```
//###########################################################################
// Description:
//! \addtogroup f2803x_example_list
//! <h1>ePWM Timer Interrupt From Flash (flash_f2803x)</h1>
//!
//! This example runs the ePWM interrupt example from flash. ePwm1 Interrupt
//! will run from RAM and puts the flash into sleep mode. ePwm2 Interrupt
//! will run from RAM and puts the flash into standby mode. ePWM3 Interrupt
//! will run from FLASH. All timers have the same period. The timers are
//! started sync'ed. An interrupt is taken on a zero event for each ePWM
//! timer.GPIO34 is toggled while in the background loop.
//! Note:
//!   - ePWM1: takes an interrupt every event
//!   - ePWM2: takes an interrupt every 2nd event
//!   - ePWM3: takes an interrupt every 3rd event
//! Thus the Interrupt count for ePWM1, ePWM4-ePWM6 should be equal
//! The interrupt count for ePWM2 should be about half that of ePWM1
//! and the interrupt count for ePWM3 should be about 1/3 that of ePWM1
//!
//! Follow these steps to run the program.
//!   - Build the project
//!   - Flash the .out file into the device.
//!   - Set the hardware jumpers to boot to Flash (put position 1 and 2 of
//!     SW2 on control Card to ON position).
//!   - Use the included GEL file to load the project, symbols
//!     defined within the project and the variables into the watch
//!     window.
//!
//! Steps that were taken to convert the ePWM example from RAM
//! to Flash execution:
//! - Change the linker cmd file to reflect the flash memory map.
//! - Make sure any initialized sections are mapped to Flash.
//!   In SDFlash utility this can be checked by the View->Coff/Hex
//!   status utility. Any section marked as "load" should be
//!   allocated to Flash.
//! - Make sure there is a branch instruction from the entry to Flash
//!   at 0x3F7FF6 to the beginning of code execution. This example
//!   uses the DSP0x_CodeStartBranch.asm file to accomplish this.
//! - Set boot mode Jumpers to "boot to Flash"
//! - For best performance from the flash, modify the waitstates
//!   and enable the flash pipeline as shown in this example.
//!   Note: any code that manipulates the flash waitstate and pipeline
//!   control must be run from RAM. Thus these functions are located
//!   in their own memory section called ramfuncs.
//!
//! \b Watch \b Variables \n
//!  - EPwm1TimerIntCount
//!  - EPwm2TimerIntCount
//!  - EPwm3TimerIntCount
//
//###########################################################################
// $TI Release: F2803x C/C++ Header Files and Peripheral Examples V130 $
// $Release Date: May  8, 2015 $
// $Copyright: Copyright (C) 2009-2015 Texas Instruments Incorporated -
//             http://www.ti.com/ ALL RIGHTS RESERVED $
//###########################################################################

#include "DSP28x_Project.h"     // Device Headerfile and Examples Include File
#include <string.h>
#include <stdint.h>

//++++ Application Code ++++++
#include "Brake_Control.h"
//++++++++++++++++++++++++++++

// Configure which ePWM timer interrupts are enabled at the PIE level:
// 1 = enabled,  0 = disabled
#define PWM1_INT_ENABLE  1
#define PWM2_INT_ENABLE  1
#define PWM3_INT_ENABLE  1
```

```
// Configure the period for each timer
#define PWM1_TIMER_TBPRD   0x13FF
#define PWM2_TIMER_TBPRD   0x13FF
#define PWM3_TIMER_TBPRD   0x13FF

// Make this long enough so that we can see an LED toggle
#define DELAY 10L

// Functions that will be run from RAM need to be assigned to
// a different section.  This section will then be mapped using
// the linker cmd file.
#pragma CODE_SECTION(epwm1_timer_isr, "ramfuncs");
#pragma CODE_SECTION(epwm2_timer_isr, "ramfuncs");
#pragma CODE_SECTION(epwm3_timer_isr, "ramfuncs");
//#pragma CODE_SECTION(adc_isr, "ramfuncs");

// Prototype statements for functions found within this file.
