMS320F28335
00827         MLC_init_F28335();
00828 #endif
00829
00830 #if MCU_TYPE == MCU_TYPE_TMS320F2812
00831         MLC_init_F2812();
00832 #endif
00833
00834 #if MCU_TYPE == MCU_TYPE_TMS570LS3137
00835         MLC_init_LS3137();
00836 #endif
00837
00838         // setup ADC
00839         MLC_ADC_reset();
00840         MLC_ADC_setup(AD_OS0, AD_OS0, AD_OS0,
      AD_RNG_5V, AD_RNG_5V, AD_RNG_5V);
00841         MLC_ADC_setup_SOC(AD_SOC_ALL_DEFAULT);
00842
00843         // test for ext module
00844         MLC_WRITE(RW_EXT_LED_REG, 0xAAAA);
00845         if (MLC_READ(RW_EXT_LED_REG) == 0xAAAA){
00846           MLC_info_struct.mod_fw_ver = MLC_READ(READ_EXT_VER);
00847         } else {
00848           MLC_info_struct.mod_fw_ver = 0x00;
00849           DBG_INFO_PUTS("External module is not present.");
00850         }
00851
00852 }
00853
00854
00855 void MLC_global_enable(){
00856         if (MLC_info_struct.cpld_fw_ver > 2){
00857 #if MCU_TYPE == MCU_TYPE_TMS320F28335
00858                 GpioDataRegs.GPBCLEAR.bit.GPIO60 = 0x1;
00859 #endif
00860
00861 #if MCU_TYPE == MCU_TYPE_TMS320F2812
00862                 GpioDataRegs.GPACLEAR.bit.GPIOA11 = 0x1;
00863 #endif
00864
00865 #if MCU_TYPE == MCU_TYPE_TMS570LS3137
00866                 // clear global enable
00867                 gioSetBit(gioPORTA, 1, 0);
00868 #endif
00869         }
00870 }
00871
00872 void MLC_global_disable(){
00873         if (MLC_info_struct.cpld_fw_ver > 2){
00874 #if MCU_TYPE == MCU_TYPE_TMS320F28335
00875                 GpioDataRegs.GPBSET.bit.GPIO60 = 0x1;
00876 #endif
00877
00878 #if MCU_TYPE == MCU_TYPE_TMS320F2812
00879                 GpioDataRegs.GPASET.bit.GPIOA11 = 0x1;
00880 #endif
00881
00882 #if MCU_TYPE == MCU_TYPE_TMS570LS3137
00883                 // clear global enable
00884                 gioSetBit(gioPORTA, 1, 1);
00885 #endif
00886         }
00887 }
00888
00889 // TMS320F2812 does not support DMA transfers
00890 #if MCU_TYPE == MCU_TYPE_TMS320F28335 || MCU_TYPE == MCU_TYPE_TMS570LS3137
00891 uint16_t MLC_DMA_active(){
00892 #if MCU_TYPE == MCU_TYPE_TMS320F28335
00893         return DmaRegs.PRIORITYSTAT.bit.ACTIVESTS;
```

```
00894 #endif
00895
00896 #if MCU_TYPE == MCU_TYPE_TMS570LS3137 //TODO: Implement DMA transfer progress function
00897         return 0;
00898 #endif
00899 }
00900
00901 void MLC_DMA_activate(){
00902 #if MCU_TYPE == MCU_TYPE_TMS320F28335
00903
00904         EALLOW;
00905         // enable start of DMA from ADC EOC interrupt
00906         DmaRegs.CH1.MODE.bit.PERINTE = 1;
00907         DmaRegs.CH2.MODE.bit.PERINTE = 1;
00908         DmaRegs.CH3.MODE.bit.PERINTE = 1;
00909         EDIS;
00910 #endif
00911
00912 #if MCU_TYPE == MCU_TYPE_TMS570LS3137 //TODO: Implement activation of ADC DMA
00913 #endif
00914 }
00915
00916 void MLC_DMA_deactivate(){
00917 #if MCU_TYPE == MCU_TYPE_TMS320F28335
00918         EALLOW;
00919         // enable start of DMA from ADC EOC interrupt
00920         DmaRegs.CH1.MODE.bit.PERINTE = 0;
00921         DmaRegs.CH2.MODE.bit.PERINTE = 0;
00922         DmaRegs.CH3.MODE.bit.PERINTE = 0;
00923         EDIS;
00924 #endif
00925
00926 #if MCU_TYPE == MCU_TYPE_TMS570LS3137 //TODO: Implement deactivation of ADC DMA
00927 #endif
00928 }
00929
00930 #endif
00931
00932 void MLC_DMA_enable_isr(uint16_t channel, interrupt void* handler){
00933 #if MCU_TYPE == MCU_TYPE_TMS320F28335
00934         DINT;
00935         EALLOW;
00936         //make sure that PIE has power
00937         PieCtrlRegs.PIECTRL.bit.ENPIE = 1;
00938         // enable group 7 - DMA interrupt
00939         IER |= M_INT7;
00940         switch (channel) {
00941                 case AD_CH1:
00942                         // CH1 uses .1
00943                         PieCtrlRegs.PIEIER7.bit.INTx1 = 0x1;
00944                         PieVectTable.DINTCH1 = handler;
00945                         break;
00946                 case AD_CH2:
00947                         // CH2 uses .2
00948                         PieCtrlRegs.PIEIER7.bit.INTx2 = 0x1;
00949                         PieVectTable.DINTCH2 = handler;
00950                         break;
00951                 case AD_CH3:
00952                         // CH3 uses .3
