```
/*****************************************************************************
 Module
     PWM16Tiva.c
 Description
     Implementation file for the 16-channel version of the PWM Library for
     the Tiva
 Notes
     Channels 0-7 are implemented using the PWM Module 0 and channels
     8-15 are on PWM Module 1
 History
 When           Who     What/Why
 -------------- ---     --------
 09/05/17       jec     fixed PWM_TIVA_SetPulseWidth to fail if requested PW
                        would be greater than or equal to the period
 06/08/17       jec     Rev 2 of the 16 channel version, now inplementing 0 &
                        100% properly
 11/14/16       jec     extended to implement all 16 channels
 11/11/15       jec     converted from PWM8Tiva.c dated 11/16/14 added 2 more
                        channels and the ability to specify how many channels
                        of PWM you want
 *****************************************************************************/
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_gpio.h"
#include "inc/hw_types.h"
#include "inc/hw_pwm.h"
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/pin_map.h"
#include "driverlib/gpio.h"
#include "driverlib/pwm.h"

#include "PWM16Tiva.h"

#define MAX_NUM_CHANNELS 16


#define ChannelTo100DCMode(_ch_) (GenTo100DCConst[((_ch_) & 0x00000001)])
#define ChannelToNormDCMode(_ch_) (GenToNormDCConst[((_ch_) & 0x00000001)])

static uint32_t ulPeriod[MAX_NUM_CHANNELS>>1];
static uint8_t  LocalDuty[MAX_NUM_CHANNELS] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
static const uint32_t ChannelToPWMconst[MAX_NUM_CHANNELS]={
                                        PWM_OUT_0,PWM_OUT_1,PWM_OUT_2,PWM_OUT_3,
                                        PWM_OUT_4,PWM_OUT_5,PWM_OUT_6,PWM_OUT_7,
                                        PWM_OUT_0,PWM_OUT_1,PWM_OUT_2,PWM_OUT_3,
                                        PWM_OUT_4,PWM_OUT_5,PWM_OUT_6,PWM_OUT_7};
static const uint32_t ChannelToPWM_MOD[MAX_NUM_CHANNELS]={
                                        PWM0_BASE,PWM0_BASE,PWM0_BASE,PWM0_BASE,
                                        PWM0_BASE,PWM0_BASE,PWM0_BASE,PWM0_BASE,
                                        PWM1_BASE,PWM1_BASE,PWM1_BASE,PWM1_BASE,
                                        PWM1_BASE,PWM1_BASE,PWM1_BASE,PWM1_BASE};
static const uint32_t ChannelToGENOffset[MAX_NUM_CHANNELS]={
                            PWM_O_0_GENA,PWM_O_0_GENB,PWM_O_1_GENA,PWM_O_1_GENB,
                            PWM_O_2_GENA,PWM_O_2_GENB,PWM_O_3_GENA,PWM_O_3_GENB,
                            PWM_O_0_GENA,PWM_O_0_GENB,PWM_O_1_GENA,PWM_O_1_GENB,
                            PWM_O_2_GENA,PWM_O_2_GENB,PWM_O_3_GENA,PWM_O_3_GENB};


// constants used to set 100% DC for GenA & GenB
// to set 100% DC set the action on ZERO to output a 1
static const uint32_t GenTo100DCConst[2]= {PWM_X_GENA_ACTZERO_ONE,
                                           PWM_X_GENB_ACTZERO_ONE};
```

```
// constants used to set Normal DC for GenA & GenB
// normal is set output to 1 on UP compare, set output to 0 on DN compare
// the ZERO action is there to cover the possible 100% to 0% transition
static const uint32_t GenToNormDCConst[2]= {
 (PWM_X_GENA_ACTCMPAU_ONE | PWM_X_GENA_ACTCMPAD_ZERO | PWM_X_GENA_ACTZERO_ZERO),