__interrupt void epwm1_timer_isr(void);
__interrupt void epwm2_timer_isr(void);
__interrupt void epwm3_timer_isr(void);
__interrupt void adc_isr(void);

void InitEPwmTimer(void);
void Adc_Config(void);
void InitPSconnectorsGpio(void);
extern void TH1_CAN(void);

//+++++++ Application Code +++++++++
extern void Brake_Control_step(void);
extern void Brake_Control_initialize(void);
//++++++++++++++++++++++++++++++++


// Global variables used in this example


// These are defined by the linker
extern Uint16 RamfuncsLoadStart;
extern Uint16 RamfuncsLoadSize;
extern Uint16 RamfuncsRunStart;
struct ECAN_REGS ECanaShadow;

void main(void)
{

        // eCAN control registers require read/write access using 32-bits.  Thus we
        // will create a set of shadow registers for this example.  These shadow
        // registers will be used to make sure the access is 32-bits and not 16.

        struct ECAN_REGS ECanaShadow;

// Step 1. Initialize System Control:
// PLL, WatchDog, enable Peripheral Clocks
// This example function is found in the DSP2803x_SysCtrl.c file.
    InitSysCtrl();



// Step 2. Initialize GPIO:
// This example function is found in the DSP2803x_Gpio.c file and
// illustrates how to set the GPIO to it's default state.
// InitGpio();  // Skipped for this example
   InitECanGpio();


// Step 3. Clear all interrupts and initialize PIE vector table:
// Disable CPU interrupts
   DINT;

// Initialize the PIE control registers to their default state.
// The default state is all PIE interrupts disabled and flags
// are cleared.
// This function is found in the DSP2803x_PieCtrl.c file.
   InitPieCtrl();

// Disable CPU interrupts and clear all CPU interrupt flags:
   IER = 0x0000;
   IFR = 0x0000;

// Initialize the PIE vector table with pointers to the shell Interrupt
// Service Routines (ISR).
// This will populate the entire table, even if the interrupt
// is not used in this example.  This is useful for debug purposes.
// The shell ISR routines are found in DSP2803x_DefaultIsr.c.
// This function is found in DSP2803x_PieVect.c.
   InitPieVectTable();

// Interrupts that are used in this example are re-mapped to
// ISR functions found within this file.
   EALLOW;  // This is needed to write to EALLOW protected registers
   PieVectTable.EPWM1_INT = &epwm1_timer_isr;
   PieVectTable.EPWM2_INT = &epwm2_timer_isr;
   PieVectTable.EPWM3_INT = &epwm3_timer_isr;
   PieVectTable.ADCINT1 = &adc_isr;
   EDIS;    // This is needed to disable write to EALLOW protected registers

// Step 4. Initialize all the Device Peripherals:


   InitEPwmTimer();    // For this example, only initialize the ePWM Timers
   InitPSconnectorsGpio(); // Initialize GPIO/Connectors
   InitECan(); // Initialize eCAN-A module


// Step 5. User specific code, enable interrupts:

// Copy time critical code and Flash setup code to RAM
// This includes the following ISR functions: epwm1_timer_isr(), epwm2_timer_isr()
// epwm3_timer_isr and and InitFlash();
// The  RamfuncsLoadStart, RamfuncsLoadEnd, and RamfuncsRunStart
// symbols are created by the linker.
