```
// ****************************************************************************
//
//   Init Interrupt Request (IRQ)
//
//   Module Created 14-June-2011 by T.Lofte
//
//   Description:
//   Initialization of C6747 Interrupt Controller Module
//
//
//
//   Makes extensive use of TI Chip Support Library Register (CSLR) 3.0 functions.
//
//   14-June-2011 Created by T Lofte
//
//
// ****************************************************************************

#include <std.h>
#include <stdio.h>
#include <c6x.h>

#include "RadDefines.h"


// INTC Register Overlay:
CSL_IntcRegsOvly  pIntcRegs= (CSL_IntcRegsOvly) CSL_INTC_0_REGS ;

// HPI Register Overlay:
extern CSL_HpiRegsOvly  pHpiRegs ;

// Edma3cc Channel Controller:
extern CSL_Edma3ccRegsOvly     pEdma3cc ;

// TIMER64P0/P1 Timer Registers:
extern CSL_TmrRegsOvly      pTmrRegs[] ;

// GPIO DEBUG Pins:
extern CSL_GpioBankOvly pGpioBank[] ;

//
// Define ISR Fetch Table
extern Uint32 isrtable[] ;

// Define C6747 Core Regsisters:
extern cregister volatile Uint32 AMR ;
extern cregister volatile Uint32 CSR ;
extern cregister volatile Uint32 ICR ;
extern cregister volatile Uint32 IER ;
extern cregister volatile Uint32 IFR ;
extern cregister volatile Uint32 IRP ;
extern cregister volatile Uint32 ISR ;
extern cregister volatile Uint32 ISTP ;
extern cregister volatile Uint32 NRP ;

// Floating Point Configuration Registers
extern cregister volatile Uint32 FADCR ;
extern cregister volatile Uint32 FAUCR ;
extern cregister volatile Uint32 FMCR ;


#define ISR_INT4    0x0010
#define ISR_INT5    0x0020
#define ISR_INT6    0x0040
#define ISR_INT7    0x0080
#define ISR_INT8    0x0100

#define IER_NMI     0x0002
#define IER_INT4    0x0010
#define IER_INT5    0x0020
#define IER_INT6    0x0040
#define IER_INT7    0x0080
#define IER_INT8    0x0100

#define ICR_INT4    0x0010
#define ICR_INT5    0x0020
#define ICR_INT6    0x0040
```

```
#define ICR_INT7    0x0080
#define ICR_INT8    0x0100

#define PRDINTSTAT12 0x02


#define CSR_GIE     0x0001

Uint32  Edma3cc_Event_Flag ;
Uint32  HPI_Event_Flag ;

// DSP Debug GPIO States:
Uint32  DSP_Debug_GPIO1 ;
Uint32  DSP_Debug_GPIO2 ;
Uint32  DSP_Debug_GPIO3 ;
Uint32  DSP_Debug_GPIO4 ;

void init_IRQ (  )
{

    // ********************************************************************************
    // Disable Global Interrupts in CSR and
    // Clear IER ==> 0
    CSR &= (Uint32) ~(CSR_GIE) ;
    IER= (Uint32) 0x0 ;



    // Setup Interrupt Controller for EDMA3 Channel Controller Interrupt and HPI interrupt:

    // Clear All EventFlags in EVTFLAG0, EVTFLAG1, EVTFLAG2, EVTFLAG3 ==> NEGATE RESETVAL
    pIntcRegs->EVTCLR[0]= (Uint32) ~(CSL_INTC_EVTCLR_RESETVAL)      ;   // Clear All EventFlag0 ;
    pIntcRegs->EVTCLR[1]= (Uint32) ~(CSL_INTC_EVTCLR_RESETVAL)      ;   // Clear All EventFlag1 ;
    pIntcRegs->EVTCLR[2]= (Uint32) ~(CSL_INTC_EVTCLR_RESETVAL)      ;   // Clear All EventFlag2 ;
    pIntcRegs->EVTCLR[3]= (Uint32) ~(CSL_INTC_EVTCLR_RESETVAL)      ;   // Clear All EventFlag3 ;


    // ************************************  EVENT MASK0 ***********************************************
    // Setup Event Mask Registers 0,1,2,3:

    // EVTMASK0: OR Events Masks and NEGATE to COMBINE:
    //  EM004= TIMER64P0_TINT12  45KHZ Interrupt Signal
    //  EM008= EDMA3CC
    //
    // Clear Edma3cc_Event_Flag ;
    // Clear TIMER64P0 TINT12 Event Flag:

    Edma3cc_Event_Flag = 0 ;
    pIntcRegs->EVTMASK[0]= (Uint32) ~( CSL_INTC_EVTMASK_EM8_MASK  | CSL_INTC_EVTMASK_EM4_MASK ) ;


    // ************************** DEBUG ********************************************
    //
