Using the Chip Support Register Configuration Macros

Platform Support Group

ABSTRACT

This document describes the Chip Support Register Configuration files provided for some Digital Media Processors (DMPs). This layer provides low-level register and bit field descriptions for the device and its peripherals, and a set of macros for basic register configuration. It may be used as a foundation for building complex drivers or on its own to perform register configuration and check peripheral status.

Contents

1 Overview of the Chip Support Register Configuration Layer

The Chip Support Register Configuration files provide register configuration support for each of the peripheral modules on selected Digital Media Processor (DMPs) through a set of C header files delivered in the Platform Support Package (PSP). Module-specific files provide register and bit field descriptions for a given peripheral, and a common file provides macros to read and modify hardware registers. Other common and system files provide for other device-specific definitions. See Table 1 for a list of supported devices.

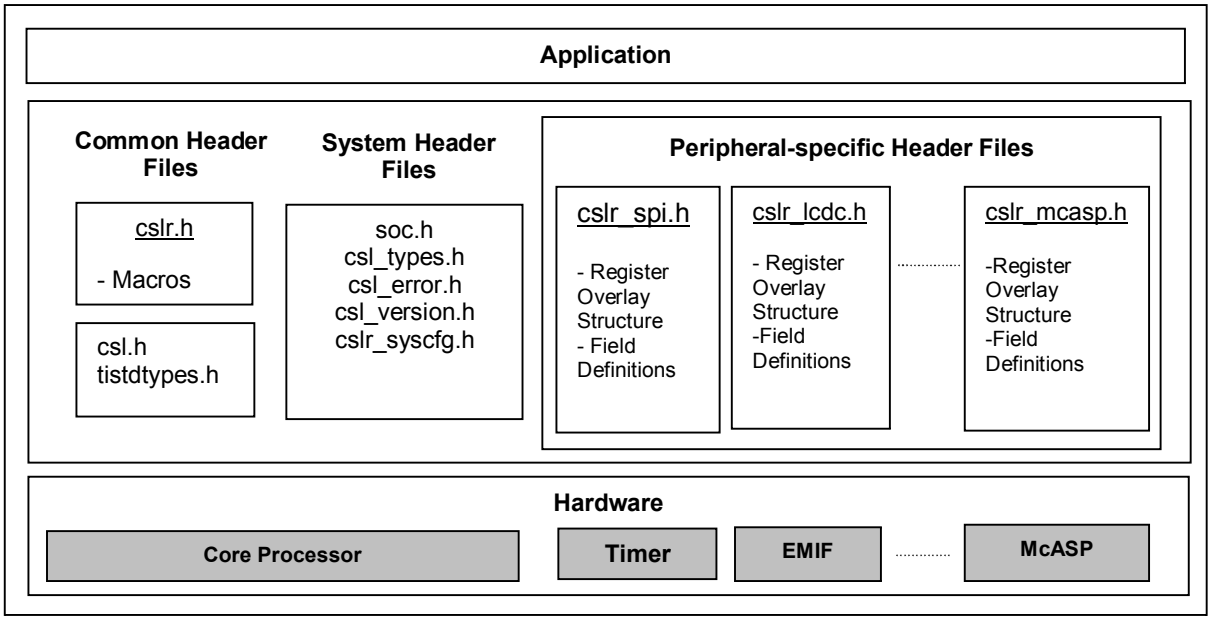
Family	Devices	Delivery Mechanism
OMAPL137	OMAPL137	DSP/BIOS PSP for the OMAPL137

Table 1. Chip Support Register Configuration Layer Supported Devices

1.1 Chip Support Register Configuration File Structure

The Chip Support Register Configuration files are made up of three types of header files: common files, system files, and peripheral-specific files. These files are summarized in Table 2 and in the figure below.

The Chip Support Register Configuration files are delivered in a Platform Support Package (PSP), in the directory ti/psp/*cslr*.



Chip Support Register Configuration File Structure

Common files are independent of a device or family, and independent of any specific registers. These define standard data types or macros. System files are specific to the device. They define peripheral instances, version information, error types, interrupt event IDs, interrupt routines, DMA channel structure, and they provide data types which may be specific to the device family.

In addition, each peripheral or module type is supported by a register configuration layer header file, which contains a register overlay structure and field definitions. The naming convention for peripheral-specific header files is cslr_<*per*>.h, where <*per*> is the abbreviation for the peripheral. For example, cslr_gpio.h is the header file for the GPIO peripheral.

> **Note:** Some peripherals are made up of multiple header file components. For example ethernet peripheral has multiple subcomponents and each of them would have CSLR file.

A system-level register layer header file named cslr_sysctl.h contains the register overlay structure and field definitions for the system module registers used for device configuration. This file also includes control registers for the timer, EDMA transfer controller and DDR2 memory controller. The other registers for these peripherals are supported in their respective peripheral header files. The memory map for the system module registers is summarized in the device datasheet.

The user need only include the peripheral header files and common header files required for the application.

File Name	File Type	Description
csl.h	Common	System initialization function.
cslr.h	Common	Macros for register and bit field manipulation.
tistdtypes.h	Common	Standard data types common to TI software products.
soc.h	Device	Peripheral instance definitions, peripheral base addresses, and other definitions common to the device, such as interrupt event IDs and DMA channel parameters.
csl_types.h	Device	Additional data types.
csl_error.h	Device	Global and peripheral-specific error codes.
csl_version.h	Device	CSL version and device ID strings.
cslr_<*per*>.h	Peripheral	Peripheral or module-specific header files, where <*per*> is the abbreviation for the peripheral.

Table 2. Chip Support Register Configuration Files

Key attributes of some of these files are described in more detail in the sections that follow.

1.2 Common File Attributes

The Chip Support Register Configuration layer defines eight macros in the file cslr.h. These macros allow the programmer to create, read, or write bit fields within a register. There are three different types of services: field make, field extract, and field insert. The macros summarized in Table 3 are described in detail in section 2.

Macro	Brief Description	Page
CSL_FMK	Field Make	7
CSL_FMKT	Field Make Token	7
CSL_FMKR	Field Make Raw	7

CSL_FEXT	Field Extract	8
CSL_FEXTR	Field Extract Raw	8
CSL_FINS	Field Insert	8
CSL_FINST	Field Insert Token	9
CSL_FINSR	Field Insert Raw	9

Table 3. Register Configuration Macros in cslr.h

Field make macros are used to create a register value from given input, and are written to the hardware register with the pointer to the register member in the Register Overlay Structure. Field make macros may be combined with OR operations in order to modify more than one field or the entire register. Unlike the field make macros, the field insert macros pass the register pointer as an argument, thus modify the specified register directly. Field extract macros read the register and return the value right-justified.

Raw macros provide the flexibility to modify or read one, multiple, or partial bit fields, because they designate the range of affected bits by location.

For macros that pass field name as an argument, the format for field name described in section 1.3.2 applies.