   memcpy((uint16_t *)&RamfuncsRunStart,(uint16_t *)&RamfuncsLoadStart, (unsigned
long)&RamfuncsLoadSize);

// Call Flash Initialization to setup flash waitstates
// This function must reside in RAM
   InitFlash();

// Initialize counters:
   EPwm1TimerIntCount = 0;
   EPwm2TimerIntCount = 0;
   EPwm3TimerIntCount = 0;
   LoopCount = 0;

// Enable CPU INT3 which is connected to EPWM1-3 INT:
   IER |= M_INT3;
   IER |= M_INT1; // Enable CPU Interrupt 1

// Enable EPWM INTn in the PIE: Group 3 interrupt 1-3
   PieCtrlRegs.PIEIER3.bit.INTx1 = PWM1_INT_ENABLE;
   PieCtrlRegs.PIEIER3.bit.INTx2 = PWM2_INT_ENABLE;
   PieCtrlRegs.PIEIER3.bit.INTx3 = PWM3_INT_ENABLE;
   PieCtrlRegs.PIEIER1.bit.INTx1 = 1;// Enable INT 1.1 in the PIE

// Enable global Interrupts and higher priority real-time debug events:
   EINT;   // Enable Global interrupt INTM
   ERTM;   // Enable Global realtime interrupt DBGM

// Step 6. IDLE loop. Just sit and loop forever (optional):
   //EALLOW;
  // GpioCtrlRegs.GPBMUX1.bit.GPIO34 = 0;
  //GpioCtrlRegs.GPBDIR.bit.GPIO34 = 1;
  // EDIS;

  // EALLOW;//21DEC21
  // FlashRegs.FPWR.bit.PWR = FLASH_SLEEP;
  // EDIS;

   InitAdc();


   for(;;)
   {
       // This loop will be interrupted, so the overall
       // delay between pin toggles will be longer.
       DELAY_US(DELAY);

       //GpioDataRegs.GPBTOGGLE.bit.GPIO34 = 1;

       if (Flag==1)
       {
          LoopCount++;
          Flag = 0;
        /*
        //struct ECAN_REGS ECanaShadow;
        //================================
               //ECanaMboxes.MBOX0.MDH.all = EPwm2TimerIntCount;
           // ECanaMboxes.MBOX0.MDL.all = EPwm2TimerIntCount;

             ECanaShadow.CANTRS.all= 0;
             ECanaShadow.CANTRS.bit.TRS1 = 1;             // Set TRS for mailbox under test
             ECanaRegs.CANTRS.all = ECanaShadow.CANTRS.all;
            do
             {
               ECanaShadow.CANTA.all = ECanaRegs.CANTA.all;
             } while(ECanaShadow.CANTA.bit.TA1 == 0 );   // Wait for TA5 bit to be set..

             ECanaShadow.CANTA.all = 0;
             ECanaShadow.CANTA.bit.TA1 = 1;                   // Clear TA5
             ECanaRegs.CANTA.all = ECanaShadow.CANTA.all;
             Flag = 0;

      */
       }
   }
}

void InitEPwmTimer()
{
   EALLOW;
   SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;      // Stop all the TB clocks
   EDIS;

  // InitEPwm1Gpio();
  // InitEPwm2Gpio();
  // InitEPwm3Gpio();

   // Setup Sync
   EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;  // Pass through
   EPwm2Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;  // Pass through
   EPwm3Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;  // Pass through

   // Allow each timer to be sync'ed

   EPwm1Regs.TBCTL.bit.PHSEN = TB_ENABLE;
   EPwm2Regs.TBCTL.bit.PHSEN = TB_ENABLE;
   EPwm3Regs.TBCTL.bit.PHSEN = TB_ENABLE;

   EPwm1Regs.TBPHS.half.TBPHS = 100;
   EPwm2Regs.TBPHS.half.TBPHS = 200;