00953                         PieCtrlRegs.PIEIER7.bit.INTx3 = 0x1;
00954                         PieVectTable.DINTCH3 = handler;
00955                         break;
00956                 default:
00957                         break;
00958         }
00959         EDIS;
00960         EINT;
00961 #endif
00962
00963 // TODO: implement MLC_DMA_enable_isr
00964 #if MCU_TYPE == MCU_TYPE_TMS570LS3137
00965         dmaEnable();            // enable DMA controller
00966         dmaReqAssign(0,1); //channel 0
00967         dmaConfigCtrlPacket((uint32_t)(BASE_ADDRESS +
      READ_AD1), (uint32_t)&AD_dma_res_1, 16);
00968         dmaSetChEnable(DMA_CH0, DMA_SW);
00969 #endif
00970 }
00971
00972 void MLC_DMA_disable_isr(uint16_t channel){
00973
00974 // TODO: implement MLC_DMA_disable_isr
00975 #if MCU_TYPE == MCU_TYPE_TMS570LS3137
00976 #endif
00977
00978 #if MCU_TYPE == MCU_TYPE_TMS320F28335
00979         DINT;
```

```
00980         EALLOW;
00981         //make sure that PIE has power
00982         PieCtrlRegs.PIECTRL.bit.ENPIE = 1;
00983         // enable group 7 - DMA interrupt
00984         //IER |= M_INT7;
00985         switch (channel) {
00986                 case DMA_ADCH1:
00987                         // CH1 uses .1
00988                         PieCtrlRegs.PIEIER7.bit.INTx1 = 0x0;
00989                         break;
00990                 case DMA_ADCH2:
00991                         // CH2 uses .2
00992                         PieCtrlRegs.PIEIER7.bit.INTx2 = 0x0;
00993                         break;
00994                 case DMA_ADCH3:
00995                         // CH3 uses .3
00996                         PieCtrlRegs.PIEIER7.bit.INTx3 = 0x0;
00997                         break;
00998                 default:
00999                         break;
01000         }
01001         EDIS;
01002         EINT;
01003 #endif
01004 }
01005 //#endif // DMA functions exclusion end
01006
01007 // setup non-default interrupt handler
01008 void MLC_ADC_enable_isr(interrupt void* handler){
01009
01010 #if MCU_TYPE == MCU_TYPE_TMS320F28335
01011         DINT;
01012         EALLOW;
01013         //make sure that PIE has power
01014         PieCtrlRegs.PIECTRL.bit.ENPIE = 1;
01015         // enable INT1.4 - XINT1
01016         IER |= M_INT1;
01017         PieCtrlRegs.PIEIER1.bit.INTx4 = 0x1;
01018         // enable pin 25
01019         GpioIntRegs.GPIOXINT1SEL.bit.GPIOSEL = 25;
01020         // use default handler if user passes null pointer
01021         if (handler == NULL){
01022                 PieVectTable.XINT1 = isr_conv_end;
01023         } else {
01024                 PieVectTable.XINT1 = handler;
01025         }
01026         // Configure XINT1
01027     XIntruptRegs.XINT1CR.bit.POLARITY = 0;      // Falling edge interrupt
01028     XIntruptRegs.XINT1CR.bit.ENABLE = 1;        // Enable Xint1
01029         EDIS;
01030         EINT;
01031 #endif
01032
01033 #if MCU_TYPE == MCU_TYPE_TMS320F2812
01034 #endif
01035
01036 #if MCU_TYPE == MCU_TYPE_TMS570LS3137 //TODO: Implement ADC convert ISR
01037         gioEnableNotification(gioPORTA, 5);
01038 #endif
01039
01040 }
01041
01042 // disable XINT1
01043 void MLC_ADC_disable_isr(void){
01044 #if MCU_TYPE ==  MCU_TYPE_TMS320F28335
01045         DINT;
01046         EALLOW;
01047         PieCtrlRegs.PIEIER1.bit.INTx4 = 0x0;
01048         EDIS;
01049         EINT;
01050 #endif
01051
01052 #if MCU_TYPE == MCU_TYPE_TMS320F2812
01053 #endif
01054
01055 //TODO: Implement disabling of ADC convend
01056 #if MCU_TYPE == MCU_TYPE_TMS570LS3137
01057 #endif
01058 }
01059
01060 // resets AD converters
01061 void MLC_ADC_reset(void){
01062         unsigned int i;
01063         // clear results
01064         for (i=0;i<AD_LEN;i++){
01065                 AD_res_1[i] = 0;
01066                 AD_res_2[i] = 0;
```

```
01067                   AD_res_3[i] = 0;
01068           }
01069           // clear also DMA buffers
01070           for (i=0;i<16;i++){
01071                   AD_dma_res_1[i] = 0;
01072                   AD_dma_res_2[i] = 0;
01073                   AD_dma_res_3[i] = 0;
01074           }
01075           // reset ADs
01076           MLC_WRITE(WRITE_SETRES_AD, 0x0000);
01077           DELAY_US(10);
01078           // release reset