1.3 Peripheral-Specific File Attributes

1.3.1 Register Overlay Structure

The register overlay structure is defined for each peripheral in its register configuration layer header file, named cslr_<*per*>.h, where <*per*> is the abbreviation for the peripheral. The register overlay structure defines peripheral hardware registers, matching the hardware memory in sequence and register offset.

The naming convention of the register overlay structure type is CSL_<*Per*>Regs, where <*Per*> is the abbreviated peripheral type. The pointer type for the register overlay structure has the convention *CSL_<*Per*>RegsOvly. For example, the register overlay structure type for a Host Port Interface (HPI) is CSL_HpiRegs, and the pointer is *CSL_HpiRegsOvly.

By assigning the base address of the peripheral instance to the structure pointer, the structure members can be used to access the peripheral registers.

The format of the register overlay structure is as follows:

```
typedef struct  {
    volatile Uint32 REGISTER_1;
    volatile Uint32 REGISTER_2;
    :
    volatile Uint32 REGISTER_N;
} CSL_<Per>Regs;
```

The format of the register overlay structure pointer type definition is as follows:

```
typedef volatile CSL_<Per>Regs  *CSL_<Per>RegsOvly;
```

As an example, here are the register overlay structure and the pointer type definition from the file cslr_uart.h:

```
/**************************************************************************\
* Register Overlay Structure
\**************************************************************************/
typedef struct  {
    volatile Uint32 RBR;
    volatile Uint32 IER;
    volatile Uint32 IIR;
    volatile Uint32 LCR;
    volatile Uint32 MCR;
    volatile Uint32 LSR;
    volatile Uint32 RSVD0[2];
    volatile Uint32 DLL;
    volatile Uint32 DLH;
    volatile Uint32 REVID1;
    volatile Uint32 REVID2;
    volatile Uint32 PWREMU_MGMT;
    volatile Uint32 MDR;
} CSL_UartRegs;
/**************************************************************************\
* Overlay structure typedef definition
\**************************************************************************/
typedef volatile CSL_UartRegs            *CSL_UartRegsOvly;
```

1.3.2 Field Definitions

The register configuration layer header file for each peripheral also contains definitions for field mask and shift values and hardware reset values for registers and bit fields.

The naming convention for these constants is:

CSL_<*PER*>_<*REG*>_<*FIELD*>_<*ACTION*>, where <*PER*> is the peripheral name, <*REG*> is the register name, <*FIELD*> is the name of the bit field. <*ACTION*> stands for MASK, SHIFT, RESETVAL, or a constant token value.

The <*PER*>_<*REG*>_<*FIELD*> portion of the constants represents the field name. This is important to the explanation of register configuration macros in section 2.

1.3.3 Bit Field Definition Example

In the UART Line Status Register, consider the member bit fields, RXFIFOE, TEMT and THRE The register configuration header file cslr_uart.h provides the following definitions relevant to this register:

```
/* LSR */


#define CSL_UART_LSR_RXFIFOE_MASK (0x00000080u)
#define CSL_UART_LSR_RXFIFOE_SHIFT (0x00000007u)
#define CSL_UART_LSR_RXFIFOE_RESETVAL (0x00000000u)
/*----RXFIFOE Tokens----*/
#define CSL_UART_LSR_RXFIFOE_NOERROR (0x00000000u)
#define CSL_UART_LSR_RXFIFOE_ERROR (0x00000001u)

#define CSL_UART_LSR_TEMT_MASK (0x00000040u)
#define CSL_UART_LSR_TEMT_SHIFT (0x00000006u)
#define CSL_UART_LSR_TEMT_RESETVAL (0x00000001u)
/*----TEMT Tokens----*/
#define CSL_UART_LSR_TEMT_FULL (0x00000000u)
#define CSL_UART_LSR_TEMT_EMPTY (0x00000001u)

#define CSL_UART_LSR_THRE_MASK (0x00000020u)
#define CSL_UART_LSR_THRE_SHIFT (0x00000005u)
#define CSL_UART_LSR_THRE_RESETVAL (0x00000001u)
```

The field names for the RXFIFOE, TEMT and THRE fields are UART_LSR_RXFIFOE, UART_LSR_TEMT and UART_LSR_THRE, respectively.

The configuration file also defines tokens. For example, tokens for checking if Transmit Holding Register is empty or contains data via, CSL_UART_LSR_TEMT_FULL or CSL_UART_LSR_TEMT_FULL respectively.

2 Macro Reference

CSL_FMK *Field Make*

Macro CSL_FMK (field, val)

Arguments field Field name, in the format <PER_REG_FIELD>
val Value

Return Value Uint32

Description Shifts and AND masks absolute value (val) to specified field location. The result can then be written to the register using the register handle.

Evaluation
```
((val) << CSL_##PER_REG_FIELD##_SHIFT) &
CSL_##PER_REG_FIELD##_MASK
```

Example To set the Watchdog Timer Enable Bit (WDEN) in the Watchdog Timer Control Register (WDTCR) to ENABLE (1):
```
tmrRegs->WDTCR |= CSL_FMK (TMR_WDTCR_WDEN, 1);
```

CSL_FMKT *Field Make Token*

Macro CSL_FMKT (field, token)

Arguments field Field name, in the format <PER_REG_FIELD>
token Token

Return Value Uint32

Description Shifts and AND masks predefined symbolic constant (token) to specified field location (field). The result can then be written to the register using the register handle.

Evaluation
```
CSL_FMK(PER_REG_FIELD, CSL_##PER_REG_FIELD##_##TOKEN)
```

Example To set the Watchdog Timer Enable Bit (WDEN) in the Watchdog Timer Control Register (WDTCR) to ENABLE (1):
```
tmrRegs->WDTCR |= CSL_FMKT (TMR_WDTCR_WDEN, ENABLE);
```

CSL_FMKR *Field Make Raw*

Macro CSL_FMKR (msb, lsb, val)

Arguments msb Most significant bit of field
lsb Least significant bit of field
val Value

Return Value Uint32

Description Shifts and AND masks absolute value (val) to specified field location, specified by raw bit positions representing the most and least significant bits of the field (msb, lsb). The result can then be written to the register using the register handle.

Evaluation
```
((val) & ((1 << ((msb) - (lsb) + 1)) - 1)) << (lsb)
```

Example To set the Watchdog Timer Enable Bit (WDEN) in the Watchdog Timer Control Register (WDTCR) to ENABLE (1):
```
tmrRegs->WDTCR |= CSL_FMKR (14, 14, 1);
```

CSL_FEXT *Field Extract*

Macro CSL_FEXT (reg, field)

Arguments reg Register
field Field name, in the format <PER_REG_FIELD>

Return Value Uint32

Description Masks bit field (field) of specified register (reg) and right-justifies.