   EPwm3Regs.TBPHS.half.TBPHS = 300;
   EPwm1Regs.TBPRD = PWM1_TIMER_TBPRD;
   EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;    // Count up
   EPwm1Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;     // Select INT on Zero event
   EPwm1Regs.ETSEL.bit.INTEN = PWM1_INT_ENABLE;  // Enable INT
   EPwm1Regs.ETPS.bit.INTPRD = ET_1ST;           // Generate INT on 1st event
   // Assumes ePWM1 clock is already enabled in InitSysCtrl();
      EPwm1Regs.ETSEL.bit.SOCAEN     = 1;           // Enable SOC on A group
      EPwm1Regs.ETSEL.bit.SOCASEL    = 4;           // Select SOC from from CPMA on upcount
      EPwm1Regs.ETPS.bit.SOCAPRD     = 1;           // Generate pulse on 1st event
      EPwm1Regs.CMPA.half.CMPA       = 0x0080;      // Set compare A value


   EPwm2Regs.TBPRD = PWM2_TIMER_TBPRD;
   EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;     // Count up
   EPwm2Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;      // Enable INT on Zero event
   EPwm2Regs.ETSEL.bit.INTEN = PWM2_INT_ENABLE;   // Enable INT
   EPwm2Regs.ETPS.bit.INTPRD = ET_2ND;            // Generate INT on 2nd event

   EPwm3Regs.TBPRD = PWM3_TIMER_TBPRD;
   EPwm3Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;     // Count up
   EPwm3Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;      // Enable INT on Zero event
   EPwm3Regs.ETSEL.bit.INTEN = PWM3_INT_ENABLE;   // Enable INT
   EPwm3Regs.ETPS.bit.INTPRD = ET_3RD;            // Generate INT on 3rd event

   EALLOW;
   SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;         // Start all the timers synced
   EDIS;


   // Configure ADC
        EALLOW;

        AdcRegs.ADCCTL1.bit.TEMPCONV  = 1;    //
        AdcRegs.ADCCTL1.bit.INTPULSEPOS      = 1;    //ADCINT1 trips after AdcResults latch
        AdcRegs.INTSEL1N2.bit.INT1E     = 1;  //Enabled ADCINT1
        AdcRegs.INTSEL1N2.bit.INT1CONT  = 0;  //Disable ADCINT1 Continuous mode
        AdcRegs.INTSEL1N2.bit.INT1SEL = 1;    //setup EOC1 to trigger ADCINT1 to fire

// Sw1_Current A1 J1:5,1
        AdcRegs.ADCSOC1CTL.bit.CHSEL  = 1;//set SOC1 channel select to ADCINA1
        AdcRegs.ADCSOC1CTL.bit.CHSEL  = 1;//set SOC1 channel select to ADCINA1
// Sw3_Current A2 J1:6,1
        AdcRegs.ADCSOC2CTL.bit.CHSEL    = 2;//set SOC2 channel select to ADCINA2
// Sw2_Current  A3 J1:7,1
        AdcRegs.ADCSOC3CTL.bit.CHSEL  = 3;//set SOC3 channel select to ADCINA3
// Sw4_Current A7 J1:8,1
        AdcRegs.ADCSOC4CTL.bit.CHSEL  = 7;//set SOC4 channel select to ADCINA7
// Ch1_Current A4 J11
        AdcRegs.ADCSOC5CTL.bit.CHSEL    = 4;//set SOC5 channel select to ADCINA4
// Ch1_Current A4 J11
        AdcRegs.ADCSOC6CTL.bit.CHSEL    = 6;//set SOC6 channel select to ADCINA6
// Board Temperature
        AdcRegs.ADCSOC7CTL.bit.CHSEL    = 5;//set SOC7 channel select to ADCINA5

        AdcRegs.ADCSOC0CTL.bit.TRIGSEL       = 5;    //set SOC0 start trigger on EPWM1A, due to