01079           MLC_WRITE(WRITE_CLRRES_AD, 0x0000);
01080           DELAY_US(10);
01081 }
01082
01083 //
01084 void MLC_ADC_start_conv(void){
01085 #if MCU_TYPE == MCU_TYPE_TMS320F28335
01086           // start all converters
01087           GpioDataRegs.GPBCLEAR.bit.GPIO32 = 0x1;
01088           GpioDataRegs.GPBCLEAR.bit.GPIO33 = 0x1;
01089           if (MLC_info_struct.pcb_hw_ver != PCB_REV_1){
01090                   GpioDataRegs.GPBCLEAR.bit.GPIO61 = 0x1;
01091           }
01092           DELAY_US(0.1);
01093           GpioDataRegs.GPBSET.bit.GPIO32 = 0x1;
01094           GpioDataRegs.GPBSET.bit.GPIO33 = 0x1;
01095           if (MLC_info_struct.pcb_hw_ver != PCB_REV_1){
01096                   GpioDataRegs.GPBSET.bit.GPIO61 = 0x1;
01097           }
01098 #endif
01099
01100 #if MCU_TYPE == MCU_TYPE_TMS320F2812
01101           // start all converters
01102           GpioDataRegs.GPDCLEAR.bit.GPIOD1 = 0x1;
01103           GpioDataRegs.GPDCLEAR.bit.GPIOD6 = 0x1;
01104           if (MLC_info_struct.pcb_hw_ver != PCB_REV_1){
01105                   GpioDataRegs.GPACLEAR.bit.GPIOA12 = 0x1;
01106           }
01107           DELAY_US(0.1);
01108           GpioDataRegs.GPDSET.bit.GPIOD1 = 0x1;
01109           GpioDataRegs.GPDSET.bit.GPIOD6 = 0x1;
01110           if (MLC_info_struct.pcb_hw_ver != PCB_REV_1){
01111                   GpioDataRegs.GPASET.bit.GPIOA12 = 0x1;
01112           }
01113 #endif
01114
01115 #if MCU_TYPE == MCU_TYPE_TMS570LS3137
01116           hetREG1->DCLR = (PIN_STARTCONVA | PIN_STARTCONVB | PIN_STARTCONVC);
01117           DELAY_US(0.1);
01118           hetREG1->DSET = (PIN_STARTCONVA | PIN_STARTCONVB | PIN_STARTCONVC);
01119 #endif
01120 }
01121
01122 void MLC_ADC_start_one_conv(uint16_t channel){
01123 #if MCU_TYPE == MCU_TYPE_TMS320F28335
01124           switch(channel){
01125           case AD_CH1:
01126                   GpioDataRegs.GPBCLEAR.bit.GPIO32 = 0x1;
01127                   DELAY_US(0.1);
01128                   GpioDataRegs.GPBSET.bit.GPIO32 = 0x1;
01129                   break;
01130           case AD_CH2:
01131                   GpioDataRegs.GPBCLEAR.bit.GPIO33 = 0x1;
01132                   DELAY_US(0.1);
01133                   GpioDataRegs.GPBSET.bit.GPIO33 = 0x1;
01134                   break;
01135           case AD_CH3:
01136                   if (MLC_info_struct.pcb_hw_ver != PCB_REV_1){
01137                           GpioDataRegs.GPBCLEAR.bit.GPIO61 = 0x1;
01138                           DELAY_US(0.1);
01139                           GpioDataRegs.GPBSET.bit.GPIO61 = 0x1;
01140                   }
01141                   break;
01142           }
01143 #endif
01144
01145 #if MCU_TYPE == MCU_TYPE_TMS320F2812
01146           switch(channel){
01147           case AD_CH1:
01148                   GpioDataRegs.GPDCLEAR.bit.GPIOD1 = 0x1;
01149                   DELAY_US(0.1);
01150                   GpioDataRegs.GPDSET.bit.GPIOD1 = 0x1;
01151                   break;
01152           case AD_CH2:
01153                   GpioDataRegs.GPDCLEAR.bit.GPIOD6 = 0x1;
```

```
01154                 DELAY_US(0.1);
01155                 GpioDataRegs.GPDSET.bit.GPIOD6 = 0x1;
01156                 break;
01157         case AD_CH3:
01158                 if (MLC_info_struct.pcb_hw_ver != PCB_REV_1){
01159                         GpioDataRegs.GPACLEAR.bit.GPIOA12 = 0x1;
01160                         DELAY_US(0.1);
01161                         GpioDataRegs.GPASET.bit.GPIOA12 = 0x1;
01162                 }
01163                 break;
01164         }
01165 #endif
01166
01167 #if MCU_TYPE == MCU_TYPE_TMS570LS3137
01168         switch(channel){
01169         case AD_CH1:
01170                 hetREG1->DCLR = PIN_STARTCONVA;
01171                 DELAY_US(0.1);
01172                 hetREG1->DSET = PIN_STARTCONVA;
01173                 break;
01174         case AD_CH2:
01175                 hetREG1->DCLR = PIN_STARTCONVB;
01176                 DELAY_US(0.1);
01177                 hetREG1->DSET = PIN_STARTCONVB;
01178                 break;
01179         case AD_CH3:
01180                 if (MLC_info_struct.pcb_hw_ver != PCB_REV_1){
01181                         hetREG1->DCLR = PIN_STARTCONVC;