Evaluation
```
((reg) & CSL_##PER_REG_FIELD##_MASK) >>
CSL_##PER_REG_FIELD##_SHIFT
```

Example Check Timer Global Control Register (TGCR) to see if Timer 3:4 (TIM34RS) is in reset:
```
if ((CSL_FEXT (tmrRegs->TGCR,
TMR_TGCR_TIM34RS))==RESET_ON)…
```

CSL_FEXTR *Field Extract Raw*

Macro CSL_FEXTR (reg, msb, lsb)

Arguments reg Register
msb Most significant bit of field
lsb Least significant bit of field

Return Value Uint32

Description Masks bit field of register (reg) as specified by raw bit positions representing the most and least significant bits of the field (msb, lsb), and right-justifies.

Evaluation
```
((reg) >> (lsb)) & ((1 << ((msb) - (lsb) + 1)) - 1)
```

Example Check Timer Global Control Register (TGCR) to see if Timer 1:2 (TIM34RS) is in reset:
```
if ((CSL_FEXTR (tmrRegs->TGCR, 0, 0))==RESET_ON) …
```

CSL_FINS *Field Insert*

Macro CSL_FINS (reg, field, val)

Arguments reg Register
field Field name in the format <PER_REG_FIELD>
val Value

Return Value None

Description Inserts the absolute value (val) at the specified field (field) in the register (reg). This macro modifies the register.

Evaluation
```
(reg) = ((reg) & ~CSL_##PER_REG_FIELD##_MASK)
    | CSL_FMK(PER_REG_FIELD, val)
```

Example Set the Enable Mode for timer 3:4 (ENAMODE34) in the Timer Control Register (TCR) to disabled:
```
CSL_FINS (tmrRegs->TCR, TMR_TCR_ENAMODE34, 0);
```

CSL_FINST *Field Insert Token*

Macro CSL_FINST (reg, field, token)

Arguments
reg Register
field Field name, in the format <PER_REG_FIELD>
token Token

Return Value None

Description Inserts predefined symbolic constant (token) at the specified field (field) in the register (reg). This macro modifies the register.

Evaluation
```
CSL_FINS((reg), PER_REG_FIELD,
CSL_##PER_REG_FIELD##_##TOKEN)
```

Example Set the Enable Mode for timer 3:4 (ENAMODE34) in the Timer Control Register (TCR) to disabled:
```
CSL_FINST (tmrRegs->TCR, TMR_TCR_ENAMODE34, DISABLED);
```

CSL_FINSR *Field Insert Raw*

Macro CSL_FINSR (reg, msb, lsb, val)

Arguments
reg Register
msb Most significant bit of field
lsb Least significant bit of field
val Value

Return Value None

Description Inserts the absolute value (val) in bit field of register (reg), as specified by raw bit positions representing the most and least significant bits of the field (msb, lsb). This macro modifies the register.

Evaluation
```
(reg) = ((reg)&~(((1<<((msb)-(lsb)+1))-1)<<(lsb)))
    | CSL_FMKR(msb,lsb,val)
```

Example Set the Enable Mode for timer 3:4 (ENAMODE34) in the Timer Control Register (TCR) to disabled:
```
CSL_FINSR (tmrRegs->TCR, 23, 22, 0);
```

3 Examples

This section contains usage examples for the Chip Support Register Configuration Macros. The Platform Support Package (PSP) also provides working examples in the *ti/psp/cslr/<evm>/examples* directory.

3.1 EMIFB Example

This example performs the following steps:

1. Enables the DDR2 module
2. Sets up the hardware to default values and Normal Mode
3. Writes the Invalid values into DDR2 SDRAM area to over write the previous values
4. Writes valid data
5. Does the data comparison to ensure the written data is proper or not
6. Displays the messages based on step 5

```
#include <ti/psp/iom/cslr/csl_types.h>
#include <ti/psp/iom/cslr/soc_OMAPL137.h>
#include <ti/psp/iom/cslr/cslr_emifb.h>
#include <ti/psp/iom/cslr/cslr_syscfg_OMAPL137.h>
#include <ti/psp/iom/cslr/cslr_psc_OMAPL137.h>
#include <stdio.h>


CSL_SyscfgRegsOvly  sysRegs     = (CSL_SyscfgRegsOvly)CSL_SYSCFG_0_REGS;
CSL_PscRegsOvly    psc1Regs    = (CSL_PscRegsOvly)CSL_PSC_1_REGS;
CSL_EmifbRegsOvly    emifbRegs = (CSL_EmifbRegsOvly)CSL_EMIFB_0_REGS;

/* DDR Base address */
Uint32 ddr_base;
/* DDR size */
Uint32 ddr_size;

/* Function to test SDRAM read write */
int sdramTest( void );

/* Function to initialize EMIFB */
int emifbInit( void );

/* Function to configure SDRAM */
int configSdram( void );

/* Function for testing invalid SDRAM address range */
Uint32 meminvaddr32( Uint32 , Uint32 );

/* Function for testing valid SDRAM address range */
Uint32 memaddr32( Uint32 , Uint32 );

/* Function for filling an SDRAM address range */
Uint32 memfill32( Uint32 , Uint32 , Uint32 );
```

```
int main( void )
{
  int result = 0;

  /* Intialize EMIF */
  result = emifbInit();
  if(result < 0)
  {
      printf("EMIFB Initialization failed\n");
      return result;
  }
  else
  {
      printf("EMIFB Initialization success\n");
  }

  /* Configure SDRAM */
  result = configSdram();
  if(result < 0)
  {
      printf("SDRAM Configuration test failed\n");
      return result;
  }
  else
  {
      printf("SDRAM Configuration success\n");
  }

  /* Run SDRAM write/read test */
  result = sdramTest();
  if(result != 0)
  {
      printf("SDRAM Read/Write example test failed\n");
      return result;
  }
  else
  {
      printf("SDRAM Read/Write example test success\n");
  }

  return 0;
}


int emifbInit( void )
{
    volatile int pscTimeoutCount = 10240;
    int result = 0;

    sysRegs->KICK0R = 0x83e70b13;  // Kick0 register + data (unlock)
    sysRegs->KICK1R = 0x95a4f1e0;  // Kick1 register + data (unlock)

    /* Set PINMUX's for enabling EMIFB */

    //0x11111188 : EMIFB, Check EMU0/RTCK : TND Verify 15_12 bits
    sysRegs->PINMUX0  = ( CSL_SYSCFG_PINMUX0_PINMUX0_31_28_EMB_WE    <<
CSL_SYSCFG_PINMUX0_PINMUX0_31_28_SHIFT |
                          CSL_SYSCFG_PINMUX0_PINMUX0_27_24_EMB_RAS   <<
CSL_SYSCFG_PINMUX0_PINMUX0_27_24_SHIFT |
```

```
                            CSL_SYSCFG_PINMUX0_PINMUX0_23_20_EMB_CAS   <<
CSL_SYSCFG_PINMUX0_PINMUX0_23_20_SHIFT |
                            CSL_SYSCFG_PINMUX0_PINMUX0_19_16_EMB_CS0   <<
CSL_SYSCFG_PINMUX0_PINMUX0_19_16_SHIFT |
                            CSL_SYSCFG_PINMUX0_PINMUX0_15_12_RESERVED1 <<
CSL_SYSCFG_PINMUX0_PINMUX0_15_12_SHIFT |
                            CSL_SYSCFG_PINMUX0_PINMUX0_11_8_EMB_SDCKE  <<
CSL_SYSCFG_PINMUX0_PINMUX0_11_8_SHIFT  |
                            CSL_SYSCFG_PINMUX0_PINMUX0_7_4_EMU0        <<
CSL_SYSCFG_PINMUX0_PINMUX0_7_4_SHIFT   |
                            CSL_SYSCFG_PINMUX0_PINMUX0_3_0_RTCK        <<
CSL_SYSCFG_PINMUX0_PINMUX0_3_0_SHIFT   );

    //0x11111111;  EMIFB
    sysRegs->PINMUX1  = ( CSL_SYSCFG_PINMUX1_PINMUX1_31_28_EMB_A5   <<
CSL_SYSCFG_PINMUX1_PINMUX1_31_28_SHIFT |
                            CSL_SYSCFG_PINMUX1_PINMUX1_27_24_EMB_A4   <<
CSL_SYSCFG_PINMUX1_PINMUX1_27_24_SHIFT |
                            CSL_SYSCFG_PINMUX1_PINMUX1_23_20_EMB_A3   <<
CSL_SYSCFG_PINMUX1_PINMUX1_23_20_SHIFT |
                            CSL_SYSCFG_PINMUX1_PINMUX1_19_16_EMB_A2   <<
CSL_SYSCFG_PINMUX1_PINMUX1_19_16_SHIFT |
                            CSL_SYSCFG_PINMUX1_PINMUX1_15_12_EMB_A1   <<
CSL_SYSCFG_PINMUX1_PINMUX1_15_12_SHIFT |
                            CSL_SYSCFG_PINMUX1_PINMUX1_11_8_EMB_A0    <<
CSL_SYSCFG_PINMUX1_PINMUX1_11_8_SHIFT  |
                            CSL_SYSCFG_PINMUX1_PINMUX1_7_4_EMB_BA0    <<
CSL_SYSCFG_PINMUX1_PINMUX1_7_4_SHIFT   |
                            CSL_SYSCFG_PINMUX1_PINMUX1_3_0_EMB_BA1    <<
CSL_SYSCFG_PINMUX1_PINMUX1_3_0_SHIFT   );

    //0x11111111;  EMIFB
    sysRegs->PINMUX2  = ( CSL_SYSCFG_PINMUX2_PINMUX2_31_28_EMB_D31 <<
CSL_SYSCFG_PINMUX2_PINMUX2_31_28_SHIFT |
                            CSL_SYSCFG_PINMUX2_PINMUX2_27_24_EMB_A12 <<
CSL_SYSCFG_PINMUX2_PINMUX2_27_24_SHIFT |
                            CSL_SYSCFG_PINMUX2_PINMUX2_23_20_EMB_A11 <<
CSL_SYSCFG_PINMUX2_PINMUX2_23_20_SHIFT |
                            CSL_SYSCFG_PINMUX2_PINMUX2_19_16_EMB_A10 <<
CSL_SYSCFG_PINMUX2_PINMUX2_19_16_SHIFT |
                            CSL_SYSCFG_PINMUX2_PINMUX2_15_12_EMB_A9  <<
CSL_SYSCFG_PINMUX2_PINMUX2_15_12_SHIFT |
                            CSL_SYSCFG_PINMUX2_PINMUX2_11_8_EMB_A8   <<
CSL_SYSCFG_PINMUX2_PINMUX2_11_8_SHIFT  |
                            CSL_SYSCFG_PINMUX2_PINMUX2_7_4_EMB_A7    <<
CSL_SYSCFG_PINMUX2_PINMUX2_7_4_SHIFT   |
                            CSL_SYSCFG_PINMUX2_PINMUX2_3_0_EMB_A6    <<
CSL_SYSCFG_PINMUX2_PINMUX2_3_0_SHIFT   );

    //0x11111111;  EMIFB
    sysRegs->PINMUX3  = ( CSL_SYSCFG_PINMUX3_PINMUX3_31_28_EMB_D23 <<
CSL_SYSCFG_PINMUX3_PINMUX3_31_28_SHIFT |
                            CSL_SYSCFG_PINMUX3_PINMUX3_27_24_EMB_D24 <<
CSL_SYSCFG_PINMUX3_PINMUX3_27_24_SHIFT |
                            CSL_SYSCFG_PINMUX3_PINMUX3_23_20_EMB_D25 <<
CSL_SYSCFG_PINMUX3_PINMUX3_23_20_SHIFT |
                            CSL_SYSCFG_PINMUX3_PINMUX3_19_16_EMB_D26 <<
CSL_SYSCFG_PINMUX3_PINMUX3_19_16_SHIFT |