    // Setup Edma3cc Interrupt Registers for Completion Interrupt on Edma3 Channel #29 (MemCpy32)
    //
    // ************************** DEBUG ********************************************
    // Edma3cc Event Set #29:
    pEdma3cc->EESR= (Uint32) (CSL_EDMA3CC_EESR_E29_SET << CSL_EDMA3CC_EESR_E29_SHIFT ) ;

    // Edma3cc Interrupt Event #29:
    pEdma3cc->IESR= (Uint32) (CSL_EDMA3CC_IESR_I29_SET  <<  CSL_EDMA3CC_IESR_I29_SHIFT ) ;

    // Setup Edma3cc DMA Region #1 Access Enable Registers:
    // pEdma3cc->DRA[0].DRAE= (Uint32) (CSL_EDMA3CC_DRAE_E29_ENABLE << CSL_EDMA3CC_DRAE_E29_SHIFT  ) ;
    pEdma3cc->DRA[1].DRAE= (Uint32) (CSL_EDMA3CC_DRAE_E29_ENABLE << CSL_EDMA3CC_DRAE_E29_SHIFT  ) ;
    // pEdma3cc->DRA[2].DRAE= (Uint32) (CSL_EDMA3CC_DRAE_E29_ENABLE << CSL_EDMA3CC_DRAE_E29_SHIFT  ) ;
    // pEdma3cc->DRA[3].DRAE= (Uint32) (CSL_EDMA3CC_DRAE_E29_ENABLE << CSL_EDMA3CC_DRAE_E29_SHIFT  ) ;

    // ************************** DEBUG ********************************************


    // ************************************  EVENT MASK1 ***********************************************
    // EVTMASK1: OR Events Masks and NEGATE to COMBINE:
    //  EM034= HPI
    //  EM040= TIMER64P1_TINT12  200HZ Interrupt Signal

    HPI_Event_Flag= 0 ;
```

```
    pIntcRegs->EVTMASK[1]= (Uint32) ~( CSL_INTC_EVTMASK_EM2_MASK    |   CSL_INTC_EVTMASK_EM8_MASK  ) ;


    // ************************************  EVENT MASK2 ***********************************************
    // EVTMASK2: OR Events Masks and NEGATE to COMBINE:
    //  NONE:
    pIntcRegs->EVTMASK[2]= (Uint32)  ~( CSL_INTC_EVTMASK_RESETVAL ) ;


    // ************************************  EVENT MASK3 ***********************************************
    // EVTMASK3: OR Events Masks and NEGATE to COMBINE:
    //  NONE:
    pIntcRegs->EVTMASK[3]= (Uint32)  ~( CSL_INTC_EVTMASK_RESETVAL ) ;

    // ***************************************************************************************************
    //
    //
    // Setup Interrupt Selector Registers:
    //
    //
    // INTMUX1:
    // INTSEL04 ==> EVT0 COMBINED
    // INTSEL05 ==> EVT1 COMBINED
    // INTSE006 ==> EVT2 COMBINED
    // INTSEL07 ==> EVT3 COMBINED

    pIntcRegs->INTMUX1= (Uint32)(CSL_INTC_EVENTID_EVT0  <<  CSL_INTC_INTMUX1_INTSEL4_SHIFT  |
                                 CSL_INTC_EVENTID_EVT1  <<  CSL_INTC_INTMUX1_INTSEL5_SHIFT  |
                                 CSL_INTC_EVENTID_EVT2  <<  CSL_INTC_INTMUX1_INTSEL6_SHIFT  |
                                 CSL_INTC_EVENTID_EVT3  <<  CSL_INTC_INTMUX1_INTSEL7_SHIFT
                                 ) ;

    // INTMUX2 ==> Use RESET VALUE:
    pIntcRegs->INTMUX2= (Uint32) (CSL_INTC_INTMUX2_RESETVAL ) ;

    // INTMUX3 ==> Use RESET VALUE:
    pIntcRegs->INTMUX3= (Uint32) (CSL_INTC_INTMUX3_RESETVAL ) ;

    // ***************************************************************************************************
    //
    // Setup Interrupt Exception Status Register:
    //
    // INTXCLR ==> Interrupt Exception Clear Registers ==> Clear Interrupt Exception Status Register INTXST
    pIntcRegs->INTXCLR= (Uint32) (CSL_INTC_INTXCLR_CLEAR_YES ) ;

    // Interrupt Exception Combiner Registers ==> RESET TO POR STATE (DISABLED)
    //
    pIntcRegs->EXPMASK[0]= (Uint32) (CSL_INTC_EXPMASK_RESETVAL )    ;
    pIntcRegs->EXPMASK[1]= (Uint32) (CSL_INTC_EXPMASK_RESETVAL )    ;
    pIntcRegs->EXPMASK[2]= (Uint32) (CSL_INTC_EXPMASK_RESETVAL )    ;
    pIntcRegs->EXPMASK[3]= (Uint32) (CSL_INTC_EXPMASK_RESETVAL )    ;
    //
    //
    //
    // Setup ISR Table Pointer (ISTP) with IST Table Base Address:
    // ISR Table Must be Aligned on 256 word (1024 byte) boundary !!