 (PWM_X_GENB_ACTCMPBU_ONE | PWM_X_GENB_ACTCMPBD_ZERO |
PWM_X_GENA_ACTZERO_ZERO)};


static const uint32_t Group2GENconst[MAX_NUM_CHANNELS>>1]={
                                    PWM_GEN_0,PWM_GEN_1,PWM_GEN_2,PWM_GEN_3,
                                    PWM_GEN_0,PWM_GEN_1,PWM_GEN_2,PWM_GEN_3};


static int8_t MaxConfiguredChannel = -1; // init to illegal value


bool PWM_TIVA_Init(uint8_t HowMany){
 static   uint8_t i;

//sanity check
    if ((HowMany == 0) || (HowMany >MAX_NUM_CHANNELS)){
      return false;
  }
  MaxConfiguredChannel = HowMany-1; // note how many we have configured

  //Configure PWM Clock to system / 32
  SysCtlPWMClockSet(SYSCTL_PWMDIV_32);

  // Calculate the period for 500 Hz including divide by 32 in PWM clock
  ulPeriod[0] = SysCtlClockGet()/32 / 500; //PWM frequency 500HZ
  // set all of the periods the same initially
  for (i=1; i<(sizeof(Group2GENconst)/sizeof(Group2GENconst[0])); i++){
    ulPeriod[i] = ulPeriod[0];
  }
  // Enable the minimal ports & PWM module.
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
  while (SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOB) != true)
    ;
  SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0);
  while (SysCtlPeripheralReady(SYSCTL_PERIPH_PWM0) != true)
    ;
  // Enable the ports for channels 4 & 5 if needed
  if (HowMany > 4){
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
    while (SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOE) != true)
      ;
  }
  // Enable the ports for channels 6 & 7 if needed
  if (HowMany > 6){
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
    while (SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOC) != true)
      ;
  }
  // Enable the ports & PWM Module for channels 8 & 9 if needed
  if (HowMany > 8){
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
    while (SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOD) != true)
      ;
    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);
    while (SysCtlPeripheralReady(SYSCTL_PERIPH_PWM1) != true)
      ;
  }
  // Enable the ports & PWM Module for channels 10 & 11 if needed
```

```
  if (HowMany > 10){
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    while (SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOA) != true)
      ;
  }
  // Enable the ports for channels 10 & 11 if needed
  if (HowMany > 10){
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    while (SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOA) != true)
      ;
  }
  // Enable the ports for channels 12 - 16 (M1PWM4-7) if needed
  if (HowMany > 12){
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    while (SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOF) != true)
      ;
  }
    //Configure PWM Options for the generators (1 generator for 2 channels)
    for (i=0; i<HowMany; i+=2){
      PWMGenConfigure(ChannelToPWM_MOD[i], Group2GENconst[i>>1],
                      PWM_GEN_MODE_UP_DOWN | PWM_GEN_MODE_NO_SYNC);
    //Set the Period (expressed in clock ticks)
      PWMGenPeriodSet(ChannelToPWM_MOD[i], Group2GENconst[i>>1],
ulPeriod[i>>1]);
    }

    //Set PWM duty to initial value
    for ( i = 0; i<HowMany; i++){
      PWM_TIVA_SetDuty( LocalDuty[i], i);
    }

    // Enable the PWM generators
    for (i=0; i<HowMany; i+=2){
      PWMGenEnable(ChannelToPWM_MOD[i], Group2GENconst[i>>1]);