01182                         DELAY_US(0.1);
01183                         hetREG1->DSET = PIN_STARTCONVC;
01184                 }
01185                 break;
01186         }
01187 #endif
01188 }
01189
01190 //
01191 void MLC_ADC_setup(unsigned int os_1, unsigned int os_2, unsigned int os_3, unsigned int rng_1
      , unsigned int rng_2, unsigned int rng_3){
01192         unsigned int old_conf;
01193 #if MCU_TYPE == MCU_TYPE_TMS320F28335
01194         // disable interrupt
01195         old_conf = PieCtrlRegs.PIEIER1.bit.INTx4;
01196         PieCtrlRegs.PIEIER1.bit.INTx4 = 0x0;
01197
01198         // write configuration, i.e. sets pins through CPLD
01199         ad_conf = (os_1 | (os_2 << 3) | (os_3 << 6) | (rng_1 << 9) | (rng_2 << 10) | (rng_3 << 11)) |
      ad_soc_conf;
01200         MLC_WRITE(WRITE_AD_CONF, ad_conf);
01201         // we must wait for changes are accepted by A/D
01202         DELAY_US(200);
01203         // restore interrupt
01204         PieCtrlRegs.PIEIER1.bit.INTx4 = old_conf;
01205         // enable AD interrupt
01206         //MLC_ADC_enable_isr(NULL);
01207 #endif
01208
01209 #if MCU_TYPE == MCU_TYPE_TMS320F2812
01210         // disable interrupt
01211         //old_conf = PieCtrlRegs.PIEIER1.bit.INTx4;
01212         //PieCtrlRegs.PIEIER1.bit.INTx4 = 0x0;
01213
01214         // TODO: 2812: implement ISR support
01215
01216         // write configuration, i.e. sets pins through CPLD
01217         ad_conf = (os_1 | (os_2 << 3) | (os_3 << 6) | (rng_1 << 9) | (rng_2 << 10) | (rng_3 << 11)) |
      ad_soc_conf;
01218         MLC_WRITE(WRITE_AD_CONF, ad_conf);
01219         // we must wait for changes are accepted by A/D
01220         DELAY_US(200);
01221         // restore interrupt
01222         //PieCtrlRegs.PIEIER1.bit.INTx4 = old_conf;
01223         // enable AD interrupt
01224         MLC_ADC_enable_isr(NULL);
01225 #endif
01226
01227 #if MCU_TYPE == MCU_TYPE_TMS570LS3137
01228         // write configuration, i.e. sets pins through CPLD
01229         ad_conf = (os_1 | (os_2 << 3) | (os_3 << 6) | (rng_1 << 9) | (rng_2 << 10) | (rng_3 << 11)) |
      ad_soc_conf;
01230         MLC_WRITE(WRITE_AD_CONF, ad_conf);
01231         // we must wait for changes are accepted by A/D
01232         DELAY_US(200);
01233         // enable AD interrupt
01234         MLC_ADC_enable_isr(NULL);
01235 #endif
01236 }
```

```
01237
01238 // wait for busy
01239 unsigned int MLC_ADC_wait(void){
01240         unsigned int timeout = 0;
01241 #if MCU_TYPE == MCU_TYPE_TMS320F28335
01242         while (GpioDataRegs.GPADAT.bit.GPIO25 == 0x1) timeout++;
01243 #endif
01244
01245 #if MCU_TYPE == MCU_TYPE_TMS320F2812
01246         while (GpioDataRegs.GPEDAT.bit.GPIOE2 == 0x1) timeout++;
01247 #endif
01248
01249 #if MCU_TYPE == MCU_TYPE_TMS570LS3137
01250         while (gioGetBit(gioPORTA, 5) == 0x1) timeout++;
01251 #endif
01252         return timeout;
01253 }
01254
01255 /*
01256  * MLC_ADC_read has two different implementations, one for TMS570LS3137 which utilizes SW triggered DMA and
       TMS320Fxxxx version with direct readings
01257  */
01258
01259 volatile int16_t* MLC_ADC_read(volatile int16_t* ad_res, uint16_t ad_addr){
01260
01261 #if QUICK_READ == 0
01262         unsigned int i;
01263 #endif
01264
01265 #if USE_FRSTDATA == 1
01266         // wait for FRST data
01267         ad_res[0] = MLC_READ(ad_addr);
01268
01269 #if MCU_TYPE == MCU_TYPE_TMS320F28335
01270         while (GpioDataRegs.GPBDAT.bit.GPIO35 == 0x0){
01271 #endif
01272 #if MCU_TYPE == MCU_TYPE_TMS320F2812
01273         while (GpioDataRegs.GPFDAT.bit.GPIOF12 == 0x0){
01274 #endif
01275 #if MCU_TYPE == MCU_TYPE_TMS570LS3137
01276         while (gioGetBit(gioPORTB, 7) == 0x0)
01277                 {
01278 #endif
01279                 ad_res[0] = MLC_READ(ad_addr);
01280                 }
01281         ad_res[1] = MLC_READ(ad_addr);
01282 #elif MCU_TYPE != MCU_TYPE_TMS570LS3137
01283         ad_res[0] = MLC_READ(ad_addr);
01284         ad_res[1] = MLC_READ(ad_addr);
01285         //DELAY_US(1);
01286 #endif
01287
01288 /*
01289  * Do not correct sign of CH1 of ADCs when using PCBv1.
01290  * This behavior provides same results with DMA transfer and "hand" transfers
01291  * of ADC data.