    ISTP= (Uint32) isrtable ;


    // ***************************************************************************************
    // Enable NMI, INT4, INT5, INT6, INT7, INT8   for Interrupt Debugging:
    // ***************************************************************************************
    IER = (Uint32) ( IER_NMI | IER_INT4 | IER_INT5 | IER_INT6 | IER_INT7 | IER_INT8 ) ;

    // ***************************************************************************************
    // Enable Global Interrupts in CSR:
    //
    CSR |= (Uint32) (CSR_GIE) ;


    return ;
}

#pragma CODE_SECTION(nmi_isr, ".fast")

interrupt void nmi_isr( void )
{
```

```
    // printf("C6747 NMI ISR STOP\n");
    // dbprintf("C6747 NMI ISR STOP\n");
    while(1);
}

#pragma CODE_SECTION(int4_isr, ".fast")

interrupt void int4_isr( void )
{
    Uint32 AMR_temp;
    Uint32 SAT_temp ;
    Uint32 FADCR_temp;
    Uint32 FAUCR_temp ;
    Uint32 FMCR_temp ;

    // dbprintf("INT4 ISR !!!  ...\n");
    // Clear INT4 Interrupt
    ICR= (Uint32) (ICR_INT4) ;

    //************************** CORE REGISTER SAVE *********************************
    SAVE_AMR(AMR_temp) ;
    SAVE_SAT(SAT_temp) ;

    // ***************************FPU REGISTER SAVE ******************************
    //
    FADCR_temp = (Uint32) FADCR ;
    FAUCR_temp=  (Uint32) FAUCR ;
    FMCR_temp=   (Uint32) FMCR  ;

    // ********************************  Edma3cc DEBUG ********************************
    // Check Mask Event Flag Register for INT4 Combined Events :
    if (pIntcRegs->MEVTFLAG[0] & CSL_INTC_EVTMASK_EM8_MASK )
    {
        // Check Edma3cc IPR Register for Event Channel #29 (MemCpy32):
        if ( pEdma3cc->IPR  & (Uint32) (CSL_EDMA3CC_IPR_I29_YES << CSL_EDMA3CC_IPR_I29_SHIFT ) )
        {
            // Clear Event #29 in Edma3cc IPR Register ==> Write to Edma3cc ICR
            pEdma3cc->ICR= (Uint32) (CSL_EDMA3CC_ICR_I29_CLEAR  << CSL_EDMA3CC_ICR_I29_SHIFT  ) ;
            // Set Global Variable Edma3cc Event Flag:
            Edma3cc_Event_Flag= (Uint32) 1 ;
        }
        // Clear Edma3cc CC0 Int1 Flag (Event 8) in Combined Event Interrupt Controller
        pIntcRegs->EVTCLR[0]= (Uint32) (CSL_INTC_EVTMASK_EM8_MASK  ) ;
    }
    // ********************************  Edma3cc DEBUG ********************************

    if (pIntcRegs->MEVTFLAG[0] & CSL_INTC_EVTMASK_EM4_MASK)
    {

       // TIMER64P0 TINT12