round-robin SOC0 converts first then SOC1
        AdcRegs.ADCSOC1CTL.bit.TRIGSEL       = 5;    //set SOC1 start trigger on EPWM1A, due to
round-robin SOC0 converts first then SOC1
        AdcRegs.ADCSOC2CTL.bit.TRIGSEL  = 5;
        AdcRegs.ADCSOC3CTL.bit.TRIGSEL  = 5;
        AdcRegs.ADCSOC4CTL.bit.TRIGSEL  = 5;
        AdcRegs.ADCSOC5CTL.bit.TRIGSEL  = 5;
        AdcRegs.ADCSOC6CTL.bit.TRIGSEL  = 5;
        AdcRegs.ADCSOC7CTL.bit.TRIGSEL  = 5;

        AdcRegs.ADCSOC0CTL.bit.ACQPS  = 6;    //set SOC0 S/H Window to 7 ADC Clock Cycles, (6
ACQPS plus 1)
        AdcRegs.ADCSOC1CTL.bit.ACQPS  = 6;    //set SOC1 S/H Window to 7 ADC Clock Cycles, (6
ACQPS plus 1)
        AdcRegs.ADCSOC2CTL.bit.ACQPS  = 6;    //set SOC1 S/H Window to 7 ADC Clock Cycles, (6
ACQPS plus 1)
```

```
        AdcRegs.ADCSOC3CTL.bit.ACQPS  = 6;
        AdcRegs.ADCSOC4CTL.bit.ACQPS  = 6;
        AdcRegs.ADCSOC5CTL.bit.ACQPS  = 6;
        AdcRegs.ADCSOC6CTL.bit.ACQPS  = 6;
        AdcRegs.ADCSOC7CTL.bit.ACQPS  = 6;
        EDIS;
}

// This ISR MUST be executed from RAM as it will put the Flash into Sleep
// Interrupt routines uses in this example:
__interrupt void epwm1_timer_isr(void)
{
   // Put the Flash to sleep
   EALLOW;
   FlashRegs.FPWR.bit.PWR = FLASH_SLEEP;
   EDIS;

   EPwm1TimerIntCount++;

   // Clear INT flag for this timer
   EPwm1Regs.ETCLR.bit.INT = 1;

   // Acknowledge this interrupt to receive more interrupts from group 3
   PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
}


// This ISR MUST be executed from RAM as it will put the Flash into Standby
__interrupt void epwm2_timer_isr(void)
{
    // Put the Flash into standby

        FlashRegs.FPWR.bit.PWR = FLASH_SLEEP;
        EDIS;
        EPwm2TimerIntCount++;


    ECanaShadow.CANRMP.all = ECanaRegs.CANRMP.all;
          if(ECanaShadow.CANRMP.bit.RMP17)
             {
                       Mode_Vehicle = ECanaMboxes.MBOX17.MDL.byte.BYTE0;
                       Enable_Vehicle = ECanaMboxes.MBOX17.MDL.byte.BYTE1;
                       x3=ECanaMboxes.MBOX17.MDL.byte.BYTE2;
                       x4=ECanaMboxes.MBOX17.MDH.byte.BYTE4;
                       x5=ECanaMboxes.MBOX17.MDH.byte.BYTE5;
                       x6=ECanaMboxes.MBOX17.MDH.byte.BYTE6;
                       x7=ECanaMboxes.MBOX17.MDH.byte.BYTE7;
             }
          ECanaShadow.CANRMP.all = ECanaRegs.CANRMP.all;
                 if(ECanaShadow.CANRMP.bit.RMP18)
                    {
                              x18_1=ECanaMboxes.MBOX18.MDL.byte.BYTE0;
                              x18_2=ECanaMboxes.MBOX18.MDL.byte.BYTE1;
                              x18_3=ECanaMboxes.MBOX18.MDL.byte.BYTE2;
                              x18_4=ECanaMboxes.MBOX18.MDH.byte.BYTE4;
                              x18_5=ECanaMboxes.MBOX18.MDH.byte.BYTE5;
                              x18_6=ECanaMboxes.MBOX18.MDH.byte.BYTE6;
                              x18_7=ECanaMboxes.MBOX18.MDH.byte.BYTE7;