01292  */
01293
01294 #if QUICK_READ == 0
01295         for (i=2;i<16;i++){
01296                 // correct bad sign in revision 0.1 of PCB
01297                 //if ((i==2) && (mlc_pcb_rev == PCB_REV_1))
01298                 //      (ad_res)[i] = -1*MLC_READ(ad_addr);
01299                 //else
01300                         (ad_res)[i] = MLC_READ(ad_addr);
01301                 //temp = MLC_READ(ad_addr);
01302         }
01303 #elif MCU_TYPE != MCU_TYPE_TMS570LS3137
01304 //      if (mlc_pcb_rev == PCB_REV_1){
01305 //              ad_res[2] = -MLC_READ(ad_addr);
01306 //              ad_res[3] = -MLC_READ(ad_addr);
01307 //      } else {
01308         ad_res[2] = MLC_READ(ad_addr);
01309         ad_res[3] = MLC_READ(ad_addr);
01310 //      }
01311         ad_res[4] = MLC_READ(ad_addr);
01312         ad_res[5] = MLC_READ(ad_addr);
01313         ad_res[6] = MLC_READ(ad_addr);
01314         ad_res[7] = MLC_READ(ad_addr);
01315         ad_res[8] = MLC_READ(ad_addr);
01316         ad_res[9] = MLC_READ(ad_addr);
01317         ad_res[10] = MLC_READ(ad_addr);
01318         ad_res[11] = MLC_READ(ad_addr);
01319         ad_res[12] = MLC_READ(ad_addr);
01320         ad_res[13] = MLC_READ(ad_addr);
01321         ad_res[14] = MLC_READ(ad_addr);
01322         ad_res[15] = MLC_READ(ad_addr);
```

```
01323 #endif
01324
01325 #if MCU_TYPE == MCU_TYPE_TMS570LS3137 && QUICK_READ == 1 && USE_FRSTDATA == 0
01326         // according to address software toggle DMA transfer
01327         switch (ad_addr) {
01328                 case READ_AD1:
01329                         dmaSetChEnable(DMA_CH0, DMA_SW);
01330                         break;
01331                 case READ_AD2:
01332                         dmaSetChEnable(DMA_CH1, DMA_SW);
01333                         break;
01334                 case READ_AD3:
01335                         dmaSetChEnable(DMA_CH2, DMA_SW);
01336                         break;
01337                 case READ_ADC_ALL:
01338                         dmaSetChEnable(DMA_CH0, DMA_SW);
01339                         dmaSetChEnable(DMA_CH1, DMA_SW);
01340                         dmaSetChEnable(DMA_CH2, DMA_SW);
01341                         break;
01342                 default:
01343                         break;
01344         }
01345         // wait for transfer finish
01346         while (dmaREG->SWCHENAS & (1U << DMA_CH0 | 1U << DMA_CH1 | 1U << DMA_CH2)){
01347                 asm("   nop ");
01348         }
01349 #endif
01350         return ad_res;
01351 }
01352
01353 // returns pointer to array where are measured values stored
01354 volatile int16_t* MLC_ADC1_get_res_ptr(void){
01355         return AD_res_1;
01356 }
01357
01358 volatile int16_t* MLC_ADC2_get_res_ptr(void){
01359         return AD_res_2;
01360 }
01361
01362 volatile int16_t* MLC_ADC3_get_res_ptr(void){
01363         return AD_res_3;
01364 }
01365
01366 volatile mlc_adc_result* MLC_ADC1_get_res_strptr(void){
01367   return AD_res_1_struct;
01368 }
01369
01370 volatile mlc_adc_result* MLC_ADC2_get_res_strptr(void){
01371   return AD_res_2_struct;
01372 }
01373
01374 volatile mlc_adc_result* MLC_ADC3_get_res_strptr(void){
01375   return AD_res_3_struct;
01376 }
01377
01378 //
01379 volatile int16_t* MLC_ADC1_read(void){
01380         return MLC_ADC_read(AD_res_1,READ_AD1);
01381 }
01382
01383 //
01384 volatile int16_t* MLC_ADC2_read(void){
01385         return MLC_ADC_read(AD_res_2,READ_AD2);
01386 }
01387
01388 //
01389 volatile int16_t* MLC_ADC3_read(void){
01390         return MLC_ADC_read(AD_res_3,READ_AD3);
01391 }
01392
01393 float MLC_ADC_calc_volt(int16_t value, uint16_t range){
01394         return (float)value*range/16384;
01395 }
01396
01397 void MLC_LCD_write_to(uint16_t pos, char chr)
01398 {
01399         MLC_WRITE(WRITE_LCD_DISP, (pos<<8) | chr);
01400 }
01401
01402
01403 // writes string to LCD disp
01404 void MLC_LCD_write_str(char* pStr, uint16_t pos){
01405         char* pTemp;
01406         unsigned char index = pos;
01407         pTemp = pStr;
01408         // parse string
01409         while (*pTemp){
```

```
01410                 MLC_WRITE(WRITE_LCD_DISP, index << 8 | *pTemp++);