        if (pTmrRegs[0]->INTCTLSTAT & (Uint32)PRDINTSTAT12 )
        {
            // TIMER64P0 TINT12 Interrupt ==> Clear Interrupt Bit in INCTLSTAT
            pTmrRegs[0]->INTCTLSTAT |= (Uint32) (PRDINTSTAT12) ;

            // Toggle GPIO[1] DEBUG Pin to Check 1HZ Interrupt rate:
            if ( DSP_Debug_GPIO1 & 0x1)
                // Turn GPIO[1] (GP1P3) ON:
                pGpioBank[GP1]->SET_DATA =  (GP1P3)  ;      // Write to GPIO SET_DATA[GP1] ==> TURN GPIO[1]
            else
                // Turn GPIO[1] (GP1P3) OFF:
                pGpioBank[GP1]->CLR_DATA =  (GP1P3)  ;      // Write to GPIO CLR_DATA[GP1] ==> TURN GPIO[1]

            DSP_Debug_GPIO1= ~(DSP_Debug_GPIO1) & 0x1 ;
        }
        // Clear TIMER64P TINT12 Event Flag :
        pIntcRegs->EVTCLR[0]= (Uint32) (CSL_INTC_EVTMASK_EM4_MASK  ) ;
    }

    //************************** CORE REGISTER RESTORE *********************************
    //
    RESTORE_AMR(AMR_temp) ;
    RESTORE_SAT(SAT_temp) ;

    // ***************************FPU REGISTER RESTORE *******************************
    //
    FADCR =  FADCR_temp ;
```

```
    FAUCR =  FAUCR_temp ;
    FMCR  =  FMCR_temp  ;



    return ;
}

#pragma CODE_SECTION(int5_isr, ".fast")

interrupt void int5_isr( void )
{
    Uint32 AMR_temp;
    Uint32 SAT_temp ;
    Uint32 FADCR_temp;
    Uint32 FAUCR_temp ;
    Uint32 FMCR_temp ;


    // dbprintf("INT5 ISR !!!  ...\n");
    // Clear INT5 Interrupt
    ICR= (Uint32) (ICR_INT5) ;

    //************************* CORE REGISTER SAVE *********************************
    SAVE_AMR(AMR_temp) ;
    SAVE_SAT(SAT_temp) ;

    // ***************************FPU REGISTER SAVE ******************************
    //
    FADCR_temp = (Uint32) FADCR ;
    FAUCR_temp=  (Uint32) FAUCR ;
    FMCR_temp=   (Uint32) FMCR  ;



    // Check Mask Event Flag Register for INT5 Combined Events :

    // Check for HPI DSP Interrupt Event !
    if (pIntcRegs->MEVTFLAG[1] & CSL_INTC_EVTMASK_EM2_MASK )
    {
        // HPI Event Flag --> Set Global HPI_Event_Flag & Clear Event
        pIntcRegs->EVTCLR[1]= (Uint32) ( CSL_INTC_EVTMASK_EM2_MASK ) ;
        HPI_Event_Flag= (Uint32) 1 ;

        // ACKnowledge HOST Interrupt by RESETTING HPI DSP INT Bit:
        pHpiRegs->HPIC |= (Uint32) ( CSL_HPI_HPIC_DSPINT_ENABLE <<  CSL_HPI_HPIC_DSPINT_SHIFT ) ;

    }

    // Check For TIMER64P1 TINT12 Period Overflow !