                    }



        //struct ECAN_REGS ECanaShadow;
//================================

   ECanaMboxes.MBOX1.MDH.byte.BYTE4 = 0x1; //Direction Command

   Rolling_Counter = 0xf;
   Inverter_Enable = 0x0;
   Inverter_Discharge = 0x0;
   Speed_Mode_Enable = 0x1;
   ECanaMboxes.MBOX1.MDH.byte.BYTE5 = 
(Rolling_Counter<<4)+(Speed_Mode_Enable<<2)+(Inverter_Discharge<<1)+Inverter_Enable;

   //ECanaMboxes.MBOX1.MDH.byte.BYTE6 = 0x6;
   //ECanaMboxes.MBOX1.MDH.byte.BYTE7 = 0x7;
   Torque_Limit_Command = 2600; //x10
   Torque_Limit_Command_High =  Torque_Limit_Command>>8;
   Torque_Limit_Command_Low = (0x00ff & Torque_Limit_Command);
   ECanaMboxes.MBOX1.MDH.word.LOW_WORD = 
(Torque_Limit_Command_Low<<8)+Torque_Limit_Command_High; //Torque_Limit_Command

   //ECanaMboxes.MBOX1.MDL.byte.BYTE0 = 0x0;
   //ECanaMboxes.MBOX1.MDL.byte.BYTE1 = 0x1;
   Torque_Command = 2400;//x10
   Torque_Command_High = Torque_Command>>8;
   Torque_Command_Low = (0x00ff & Torque_Command);
   ECanaMboxes.MBOX1.MDL.word.HI_WORD = (Torque_Command_Low<<8)+Torque_Command_High; 
//Torque_Command

   //ECanaMboxes.MBOX1.MDL.byte.BYTE2 = 0x2;
   //ECanaMboxes.MBOX1.MDL.byte.BYTE3 = 0x3;
   Speed_Command = 543;
   Speed_Command_High = Speed_Command>>8;
   Speed_Command_Low = (0x00ff & Speed_Command);
   ECanaMboxes.MBOX1.MDL.word.LOW_WORD =  (Speed_Command_Low<<8)+Speed_Command_High; 
//Speed_Command


     ECanaShadow.CANTRS.all= 0;
     ECanaShadow.CANTRS.bit.TRS1 = 1;             // Set TRS for mailbox under test
     ECanaRegs.CANTRS.all = ECanaShadow.CANTRS.all;
    do
     {
       ECanaShadow.CANTA.all = ECanaRegs.CANTA.all;
      } while(ECanaShadow.CANTA.bit.TA1 == 0 );   // Wait for TA5 bit to be set..

     ECanaShadow.CANTA.all = 0;
     ECanaShadow.CANTA.bit.TA1 = 1;                   // Clear TA5
     ECanaRegs.CANTA.all = ECanaShadow.CANTA.all;


//J2 Input
       //Brake_Control.
       //Brake_Control_U.Sw1Cmd = GpioDataRegs.GPADAT.bit.GPIO23; // J2:5 Input GPIO
       DELAY_US(2);
       Brake_Control_U.Sw2Cmd = GpioDataRegs.GPADAT.bit.GPIO24; // J2:6 Input GPIO 
(LC1:AmmoDoor)
       DELAY_US(2);
       //Brake_Control_U.Sw3Cmd = GpioDataRegs.GPBDAT.bit.GPIO33; // J2:7 Input GPIO
       DELAY_US(2);
       Brake_Control_U.Sw4Cmd = GpioDataRegs.GPADAT.bit.GPIO10; // J12:3 Input 
GPIO(LC1:StubRemover)
       DELAY_US(2);
       //Brake_Control_U.Sw1_Current =  Sw1_Current;
       //Brake_Control_U.Sw2_Current =  Sw2_Current;
       //Brake_Control_U.Sw3_Current =  Sw3_Current;
       //Brake_Control_U.Sw4_Current =  Sw4_Current;
       //Brake_Control_U.Board_Temperature = Temperature;

         // Brake_Control_step();



//J1 Output
   //==== TEST CODE=====
        DELAY_US(2);
       GpioDataRegs.GPADAT.bit.GPIO4 = Brake_Control_U.Sw4Cmd;//GPIO10 (LC1:StubRemover)
        DELAY_US(2);
       //GpioDataRegs.GPADAT.bit.GPIO3 = Brake_Control_U.Sw1Cmd;//GPIO23