01411                 index++;
01412         }
01413 }
01414
01415 void MLC_LCD_clrscr(void){
01416         char i;
01417         for (i=0;i<32;i++){
01418                 MLC_WRITE(WRITE_LCD_DISP, (i<<8) | ' ');
01419         }
01420 }
01421
01422 void MLC_ADC_setup_SOC(uint16_t adsoc){
01423         ad_soc_conf = adsoc;
01424         ad_conf = (ad_conf & 0x0FFF) | adsoc;
01425         MLC_WRITE(WRITE_AD_CONF, ad_conf);
01426 }
01427
01428 void MLC_DAC_init(void)
01429 {
01430 #if MCU_TYPE == MCU_TYPE_TMS320F28335
01431         EALLOW;
01432         // setup pins of SPI
01433         GpioCtrlRegs.GPBPUD.bit.GPIO54 = 0;
01434         GpioCtrlRegs.GPBPUD.bit.GPIO56 = 0;
01435         GpioCtrlRegs.GPBPUD.bit.GPIO57 = 0;
01436
01437         GpioCtrlRegs.GPBMUX2.bit.GPIO54 = 1;
01438         GpioCtrlRegs.GPBMUX2.bit.GPIO56 = 1;
01439         GpioCtrlRegs.GPBMUX2.bit.GPIO57 = 1;
01440
01441         GpioCtrlRegs.GPBQSEL2.bit.GPIO54 = 3;
01442         GpioCtrlRegs.GPBQSEL2.bit.GPIO56 = 3;
01443         GpioCtrlRegs.GPBQSEL2.bit.GPIO57 = 3;
01444
01445         EDIS;
01446 #endif
01447
01448 #if MCU_TYPE == MCU_TYPE_TMS320F2812
01449         EALLOW;
01450         GpioMuxRegs.GPFMUX.bit.SPICLKA_GPIOF2 = 1;
01451         GpioMuxRegs.GPFMUX.bit.SPISIMOA_GPIOF0 = 1;
01452         GpioMuxRegs.GPFMUX.bit.SPISTEA_GPIOF3 = 1;
01453         EDIS;
01454 #endif
01455
01456
01457 #if MCU_TYPE == MCU_TYPE_TMS320F2812 || MCU_TYPE == MCU_TYPE_TMS320F28335
01458         // Inicializace SPI FIFO (pouziva se pouze TxFIFO)
01459     SpiaRegs.SPIFFTX.all=0xE040;
01460     SpiaRegs.SPIFFRX.all=0x405f;        // RXFIFO vypnuto (drzeno v RESET stavu)
01461     SpiaRegs.SPIFFCT.all=1;                     // zpozdeni zapisu TXFIFO -> SPIDAT ... DAC vyzaduje
     minimalne 50ns
01462                                                                 // nutno vyzkouset ... zpozdeni 2
     odpovida cca 100ns
01463
01464         SpiaRegs.SPICCR.bit.SPISWRESET =0;              // zablokovani SPI
01465         SpiaRegs.SPICTL.all =0x0006;                    // Enable master mode, normal phase,
01466         SpiaRegs.SPIBRR =0x0;                                   // SPICLK max = LSPCLK/4
01467     SpiaRegs.SPIPRI.bit.FREE = 1;           // debug udalosti neovlivnuji cinnost SPI
01468     SpiaRegs.SPICCR.all =0x008F;                    // odblokovani SPI a nastaveni prenosu 16-bit slova
01469 #endif
01470
01471 #if MCU_TYPE == MCU_TYPE_TMS570LS3137
01472     // setup SPI parameters
01473     spi2_conf.CSNR = 0; // select CS0
01474     spi2_conf.CS_HOLD = 0; // no hold of CS
01475     spi2_conf.DFSEL = 0; // data fomat 0 select
01476     spi2_conf.WDEL = 0; // no delay
01477     // init SPI
01478     spiInit();
01479 #endif
01480     // reset and setup DACs
01481     MLC_DAC_reset();
01482
01483     // preset ADC channels
01484     dac_sel_ch[DAC_CH_A] = DAC_CH_A;
01485     dac_sel_ch[DAC_CH_B] = DAC_CH_B;
01486     dac_sel_ch[DAC_CH_C] = DAC_CH_C;
01487     dac_sel_ch[DAC_CH_D] = DAC_CH_D;
01488     dac_sel_ch[DAC_CH_E] = DAC_CH_E;
01489     dac_sel_ch[DAC_CH_F] = DAC_CH_F;
01490     dac_sel_ch[DAC_CH_G] = DAC_CH_G;
01491     dac_sel_ch[DAC_CH_H] = DAC_CH_H;
01492 }
01493
01494 void MLC_DAC_control(uint16_t cmd)
```

```
01495 {
01496 #if MCU_TYPE == MCU_TYPE_TMS320F2812 || MCU_TYPE == MCU_TYPE_TMS320F28335
01497   SpiaRegs.SPITXBUF=cmd;
01498 #endif
01499
01500 #if MCU_TYPE == MCU_TYPE_TMS570LS3137
01501   spiTransmitData(spiREG2, &spi2_conf, 1, &cmd);
01502 #endif
01503 }
01504
01505
01506 void MLC_DAC_reset(void)
01507 {
01508   uint16_t config = 0x8000;
01509
01510 #if MCU_TYPE == MCU_TYPE_TMS320F2812 || MCU_TYPE == MCU_TYPE_TMS320F28335
01511   // reset DAC
01512   SpiaRegs.SPITXBUF=0xF000;
01513   //SpiaRegs.SPITXBUF=0xE000;
01514
01515   // wait for reset done
01516   DELAY_US(1000);
01517
01518   // set LDAC to ignored
01519   SpiaRegs.SPITXBUF=0xA000;
01520 #endif
01521
01522 #if MCU_TYPE == MCU_TYPE_TMS570LS3137 //TODO: Implement it
01523   config = 0xF000;
01524   spiTransmitData(spiREG2, &spi2_conf, 1, &config);
01525   DELAY_US(1000);
01526   config = 0xA000;
01527   spiTransmitData(spiREG2, &spi2_conf, 1, &config);
01528   config = 0x8000;