                            CSL_SYSCFG_PINMUX3_PINMUX3_15_12_EMB_D27 <<
CSL_SYSCFG_PINMUX3_PINMUX3_15_12_SHIFT |
```

```
                          CSL_SYSCFG_PINMUX3_PINMUX3_11_8_EMB_D28  <<
CSL_SYSCFG_PINMUX3_PINMUX3_11_8_SHIFT  |
                          CSL_SYSCFG_PINMUX3_PINMUX3_7_4_EMB_D29   <<
CSL_SYSCFG_PINMUX3_PINMUX3_7_4_SHIFT   |
                          CSL_SYSCFG_PINMUX3_PINMUX3_3_0_EMB_D30   <<
CSL_SYSCFG_PINMUX3_PINMUX3_3_0_SHIFT   );

    //0x11111111;  EMIFB
    sysRegs->PINMUX4  = ( CSL_SYSCFG_PINMUX4_PINMUX4_31_28_EMB_WE_DQM3  <<
CSL_SYSCFG_PINMUX4_PINMUX4_31_28_SHIFT |
                          CSL_SYSCFG_PINMUX4_PINMUX4_27_24_EMB_D16      <<
CSL_SYSCFG_PINMUX4_PINMUX4_27_24_SHIFT |
                          CSL_SYSCFG_PINMUX4_PINMUX4_23_20_EMB_D17      <<
CSL_SYSCFG_PINMUX4_PINMUX4_23_20_SHIFT |
                          CSL_SYSCFG_PINMUX4_PINMUX4_19_16_EMB_D18      <<
CSL_SYSCFG_PINMUX4_PINMUX4_19_16_SHIFT |
                          CSL_SYSCFG_PINMUX4_PINMUX4_15_12_EMB_D19      <<
CSL_SYSCFG_PINMUX4_PINMUX4_15_12_SHIFT |
                          CSL_SYSCFG_PINMUX4_PINMUX4_11_8_EMB_D20       <<
CSL_SYSCFG_PINMUX4_PINMUX4_11_8_SHIFT  |
                          CSL_SYSCFG_PINMUX4_PINMUX4_7_4_EMB_D21        <<
CSL_SYSCFG_PINMUX4_PINMUX4_7_4_SHIFT   |
                          CSL_SYSCFG_PINMUX4_PINMUX4_3_0_EMB_D22        <<
CSL_SYSCFG_PINMUX4_PINMUX4_3_0_SHIFT   );

    //0x11111111;  EMIFB
    sysRegs->PINMUX5  = ( CSL_SYSCFG_PINMUX5_PINMUX5_31_28_EMB_D6       <<
CSL_SYSCFG_PINMUX5_PINMUX5_31_28_SHIFT |
                          CSL_SYSCFG_PINMUX5_PINMUX5_27_24_EMB_D5       <<
CSL_SYSCFG_PINMUX5_PINMUX5_27_24_SHIFT |
                          CSL_SYSCFG_PINMUX5_PINMUX5_23_20_EMB_D4       <<
CSL_SYSCFG_PINMUX5_PINMUX5_23_20_SHIFT |
                          CSL_SYSCFG_PINMUX5_PINMUX5_19_16_EMB_D3       <<
CSL_SYSCFG_PINMUX5_PINMUX5_19_16_SHIFT |
                          CSL_SYSCFG_PINMUX5_PINMUX5_15_12_EMB_D2       <<
CSL_SYSCFG_PINMUX5_PINMUX5_15_12_SHIFT |
                          CSL_SYSCFG_PINMUX5_PINMUX5_11_8_EMB_D1        <<
CSL_SYSCFG_PINMUX5_PINMUX5_11_8_SHIFT  |
                          CSL_SYSCFG_PINMUX5_PINMUX5_7_4_EMB_D0         <<
CSL_SYSCFG_PINMUX5_PINMUX5_7_4_SHIFT   |
                          CSL_SYSCFG_PINMUX5_PINMUX5_3_0_EMB_WE_DQM2    <<
CSL_SYSCFG_PINMUX5_PINMUX5_3_0_SHIFT   );

    //0x11111111;  EMIFB
    sysRegs->PINMUX6  = ( CSL_SYSCFG_PINMUX6_PINMUX6_31_28_EMB_D14      <<
CSL_SYSCFG_PINMUX6_PINMUX6_31_28_SHIFT |
                          CSL_SYSCFG_PINMUX6_PINMUX6_27_24_EMB_D13      <<
CSL_SYSCFG_PINMUX6_PINMUX6_27_24_SHIFT |
                          CSL_SYSCFG_PINMUX6_PINMUX6_23_20_EMB_D12      <<
CSL_SYSCFG_PINMUX6_PINMUX6_23_20_SHIFT |
                          CSL_SYSCFG_PINMUX6_PINMUX6_19_16_EMB_D11      <<
CSL_SYSCFG_PINMUX6_PINMUX6_19_16_SHIFT |
                          CSL_SYSCFG_PINMUX6_PINMUX6_15_12_EMB_D10      <<
CSL_SYSCFG_PINMUX6_PINMUX6_15_12_SHIFT |
                          CSL_SYSCFG_PINMUX6_PINMUX6_11_8_EMB_D9        <<
CSL_SYSCFG_PINMUX6_PINMUX6_11_8_SHIFT  |
                          CSL_SYSCFG_PINMUX6_PINMUX6_7_4_EMB_D8         <<
CSL_SYSCFG_PINMUX6_PINMUX6_7_4_SHIFT   |
                          CSL_SYSCFG_PINMUX6_PINMUX6_3_0_EMB_D7         <<
CSL_SYSCFG_PINMUX6_PINMUX6_3_0_SHIFT   );
```

```
    //0x11111111;  EMIFB, SPI0
    sysRegs->PINMUX7  = ( CSL_SYSCFG_PINMUX7_PINMUX7_31_28_SPI0_SCS0    <<
CSL_SYSCFG_PINMUX7_PINMUX7_31_28_SHIFT |
                          CSL_SYSCFG_PINMUX7_PINMUX7_27_24_SPI0_ENA     <<
CSL_SYSCFG_PINMUX7_PINMUX7_27_24_SHIFT |
                          CSL_SYSCFG_PINMUX7_PINMUX7_23_20_SPI0_CLK     <<
CSL_SYSCFG_PINMUX7_PINMUX7_23_20_SHIFT |
                          CSL_SYSCFG_PINMUX7_PINMUX7_19_16_SPI0_SIMO0   <<
CSL_SYSCFG_PINMUX7_PINMUX7_19_16_SHIFT |
                          CSL_SYSCFG_PINMUX7_PINMUX7_15_12_SPI0_SOMI0   <<
CSL_SYSCFG_PINMUX7_PINMUX7_15_12_SHIFT |
                          CSL_SYSCFG_PINMUX7_PINMUX7_11_8_EMB_WE_DQM0   <<
CSL_SYSCFG_PINMUX7_PINMUX7_11_8_SHIFT  |
                          CSL_SYSCFG_PINMUX7_PINMUX7_7_4_EMB_WE_DQM1    <<
CSL_SYSCFG_PINMUX7_PINMUX7_7_4_SHIFT   |
                          CSL_SYSCFG_PINMUX7_PINMUX7_3_0_EMB_D15        <<
CSL_SYSCFG_PINMUX7_PINMUX7_3_0_SHIFT   );


    /* Bring the EMIFB module out of reset */
    // deassert EMIFB local PSC reset and set NEXT state to ENABLE
    psc1Regs->MDCTL[CSL_PSC_EMIFB] = CSL_FMKT( PSC_MDCTL_NEXT, ENABLE )
                                | CSL_FMKT( PSC_MDCTL_LRST, DEASSERT );
    // move EMIFB PSC to Next state
    psc1Regs->PTCMD = CSL_FMKT(  PSC_PTCMD_GO0, SET );

    // wait for transition
    while( ( CSL_FEXT( psc1Regs->MDSTAT[CSL_PSC_EMIFB], PSC_MDSTAT_STATE )
             != CSL_PSC_MDSTAT_STATE_ENABLE )  && (pscTimeoutCount > 0) )
    {
        pscTimeoutCount--;
    }

    if(pscTimeoutCount == 0)
    {
        printf("EMIFB module power up timed out\n");
        result= -1;
    }

    return result;
}


int configSdram (void )
{
    volatile Uint32 temp = 0;

    // ISSI IS42S16160B-6BL SDRAM, 2 x 16M x 16 (32-bit data path), 133MHz
    temp = emifbRegs->SDCFG;
    temp =  ( ( CSL_EMIFB_SDCFG_TIMUNLOCK_SET << CSL_EMIFB_SDCFG_TIMUNLOCK_SHIFT)      |
// Unlock timing registers
              ( CSL_EMIFB_SDCFG_CL_TWO << CSL_EMIFB_SDCFG_CL_SHIFT )
|   // CAS latency is 2
              ( CSL_EMIFB_SDCFG_IBANK_FOUR << CSL_EMIFB_SDCFG_IBANK_SHIFT )
|   // 4 bank SDRAM devices
              ( CSL_EMIFB_SDCFG_PAGESIZE_512W_PAGE << CSL_EMIFB_SDCFG_PAGESIZE_SHIFT )