        DELAY_US(2);
       GpioDataRegs.GPADAT.bit.GPIO5 = Brake_Control_U.Sw2Cmd;//GPIO24 (LC1:AmmoDoor)
```

```
         DELAY_US(2);
        //GpioDataRegs.GPADAT.bit.GPIO3 = Brake_Control_U.Sw3Cmd;//GPIO33
         DELAY_US(2);
   //=================

         /*

        GpioDataRegs.GPADAT.bit.GPIO0 = (Brake_Control_U.Sw4Cmd|| Brake_Control_U.Sw3Cmd);//
Stub Remover Brake - J17
        //GpioDataRegs.GPACLEAR.bit.GPIO0 = Brake1_Off;//Brake_Control_U.Sw1Cmd;
         DELAY_US(2);
        //GpioDataRegs.GPASET.bit.GPIO0 = Brake_Control_Y.SW1En_Set; //Sw1En_Set;
         DELAY_US(2);
        GpioDataRegs.GPADAT.bit.GPIO1 = Brake_Test2;//
Brake_Control_U.Sw1Cmd;//Brake_Control_Y.Sw2En_Set; //Sw2En_Set;
        DELAY_US(2);
        GpioDataRegs.GPADAT.bit.GPIO2 = Brake_Test3;//  Brake_Control_U.Sw1Cmd;//Stubber motor
        //GpioDataRegs.GPASET.bit.GPIO2 = Sw3En_Set;//Brake_Control_Y.Sw3En_Set; //Sw3En_Set;
        DELAY_US(2);
         GpioDataRegs.GPADAT.bit.GPIO3 = Brake_Control_U.Sw1Cmd;// Breech Motor Brake
         DELAY_US(2);
        //GpioDataRegs.GPACLEAR.bit.GPIO0 = Brake_Control_Y.Sw1En_Clear; // Sw1En_Clear;
        //GpioDataRegs.GPACLEAR.bit.GPIO1 = Brake_Control_Y.Sw2En_Clear; //Sw2En_Clear;
        //GpioDataRegs.GPACLEAR.bit.GPIO2 = Sw3En_Clear;//Brake_Control_Y.Sw3En_Clear;
//Sw3En_Clear;
        //GpioDataRegs.GPACLEAR.bit.GPIO3 = Brake_Control_Y.Sw4En_Clear; //Sw4En_Clear;
*/
//J2 Output
         DELAY_US(200);
        GpioDataRegs.GPADAT.bit.GPIO6 =  Brake_Control_U.Sw4Cmd;//(LC1:StubRemover)
        DELAY_US(2);
        //GpioDataRegs.GPADAT.bit.GPIO7 =  Brake_Control_U.Sw4Cmd;// Stub Remover
Brake;Ack_test2;//Brake_Control_Y.Sw2Ack_Set; //Sw2Ack_Set;
        DELAY_US(2);
        GpioDataRegs.GPADAT.bit.GPIO7 =  Brake_Control_U.Sw2Cmd ;//(LC1:AmmoDoor)
        DELAY_US(2);
        //GpioDataRegs.GPADAT.bit.GPIO9 =  Ack_test4;//Brake_Control_Y.Sw4Ack_Set; //Sw4Ack_Set;

        //GpioDataRegs.GPACLEAR.bit.GPIO6 = Brake_Control_Y.Sw1Ack_Clear; //Sw1Ack_Clear;
        //GpioDataRegs.GPACLEAR.bit.GPIO7 = Brake_Control_Y.Sw2Ack_Clear; //Sw2Ack_Clear;
        //GpioDataRegs.GPACLEAR.bit.GPIO8 = Brake_Control_Y.Sw3Ack_Clear; //Sw3Ack_Clear;
        //GpioDataRegs.GPACLEAR.bit.GPIO9 = Brake_Control_Y.Sw4Ack_Clear; //Sw4Ack_Clear;

//J11 and J10
        //GpioDataRegs.GPASET.bit.GPIO4 = Fan1_On;
        DELAY_US(2);
        //GpioDataRegs.GPACLEAR.bit.GPIO4 = Fan1_Off;
        DELAY_US(2);
        //GpioDataRegs.GPASET.bit.GPIO5 = Fan2_On;
        DELAY_US(2);
        //GpioDataRegs.GPACLEAR.bit.GPIO5 = Fan2_Off;
        DELAY_US(2);


   // Clear INT flag for this timer
   EPwm2Regs.ETCLR.bit.INT = 1;

   // Acknowledge this interrupt to receive more interrupts from group 3
   PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
}

__interrupt void epwm3_timer_isr(void)
{
        // Uint16 i;

        // Put the Flash to sleep

        EALLOW;