01529 #endif
01530   // setup DAs
01531   // gain 2xVref, Ref buffered
01532 #if DAC_USE_GAINx2 == 1
01533   config |= 0x0030;
01534 #endif
01535 #if DAC_USE_REF_BUFFER == 1
01536   config |= 0x000C;
01537 #endif
01538 #if DAC_USE_VCC_AS_REF == 1
01539   config |= 0x0003;
01540 #endif
01541
01542 #if MCU_TYPE == MCU_TYPE_TMS320F2812 || MCU_TYPE == MCU_TYPE_TMS320F28335
01543   SpiaRegs.SPITXBUF=config;
01544 #endif
01545
01546 #if MCU_TYPE == MCU_TYPE_TMS570LS3137 //TODO: Implement it
01547   spiTransmitData(spiREG2, &spi2_conf, 1, &config);
01548 #endif
01549 }
01550
01551 void MLC_DAC_write(uint16_t count)
01552 {
01553   unsigned int i;
01554   uint16_t data[8];
01555
01556   // avoid over indexing
01557   if (count > 8){
01558         count = 8;
01559   }
01560
01561   for (i=0;i<count;i++){
01562         // send data to DAC, take care about bad values, they will not interfere with dac_sel_ch
01563 #if MCU_TYPE == MCU_TYPE_TMS320F2812 || MCU_TYPE == MCU_TYPE_TMS320F28335
01564         SpiaRegs.SPITXBUF = ((*(dac_values+i)&0x0FFF)|(*(dac_sel_ch+i)<<12));
01565 #endif
01566
01567 #if MCU_TYPE == MCU_TYPE_TMS570LS3137 //TODO: Implement better send of buffer with DMA
01568 //        *(dac_values+i) = ((*(dac_values+i)&0x0FFF)|(*(dac_sel_ch+i)<<12));
01569         data[i] = ((*(dac_values+i)&0x0FFF)|(*(dac_sel_ch+i)<<12));
01570 #endif
01571   }
01572 #if MCU_TYPE == MCU_TYPE_TMS570LS3137
01573   spiTransmitData(spiREG2, &spi2_conf, count, data);
01574 #endif
01575 }
01576
01577 // konfigurace USART modulu
01578 void MLC_SCI_init(unsigned long int baudrate)
01579 {
01580
01581 #if MCU_TYPE == MCU_TYPE_TMS320F28335
```

```
01582         unsigned int temp;
01583         EALLOW;
01584         // Select GPIOs to be Sci pins
01585         GpioCtrlRegs.GPBMUX2.bit.GPIO62 = 0x1;
01586         GpioCtrlRegs.GPBMUX2.bit.GPIO63 = 0x1;
01587         EDIS;
01588
01589         // 1-stopbit, bez parity, 8b data
01590         ScicRegs.SCICCR.all = 0x0007;
01591         ScicRegs.SCICCR.bit.LOOPBKENA = 0;
01592         // reset and allow RX and TX
01593         ScicRegs.SCICTL1.all =0x0003;
01594
01595 #if USE_SCI_TX_FIFO == 1
01596         ScicRegs.SCIFFTX.bit.SCIFFENA = 1;
01597         ScicRegs.SCIFFTX.bit.TXFIFOXRESET = 0;
01598         ScicRegs.SCIFFTX.bit.TXFIFOXRESET = 1;
01599 #endif
01600
01601 #if USE_SCI_RX_FIFO == 1
01602         ScicRegs.SCIFFTX.bit.SCIFFENA = 1;
01603         ScicRegs.SCIFFRX.bit.RXFIFORESET = 0;
01604         ScicRegs.SCIFFRX.bit.RXFIFORESET = 1;
01605 #endif
01606
01607         // calculate baudrate
01608         // 150000000/32/baudrate
01609         temp = ((150000000 >> 5)/baudrate)-1;
01610         // setup baudrate
01611         ScicRegs.SCIHBAUD = (temp & 0xFF00)>>8;
01612         ScicRegs.SCILBAUD = (temp & 0x00FF);
01613         // release from reset and allow RX and TX
01614         ScicRegs.SCICTL1.all =0x0023;
01615         // soft mode - it solves data corruption when debugger access DSP
01616         ScicRegs.SCIPRI.bit.SOFT = 0x1;
01617 #endif
01618
01619 #if MCU_TYPE == MCU_TYPE_TMS320F2812
01620         EALLOW;
01621         GpioMuxRegs.GPGMUX.bit.SCITXDB_GPIOG4 = 0x1;
01622         GpioMuxRegs.GPGMUX.bit.SCIRXDB_GPIOG5 = 0x1;
01623         EDIS;
01624
01625         ScibRegs.SCICCR.all = 0x0007;
01626         ScibRegs.SCICCR.bit.LOOPBKENA = 0;
01627         // reset and allow RX and TX
01628         ScibRegs.SCICTL1.all =0x0003;
01629
01630 #if USE_SCI_TX_FIFO == 1
01631         ScibRegs.SCIFFTX.bit.SCIFFENA = 1;
01632         ScibRegs.SCIFFTX.bit.TXFIFOXRESET = 0;
01633         ScibRegs.SCIFFTX.bit.TXFIFOXRESET = 1;
01634 #endif
01635
01636 #if USE_SCI_RX_FIFO == 1
01637         ScibRegs.SCIFFTX.bit.SCIFFENA = 1;
01638         ScibRegs.SCIFFRX.bit.RXFIFORESET = 0;
01639         ScibRegs.SCIFFRX.bit.RXFIFORESET = 1;
01640 #endif
01641 #endif
01642
01643 #if MCU_TYPE == MCU_TYPE_TMS320F2812 || MCU_TYPE == MCU_TYPE_TMS320F28335
01644         // calculate baudrate
01645         // 150000000/32/baudrate
01646         temp = ((150000000 >> 5)/baudrate)-1;
01647         // setup baudrate
01648         ScibRegs.SCIHBAUD = (temp & 0xFF00)>>8;
01649         ScibRegs.SCILBAUD = (temp & 0x00FF);