);   // 512-word pages requiring 9 column address bits
    emifbRegs->SDCFG = temp;


    temp = emifbRegs->SDRFC;
```

```
    temp =  ( ( CSL_EMIFB_SDRFC_LP_MODE_LPMODE << CSL_EMIFB_SDRFC_LP_MODE_SHIFT)
|   // Low power mode disabled
              ( CSL_EMIFB_SDRFC_MCLKSTOP_EN_MCLKSTOP_DIS <<
CSL_EMIFB_SDRFC_MCLKSTOP_EN_SHIFT) |   // MCLK stoping disabled
              ( CSL_EMIFB_SDRFC_SR_PD_SELF_REFRESH << CSL_EMIFB_SDRFC_SR_PD_SHIFT)
|   // Selects self refresh instead of power down
              ( 1040 << CSL_EMIFB_SDRFC_REFRESH_RATE_SHIFT)
);   // Refresh rate = 7812.5ns / 7.5ns
    emifbRegs->SDRFC = temp;


    temp = emifbRegs->SDTIM1;
    temp =  ( ( 25 << CSL_EMIFB_SDTIM1_T_RFC_SHIFT )|   // (67.5ns / 7.55ns) - 1 = TRFC
@ 133MHz
              ( 2  << CSL_EMIFB_SDTIM1_T_RP_SHIFT  )|   // (20ns / 7.5ns) - 1 =TRP
              ( 2  << CSL_EMIFB_SDTIM1_T_RCD_SHIFT )|   // (20ns / 7.5ns) - 1 = TRCD
              ( 1  << CSL_EMIFB_SDTIM1_T_WR_SHIFT  )|   // (14ns / 7.5ns) - 1 = TWR
              ( 5  << CSL_EMIFB_SDTIM1_T_RAS_SHIFT )|   // (45ns / 7.5ns) - 1 = TRAS
              ( 8  << CSL_EMIFB_SDTIM1_T_RC_SHIFT  )|   // (67.5ns / 7.5ns) - 1 = TRC
              ( 2  << CSL_EMIFB_SDTIM1_T_RRD_SHIFT ) );   // *(((4 * 14ns) + (2 * 7.5ns))
/ (4 * 7.5ns)) -1. = TRRD
                                                          // but it says to use this
formula if 8 banks but only 4 are used here.
                                                          // and SDCFG1 register only
suports upto 4 banks.
    emifbRegs->SDTIM1 = temp;


    temp = emifbRegs->SDTIM2;
    temp = ( (14 << CSL_EMIFB_SDTIM2_T_RAS_MAX_SHIFT) |   // not sure how they got this
number. the datasheet says value should be
                                                          // "Maximum number of
refresh_rate intervals from Activate to Precharge command"
                                                          // but has no equation. TRASMAX
is 120k.
              ( 9 << CSL_EMIFB_SDTIM2_T_XSR_SHIFT)     |   // ( 70 / 7.5) - 1
              ( 5 << CSL_EMIFB_SDTIM2_T_CKE_SHIFT)     );        // ( 45 / 7.5 ) - 1
    emifbRegs->SDTIM2 = temp;


    temp = emifbRegs->SDCFG ;
    temp = ( ( CSL_EMIFB_SDCFG_SDREN_SDR_ENABLE << CSL_EMIFB_SDCFG_SDREN_SHIFT)
|
              ( CSL_EMIFB_SDCFG_TIMUNLOCK_CLEAR << CSL_EMIFB_SDCFG_TIMUNLOCK_SHIFT)
|    // lock timing registers
              ( CSL_EMIFB_SDCFG_CL_TWO << CSL_EMIFB_SDCFG_CL_SHIFT )
|    // CAS latency is 2
              ( CSL_EMIFB_SDCFG_IBANK_FOUR << CSL_EMIFB_SDCFG_IBANK_SHIFT )
|    // 4 bank SDRAM devices
              ( CSL_EMIFB_SDCFG_PAGESIZE_512W_PAGE << CSL_EMIFB_SDCFG_PAGESIZE_SHIFT )  );
// 512-word pages requiring 9 column address bits
    emifbRegs->SDCFG = temp;
    return 0;
}

int sdramTest( void )
{
        Int16 i, errors = 0;

    ddr_base = 0xc0004000;       // DDR memory
    ddr_size = 0x00010000;       // 1 MB
```

```
    printf( "  > Data test (quick)\n" );
    if ( memfill32( ddr_base, ddr_size, 0xFFFFFFFF ) )
        errors += 1;

    if ( memfill32( ddr_base, ddr_size, 0xAAAAAAAA ) )
        errors += 2;

    if ( memfill32( ddr_base, ddr_size, 0x55555555 ) )
        errors += 4;

    if ( memfill32( ddr_base, ddr_size, 0x00000000 ) )
        errors += 8;

    if ( errors )
        printf( "        > Error = 0x%x\n", errors );

#if(1)
    ddr_base = 0xc0004000;       // DDR memory
    ddr_size = 0x03FFC000;       // 63 MB+

    printf( "  > Addr test (quick)\n    " );
    for (i = 0; i < 11; i++)
    {
        printf("A%d ", i + 16);
        if ( memaddr32( ddr_base + (0x10000 << i), 0x10000 ) )
        {
            printf("(X) ");
            errors += 16;
        }
    }
    printf("\n");

    printf( "  > Inv addr test (quick)\n    " );
    for (i = 0; i < 11; i++)
    {
        printf("A%d ", i + 16);
        if ( meminvaddr32( ddr_base + (0x10000 << i), 0x10000 ) )
        {
            printf("(X) ");
            errors += 16;
        }
    }
    printf("\n");
#endif
    return errors;
}

Uint32 meminvaddr32( Uint32 start, Uint32 len )
{
    Uint32 i;
    Uint32 end = start + len;
    Uint32 errorcount = 0;
    Uint32 *pdata;

    /* Write Pattern */
    pdata = (Uint32 *)start;
    for ( i = start; i < end; i += 4 )
    {
        *pdata++ = ~i;
    }
```

```
    /* Read Pattern */
    pdata = (Uint32 *)start;
    for ( i = start; i < end; i += 4 )
    {
        if ( *pdata++ != ~i )
        {
            errorcount++;
            break;
        }
    }

    return errorcount;
}

Uint32 memaddr32( Uint32 start, Uint32 len )
{
    Uint32 i;
    Uint32 end = start + len;
    Uint32 errorcount = 0;
    Uint32 *pdata;


    /* Write Pattern */
    pdata = (Uint32 *)start;
    for ( i = start; i < end; i += 16 )
    {
        *pdata++ = i;
        *pdata++ = i + 4;
        *pdata++ = i + 8;
        *pdata++ = i + 12;
    }

    /* Read Pattern */
    pdata  = (Uint32 *)start;
    for ( i = start; i < end; i += 4 )
    {
        if ( *pdata++ != i )
        {
            errorcount++;
            break;
        }
    }

    return errorcount;
}

Uint32 memfill32( Uint32 start, Uint32 len, Uint32 val )
{
    Uint32 i;
    Uint32 end = start + len;
    Uint32 errorcount = 0;
    Uint32 *pdata;

    /* Write Pattern */
    pdata = (Uint32 *)start;
    for ( i = start; i < end; i += 4 )
    {
        *pdata++ = val;
    }
```

```
    /* Read Pattern */
    pdata = (Uint32 *)start;
    for ( i = start; i < end; i += 4 )
    {
        if ( *pdata++ != val )
        {
            errorcount++;
            break;
        }
    }

    return errorcount;
}
```