            FlashRegs.FPWR.bit.PWR = FLASH_STANDBY;
        EDIS;
         EPwm3TimerIntCount++;
         Flag = 1;

/*
        // struct ECAN_REGS ECanaShadow;
        //==============================

              //ECanaMboxes.MBOX0.MDH.all = EPwm2TimerIntCount;
           // ECanaMboxes.MBOX0.MDL.all = EPwm2TimerIntCount;

             ECanaShadow.CANTRS.all= 0;
             ECanaShadow.CANTRS.bit.TRS0 = 1;            // Set TRS for mailbox under test
             ECanaRegs.CANTRS.all = ECanaShadow.CANTRS.all;
            do
             {
               ECanaShadow.CANTA.all = ECanaRegs.CANTA.all;
              } while(ECanaShadow.CANTA.bit.TA0 == 0 );   // Wait for TA5 bit to be set..

             ECanaShadow.CANTA.all = 0;
             ECanaShadow.CANTA.bit.TA0 = 1;                  // Clear TA5
             ECanaRegs.CANTA.all = ECanaShadow.CANTA.all;
*/
        //==============================
        /*
         Count++;

               if(Count<=500)
               {
                GpioDataRegs.GPADAT.bit.GPIO5 = 1;
               DELAY_US(2);
                GpioDataRegs.GPADAT.bit.GPIO4 = 1;
               DELAY_US(2);
               }
          if ( Count >= 1000)
                      {
                      DELAY_US(2);
                 GpioDataRegs.GPADAT.bit.GPIO5 = 0;
                 DELAY_US(2);
                 GpioDataRegs.GPADAT.bit.GPIO4 = 0;
                 DELAY_US(2);

                      }
          if ( Count >= 1500) Count=0;
*/
    // Short Delay to simulate some ISR Code
    //for(i = 1; i < 0x01FF; i++) {}

   // Clear INT flag for this timer
   EPwm3Regs.ETCLR.bit.INT = 1;

   // Acknowledge this interrupt to receive more interrupts from group 3
   PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
}


__interrupt void  adc_isr(void)
{
        // Put the Flash to sleep
    EALLOW;
        FlashRegs.FPWR.bit.PWR = FLASH_SLEEP;
        EDIS;
        ConversionCount++;

   Sw1_Current = AdcResult.ADCRESULT1;
   Sw3_Current = AdcResult.ADCRESULT2;
   Sw2_Current = AdcResult.ADCRESULT3;
   Sw4_Current = AdcResult.ADCRESULT4;
   Ch1_Current = AdcResult.ADCRESULT5;
   Ch1_Current = AdcResult.ADCRESULT6;
   Temperature = AdcResult.ADCRESULT7;



   //Convert the raw temperature sensor measurement into temperature
   degC = GetTemperatureC(Temperature);

  if( degC >= 65)
  {
         GpioDataRegs.GPADAT.bit.GPIO3 = 1;
         Flag_FanOn = 1;
  }
  if( (degC <= 55)&&(Flag_FanOn == 1))
  {
         GpioDataRegs.GPADAT.bit.GPIO3 = 0;
         Flag_FanOn = 0;
  }


   AdcRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;        //Clear ADCINT1 flag reinitialize for
next SOC
   PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;   // Acknowledge interrupt to PIE
}


//===========================================================================
// No more.
//===========================================================================
```