01650         // release from reset and allow RX and TX
01651         ScibRegs.SCICTL1.all =0x0023;
01652         // soft mode - it solves data corruption when debugger access DSP
01653         ScibRegs.SCIPRI.bit.SOFT = 0x1;
01654 #endif
01655 }
01656
01657 char MLC_SCI_char_avail(void)
01658 {
01659 #if MCU_TYPE == MCU_TYPE_TMS320F28335
01660 #if USE_SCI_RX_FIFO == 0
01661         return ScicRegs.SCIRXST.bit.RXRDY;
01662 #else
01663         return ScicRegs.SCIFFRX.bit.RXFFST;
01664 #endif
01665 #endif
01666
01667 #if MCU_TYPE == MCU_TYPE_TMS320F2812
01668 #if USE_SCI_RX_FIFO == 0
```

```
01669         return ScibRegs.SCIRXST.bit.RXRDY;
01670 #else
01671         return ScibRegs.SCIFFRX.bit.RXFIFST;
01672 #endif
01673 #endif
01674
01675 #if MCU_TYPE == MCU_TYPE_TMS570LS3137 //TODO: Implement it
01676         return 0;
01677 #endif
01678 }
01679
01680 char MLC_SCI_wait_send(void){
01681 #if MCU_TYPE == MCU_TYPE_TMS320F28335
01682         return ScicRegs.SCICTL2.bit.TXEMPTY;
01683 #endif
01684
01685 #if MCU_TYPE == MCU_TYPE_TMS320F2812
01686         return ScibRegs.SCICTL2.bit.TXEMPTY;
01687 #endif
01688
01689 #if MCU_TYPE == MCU_TYPE_TMS570LS3137 //TODO: Implement it
01690         return 0;
01691 #endif
01692 }
01693
01694 char MLC_SCI_recv_char(void)
01695 {
01696 #if MCU_TYPE == MCU_TYPE_TMS320F28335
01697         // wait for char
01698 #if USE_SCI_RX_FIFO == 0
01699         while (ScicRegs.SCIRXST.bit.RXRDY == 0);
01700 #else
01701         while (ScicRegs.SCIFFRX.bit.RXFFST == 0);
01702 #endif
01703         return ScicRegs.SCIRXBUF.all;
01704 #endif
01705
01706 #if MCU_TYPE == MCU_TYPE_TMS320F2812
01707 #if USE_SCI_RX_FIFO == 0
01708         while (ScibRegs.SCIRXST.bit.RXRDY == 0);
01709 #else
01710         while (ScibRegs.SCIFFRX.bit.RXFIFST == 0);
01711 #endif
01712         return ScibRegs.SCIRXBUF.all;
01713 #endif
01714
01715 #if MCU_TYPE == MCU_TYPE_TMS570LS3137 //TODO: Implement it
01716         return 0;
01717 #endif
01718
01719 }
01720
01721 // blocking send char
01722 void MLC_SCI_send_char(char c)
01723 {
01724 #if MCU_TYPE == MCU_TYPE_TMS320F28335
01725         // wait for send
01726 #if USE_SCI_TX_FIFO == 0
01727         while (ScicRegs.SCICTL2.bit.TXEMPTY == 0);
01728 #else
01729         while (ScicRegs.SCIFFTX.bit.TXFFST > 0xF);
01730 #endif
01731         ScicRegs.SCITXBUF = c;
01732 #endif
01733
01734 #if MCU_TYPE == MCU_TYPE_TMS320F2812
01735 #if USE_SCI_TX_FIFO == 0
01736         while (ScibRegs.SCICTL2.bit.TXEMPTY == 0);
01737 #else
01738         while (ScibRegs.SCIFFTX.bit.TXFFST > 0xF);
01739 #endif
01740         ScibRegs.SCITXBUF = c;
01741 #endif
01742
01743 #if MCU_TYPE == MCU_TYPE_TMS570LS3137 //TODO: Implement it
01744 #endif
01745 }
01746
01747 void MLC_SCI_send_char_wait(char c)
01748 {
01749 #if MCU_TYPE == MCU_TYPE_TMS320F28335
01750         MLC_SCI_send_char(c);
01751         while (!ScicRegs.SCICTL2.bit.TXEMPTY);
01752 #endif
01753
01754 #if MCU_TYPE == MCU_TYPE_TMS320F2812
01755         MLC_SCI_send_char(c);
```

```
01756         while (!ScibRegs.SCICTL2.bit.TXEMPTY);
01757 #endif
01758
01759 #if MCU_TYPE == MCU_TYPE_TMS570LS3137 //TODO: Implement it
01760 #endif
01761 }
01762
01763 // blocking send str
01764 void MLC_SCI_send_str(char* str)
01765 {
01766         while (*str != '\0')
01767         {
01768                 MLC_SCI_send_char(*str++);
01769         }
01770 }
01771
01772 // set pwr output while other untouched
01773 void MLC_PWR_on(uint16_t pwr){
01774         pwr_out_state |= pwr ;
01775         MLC_WRITE(WRITE_PWR_OUT, pwr_out_state);
01776 }
01777
01778 // reset pwr output while other untouched
01779 void MLC_PWR_off(uint16_t pwr){
01780         pwr_out_state &=  ~pwr;
01781         MLC_WRITE(WRITE_PWR_OUT, pwr_out_state);
01782 }
01783
01784 // set debug output while other untouched
01785 void MLC_DBG_set(uint16_t dbg){
01786         dbg_state |= dbg;
01787         MLC_WRITE(WRITE_DBGLEDS, dbg_state);
01788 }
01789
01790 // clear debug output while other untouched
01791 void MLC_DBG_clear(uint16_t dbg){
01792         dbg_state &=  ~dbg;
01793         MLC_WRITE(WRITE_DBGLEDS, dbg_state);
01794 }
01795
01796 /* EOF */
```