3.2 PLLC Example

The given example describes the delay routine, main routine which calls example routine, actual routine which configures the PLLC.

```
#include <stdio.h>
#include <ti/psp/iom/cslr/cslr_pllc.h>
#include <ti/psp/iom/cslr/soc_OMAPL137.h>

static void setupPll1(int pll_multiplier);
static int test_pll1();

/* Pointer to register overlay structure */
CSL_PllcRegsOvly pllcRegs = ((CSL_PllcRegsOvly)CSL_PLLC_0_REGS);

/*
 * ======================================================================
 *   @func   sw_wait
 *
 *   @desc
 *     This is the delay routine.
 *
 * ======================================================================
 */
void sw_wait(int delay)
{
    volatile int i;
    for( i = 0; i < delay; i++ ) {
    }
}

/*
 * ======================================================================
 *   @func   main
 *
 *   @desc
 *     This is the main routine which calls example routine.
 *
 * ======================================================================
 */

int main()
{
    printf("Configure PLL1 with register layer macros\n");
    printf("Please wait System PLL Initialization is in Progress.....\n");
```

```
    return(test_pll1());
}

/*
 * ============================================================================
 *   @func   setupPll1
 *
 *   @desc
 *     This is the actual routine which configures PLL0.
 *
 * ============================================================================
 */
void setupPll1(int pll_multiplier)
{
    /* Set PLLENSRC '0', PLL Enable(PLLEN) selection is controlled through MMR */
    CSL_FINST(pllcRegs->PLLCTL, PLLC_PLLCTL_PLLENSRC, CLEAR);

    /*Set PLL BYPASS MODE */
    CSL_FINST(pllcRegs->PLLCTL, PLLC_PLLCTL_PLLEN, BYPASS);

    /*wait for some cycles to allow PLLEN mux switches properly to bypass clock*/
    sw_wait(150);

    /* Reset the PLL */
    CSL_FINST(pllcRegs->PLLCTL, PLLC_PLLCTL_PLLRST, ASSERT);

    /*PLL stabilisation time*/
    sw_wait(1500);

    /*Program PREDIV Reg, POSTDIV register and OSCDIV1 Reg
    1.predvien_pi is set to '1'
    2.prediv_ratio_lock_pi is set to '1', RATIO field of PREDIV is locked
    3.Set the PLLM Register
    4.Dont program POSTDIV Register
    */

    /* Set PLL Multiplier */
    pllcRegs->PLLM = pll_multiplier;

    /*wait for PLL to Reset properly=>PLL reset Time*/
    sw_wait(128);

    /*Bring PLL out of Reset*/
    CSL_FINST(pllcRegs->PLLCTL, PLLC_PLLCTL_PLLRST, DEASSERT);

    /*Wait for PLL to LOCK atleast 2000 MXI clock or Reference clock cycles*/
    sw_wait(2000);

    /*Enable the PLL Bit of PLLCTL*/
    CSL_FINST(pllcRegs->PLLCTL, PLLC_PLLCTL_PLLEN, PLL);
}
```

```
/*
 * ============================================================================
 *   @func   test_pll0
 *
 *   @desc
 *     This is the dummy function.
 *
 * ============================================================================
 */
int test_pll1()
{
    setupPll1(20);

    printf("PLL1 has been configured\n");

    return(0);
}
```