    if (pIntcRegs->MEVTFLAG[1] & CSL_INTC_EVTMASK_EM8_MASK )
    {
       // TIMER64P1 TINT12
        if (pTmrRegs[1]->INTCTLSTAT & (Uint32)PRDINTSTAT12 )
        {
            // TIMER64P1 TINT12 Interrupt ==> Clear Interrupt Bit in INCTLSTAT
            pTmrRegs[1]->INTCTLSTAT |= (Uint32) (PRDINTSTAT12) ;

            // Toggle GPIO[2] DEBUG Pin to Check 200HZ Interrupt rate:
            if ( DSP_Debug_GPIO2 & 0x1)
                // Turn GPIO[2] (GP1P4) ON:
                pGpioBank[GP1]->SET_DATA =  (GP1P4)  ;      // Write to GPIO SET_DATA[GP2] ==> TURN GPIO[2]
            else
                // Turn GPIO[2] (GP1P4) OFF:
                pGpioBank[GP1]->CLR_DATA =  (GP1P4)  ;      // Write to GPIO CLR_DATA[GP2] ==> TURN GPIO[2]
            // Toggle DSP_Debug_GPIO2:
            DSP_Debug_GPIO2= ~(DSP_Debug_GPIO2) & 0x1 ;
        }
        // TIMER64P1_TINT12 Period Overflow Interrupt ===> ClEAR EVENT MASK1 Combined Interrupts
        pIntcRegs->EVTCLR[1]= (Uint32) ( CSL_INTC_EVTMASK_EM8_MASK ) ;
    }

    //************************* CORE REGISTER RESTORE *********************************
    //
    RESTORE_AMR(AMR_temp) ;
    RESTORE_SAT(SAT_temp) ;
```

```
    // ****************************FPU REGISTER RESTORE *******************************
    //
    FADCR =  FADCR_temp ;
    FAUCR =  FAUCR_temp ;
    FMCR  =  FMCR_temp  ;

    return ;
}

#pragma CODE_SECTION(int6_isr, ".fast")

interrupt void int6_isr( void )
{

    Uint32 AMR_temp;
    Uint32 SAT_temp ;
    Uint32 FADCR_temp;
    Uint32 FAUCR_temp ;
    Uint32 FMCR_temp ;


    // dbprintf("INT6 ISR !!!  ...\n");
    // Clear INT6 Interrupt
    ICR= (Uint32) (ICR_INT6) ;

    //************************** CORE REGISTER SAVE *********************************
    SAVE_AMR(AMR_temp) ;
    SAVE_SAT(SAT_temp) ;

    // ****************************FPU REGISTER SAVE *******************************
    //
    FADCR_temp = (Uint32) FADCR ;
    FAUCR_temp=  (Uint32) FAUCR ;
    FMCR_temp=   (Uint32) FMCR  ;




    //************************** CORE REGISTER RESTORE *********************************
    //
    RESTORE_AMR(AMR_temp) ;
    RESTORE_SAT(SAT_temp) ;

    // ****************************FPU REGISTER RESTORE *******************************
    //
    FADCR =  FADCR_temp ;
    FAUCR =  FAUCR_temp ;
    FMCR  =  FMCR_temp  ;

    return ;
}

#pragma CODE_SECTION(int7_isr, ".fast")

interrupt void int7_isr( void )
{

    Uint32 AMR_temp;
    Uint32 SAT_temp ;
    Uint32 FADCR_temp;
    Uint32 FAUCR_temp ;
    Uint32 FMCR_temp ;


    // dbprintf("INT7 ISR !!!  ...\n");
    // Clear INT7 Interrupt
    ICR= (Uint32) (ICR_INT7) ;

    //************************** CORE REGISTER SAVE *********************************
    SAVE_AMR(AMR_temp) ;
    SAVE_SAT(SAT_temp) ;

    // ****************************FPU REGISTER SAVE *******************************
    //
    FADCR_temp = (Uint32) FADCR ;
    FAUCR_temp=  (Uint32) FAUCR ;
```

```
    FMCR_temp=   (Uint32) FMCR  ;



    //************************** CORE REGISTER RESTORE *********************************
    //
    RESTORE_AMR(AMR_temp) ;
    RESTORE_SAT(SAT_temp) ;

    // ****************************FPU REGISTER RESTORE *********************************
    //
    FADCR =  FADCR_temp ;
    FAUCR =  FAUCR_temp ;
    FMCR  =  FMCR_temp  ;



    return ;
}

#pragma CODE_SECTION(int8_isr, ".fast")

interrupt void int8_isr( void )
{
    Uint32 AMR_temp;
    Uint32 SAT_temp ;
    Uint32 FADCR_temp;
    Uint32 FAUCR_temp ;
    Uint32 FMCR_temp ;


    // dbprintf("INT8 ISR !!!  ...\n");
    // Clear INT8 Interrupt
    ICR= (Uint32) (ICR_INT8) ;


    //************************** CORE REGISTER SAVE *********************************
    SAVE_AMR(AMR_temp) ;
    SAVE_SAT(SAT_temp) ;

    // ****************************FPU REGISTER SAVE *********************************
    //
    FADCR_temp = (Uint32) FADCR ;
    FAUCR_temp=  (Uint32) FAUCR ;
    FMCR_temp=   (Uint32) FMCR  ;






    //************************** CORE REGISTER RESTORE *********************************
    //
    RESTORE_AMR(AMR_temp) ;
    RESTORE_SAT(SAT_temp) ;

    // ****************************FPU REGISTER RESTORE *********************************
    //
    FADCR =  FADCR_temp ;
    FAUCR =  FAUCR_temp ;
    FMCR  =  FMCR_temp  ;



    return ;
}
```