3.3 GPIO Example

This example demonstrates the use of GPIO module. The sample does this by configuring the GPIO pin GPIO0_7(configured as output pin) as an interrupt pin. The task then sets the status of the pin to high to trigger an interrupt which is then serviced by an ISR function.

```
#include <stdio.h>
#include <c6x.h>
#include <ti/psp/iom/cslr/cslr_gpio.h>
#include <ti/psp/iom/cslr/cslr_syscfg_OMAPL137.h>
#include <ti/psp/iom/cslr/soc_OMAPL137.h>
#include <ti/psp/iom/cslr/cslr_psc_OMAPL137.h>
#include <ti/psp/iom/cslr/cslr_intc.h>

/*============================================================================*/
/*                            GLOBAL VARIABLES                                */
/*============================================================================*/

/* sys config registers overlay                                               */
CSL_SyscfgRegsOvly   sysRegs  = (CSL_SyscfgRegsOvly)(CSL_SYSCFG_0_REGS);
/* Psc register overlay                                                       */
CSL_PscRegsOvly      psc1Regs = (CSL_PscRegsOvly)(CSL_PSC_1_REGS);
/* Gpio register overlay                                                      */
CSL_GpioRegsOvly     gpioRegs = (CSL_GpioRegsOvly)(CSL_GPIO_0_REGS);
/* Interrupt Controller Register Overlay                                      */
CSL_IntcRegsOvly intcRegs = (CSL_IntcRegsOvly)CSL_INTC_0_REGS;

/*============================================================================*/
/*                        EXTERNAL FUNCTION PROTOTYPES                        */
/*============================================================================*/

extern void intcVectorTable(void);

static void delay(Uint32 count);


volatile Int32 status = 0;

#define GPIO_LPSC_NUM   3
#define GPIO0_EVENT    65
```

```
#define MAX_BLINK       4



void gpioExample(void)
{
    Uint32  ledBlinkCount = 0;
    volatile Uint32 temp = 0;
    volatile Uint32 pscTimeoutCount = 10240u;

    /* Key to be written to enable the pin mux registers to be written        */
    sysRegs->KICK0R = 0x83e70b13;
    sysRegs->KICK1R = 0x95A4F1E0;

    /* mux between EMA_D8 and GPIO0_8 : enable GPIO0_8 (User Switch - "SW3-1")  */
    sysRegs->PINMUX14 = ( (CSL_SYSCFG_PINMUX14_PINMUX14_27_24_GPIO0_8) << \
                         (CSL_SYSCFG_PINMUX14_PINMUX14_27_24_SHIFT) );

    /* mux between EMA_D12 and GPIO0_12 : enable GPIO0_12 (User Led - "DS1")    */
    sysRegs->PINMUX15 = ( (CSL_SYSCFG_PINMUX15_PINMUX15_11_8_GPIO0_12) << \
                         (CSL_SYSCFG_PINMUX15_PINMUX15_11_8_SHIFT)  );

    /* lock the pinmux registers                                              */
    sysRegs->KICK0R = 0x00000000;
    sysRegs->KICK1R = 0x00000000;

  /* Bring the GPIO module out of sleep state                                 */
  /* Configure the GPIO Module to Enable state */
  psc1Regs->MDCTL[GPIO_LPSC_NUM] =
                               ( (psc1Regs->MDCTL[GPIO_LPSC_NUM] & 0xFFFFFFE0) | \
                                  CSL_PSC_MDSTAT_STATE_ENABLE );
  /* Kick start the Enable Command */
  temp = psc1Regs->PTCMD;
  temp = ( (temp & CSL_PSC_PTCMD_GO0_MASK) |
           (CSL_PSC_PTCMD_GO0_SET << CSL_PSC_PTCMD_GO0_SHIFT) );
  psc1Regs->PTCMD |= temp;

  /*Wait for the power state transition to occur */
  while ( ((psc1Regs->PTSTAT & (CSL_PSC_PTSTAT_GOSTAT0_IN_TRANSITION)) != 0)
                     && (pscTimeoutCount>0) )
  {
      pscTimeoutCount--;
  }

  /* Check if PSC state transition timed out */
  if(pscTimeoutCount == 0)
  {
      printf("GPIO PSC transition to ON state timed out\n");
      return;
  }

  /* Wait for MODSTAT = ENABLE/DISABLE from LPSC */
  pscTimeoutCount = 10240u;
  while( ((psc1Regs->MDSTAT[GPIO_LPSC_NUM] & (CSL_PSC_MDSTAT_STATE_MASK))
                != CSL_PSC_MDSTAT_STATE_ENABLE) && (pscTimeoutCount>0))
  {
      pscTimeoutCount--;
  }

  /* If timeout, the resource may not be functioning */
  if (0 == pscTimeoutCount)
```

```
  {
     printf("GPIO Module Enable timed out\n");
     return;
  }

  /* Configure GPIO0_12 (GPIO0_12_PIN) as an output */
  gpioRegs->BANK[0].DIR &= ~(CSL_GPIO_DIR_DIR_IN << CSL_GPIO_DIR_DIR12_SHIFT);

  /* Configure GPIO0_8 (GPIO0_8_PIN) as an input */
  temp = gpioRegs->BANK[0].DIR;
  temp = ( (temp & CSL_GPIO_DIR_DIR8_MASK) |
                     (CSL_GPIO_DIR_DIR_IN << CSL_GPIO_DIR_DIR8_SHIFT) );
  gpioRegs->BANK[0].DIR |= temp;


  /* Set Data high in SET_DATA register for GPIO(GPIO0_12_PIN).                *
   * This turns the LED off -see schematic                                    */
  temp = gpioRegs->BANK[0].SET_DATA;
  temp = ( (temp & CSL_GPIO_SET_DATA_SET12_MASK) |
            (CSL_GPIO_SET_DATA_SET_SET << CSL_GPIO_SET_DATA_SET12_SHIFT));
  gpioRegs->BANK[0].SET_DATA |= temp;

  /* Enable GPIO Bank interrupt for bank 0                                    */
  temp = gpioRegs->BINTEN;
  temp = ( (temp & CSL_GPIO_BINTEN_EN0_MASK) |
                   (CSL_GPIO_BINTEN_EN0_ENABLE << CSL_GPIO_BINTEN_EN0_SHIFT) );
  gpioRegs->BINTEN |= temp;

  /* Configure GPIO(GPIO0_8_PIN) to generate interrupt on rising edge         */
  temp = gpioRegs->BANK[0].SET_RIS_TRIG;
  temp = ( (temp & CSL_GPIO_SET_RIS_TRIG_SETRIS8_MASK) |
            (CSL_GPIO_SET_RIS_TRIG_SETRIS_ENABLE << CSL_GPIO_SET_RIS_TRIG_SETRIS8_SHIFT)
);
  gpioRegs->BANK[0].SET_RIS_TRIG |= temp;

    /* map GPIO bank 0 event to cpu int4                                      */
    CSL_FINS(intcRegs->INTMUX1, INTC_INTMUX1_INTSEL4,GPIO0_EVENT);

    /* set ISTP to point to the vector table address                          */
    ISTP = (unsigned int)intcVectorTable;

    /* clear all interrupts, bits 4 thru 15                                   */
    ICR = 0xFFF0;

    /* enable the bits for non maskable interrupt and CPUINT4                 */
    IER = 0x12;

    /* enable interrupts, set GIE bit                                         */
    _enable_interrupts();

    printf("Waiting for GPIO Interrupt\n");

    while(0 == status)
    {
        printf("Waiting for user to configure SW3-1\n");
        delay(5000);
    }

     printf("GPIO Interrupt occured !\n");

    _disable_interrupts();
```

```
    while (ledBlinkCount++ < MAX_BLINK)
    {
        /* Make the GPIO pin (GPIO0_12_PIN) conected to the LED to low. *
         * This turns on the LED - see schematic         */
        temp = gpioRegs->BANK[0].CLR_DATA;
        temp = ( (temp & CSL_GPIO_CLR_DATA_CLR12_MASK) |
             (CSL_GPIO_CLR_DATA_CLR_CLR << CSL_GPIO_CLR_DATA_CLR12_SHIFT) );
        gpioRegs->BANK[0].CLR_DATA |= temp;

        delay(2000);

        /* Make the GPIO pin (GPIO0_12_PIN) conected to the LED to high *
         * This turns the off the LED - see schematic    */
        temp = gpioRegs->BANK[0].SET_DATA;
        temp = ( (temp & CSL_GPIO_SET_DATA_SET12_MASK) |
             (CSL_GPIO_SET_DATA_SET_SET << CSL_GPIO_SET_DATA_SET12_SHIFT) );
        gpioRegs->BANK[0].SET_DATA |= temp;


        delay(2000);
    }

    printf("End of GPIO sample application!\n");
}



void main (void)
{
   gpioExample();
}


interrupt void GPIO_input_isr()
{
    /* Let this be here now. I want to see the Heart beat :-) */
    status=1;
}


/*
 * \brief    Function to introduce a delay in to the program.
 *
 * \param    count [IN]  delay count to wait
 * \return   None
 *
 */
static void delay(Uint32 count)
{
    volatile Uint32 tempCount = 0;
    volatile Uint32 dummyCount = 0;

    for (tempCount = 0; tempCount < count; tempCount++)
    {
        for (dummyCount = 0; dummyCount < count; dummyCount++)
        {
            /* dummy loop to wait for some time  */
        }
    }
}
```

4 References

- OMAPL137 System-on-Chip (SoC) Reference Guide