    }
    // set synchonous update
    PWMSyncUpdate(PWM0_BASE, PWM_GEN_0_BIT | PWM_GEN_1_BIT | PWM_GEN_2_BIT |
                  PWM_GEN_3_BIT);
    if (HowMany >8){ // only mess with PWM module 1 if we need to
      PWMSyncUpdate(PWM1_BASE, PWM_GEN_0_BIT | PWM_GEN_1_BIT | PWM_GEN_2_BIT |
                  PWM_GEN_3_BIT);
    }
    // Turn on the Output pins as required

    switch ( HowMany ){
    case 16: // intentional fall throughs to the lower channels
      GPIOPinConfigure(GPIO_PF3_M1PWM7);
      GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_3);
      PWMOutputState(PWM1_BASE, PWM_OUT_7_BIT, true);
    case 15: // intentional fall throughs to the lower channels
      GPIOPinConfigure(GPIO_PF2_M1PWM6);
      GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_2);
      PWMOutputState(PWM1_BASE, PWM_OUT_6_BIT, true);
    case 14: // intentional fall throughs to the lower channels
      GPIOPinConfigure(GPIO_PF1_M1PWM5);
      GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_1);
      PWMOutputState(PWM1_BASE, PWM_OUT_5_BIT, true);
    case 13: // intentional fall throughs to the lower channels
      // PF0 requires special treatment because of lock on NMI function
      // First open the lock and select the bits we want to modify in the
      // GPIO commit register, then modify the pin type
      HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
      HWREG(GPIO_PORTF_BASE + GPIO_O_CR) = 0x1;
      GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_0);
```

```
      GPIOPinConfigure(GPIO_PF0_M1PWM4);
      PWMOutputState(PWM1_BASE, PWM_OUT_4_BIT, true);
    case 12: // intentional fall throughs to the lower channels
      GPIOPinConfigure(GPIO_PA7_M1PWM3);
      GPIOPinTypePWM(GPIO_PORTA_BASE, GPIO_PIN_7);
      PWMOutputState(PWM1_BASE, PWM_OUT_3_BIT, true);
    case 11:
      GPIOPinConfigure(GPIO_PA6_M1PWM2);
      GPIOPinTypePWM(GPIO_PORTA_BASE, GPIO_PIN_6);
      PWMOutputState(PWM1_BASE, PWM_OUT_2_BIT, true);
    case 10: // intentional fall throughs to the lower channels
      GPIOPinConfigure(GPIO_PD1_M1PWM1);
      GPIOPinTypePWM(GPIO_PORTD_BASE, GPIO_PIN_1);
      PWMOutputState(PWM1_BASE, PWM_OUT_1_BIT, true);
    case 9:
      GPIOPinConfigure(GPIO_PD0_M1PWM0);
      GPIOPinTypePWM(GPIO_PORTD_BASE, GPIO_PIN_0);
      PWMOutputState(PWM1_BASE, PWM_OUT_0_BIT, true);
    case 8:
      GPIOPinConfigure(GPIO_PC5_M0PWM7);
      GPIOPinTypePWM(GPIO_PORTC_BASE, GPIO_PIN_5);
      PWMOutputState(PWM0_BASE, PWM_OUT_7_BIT, true);
    case 7:
      GPIOPinConfigure(GPIO_PC4_M0PWM6);
      GPIOPinTypePWM(GPIO_PORTC_BASE, GPIO_PIN_4);
      PWMOutputState(PWM0_BASE, PWM_OUT_6_BIT, true);
    case 6:
      GPIOPinConfigure(GPIO_PE5_M0PWM5);
      GPIOPinTypePWM(GPIO_PORTE_BASE, GPIO_PIN_5);
      PWMOutputState(PWM0_BASE, PWM_OUT_5_BIT, true);
    case 5:
      GPIOPinConfigure(GPIO_PE4_M0PWM4);
      GPIOPinTypePWM(GPIO_PORTE_BASE, GPIO_PIN_4);
      PWMOutputState(PWM0_BASE, PWM_OUT_4_BIT, true);
    case 4:
      GPIOPinConfigure(GPIO_PB5_M0PWM3);
      GPIOPinTypePWM(GPIO_PORTB_BASE, GPIO_PIN_5);
      PWMOutputState(PWM0_BASE, PWM_OUT_3_BIT, true);
    case 3:
      GPIOPinConfigure(GPIO_PB4_M0PWM2);
      GPIOPinTypePWM(GPIO_PORTB_BASE, GPIO_PIN_4);
      PWMOutputState(PWM0_BASE, PWM_OUT_2_BIT, true);
    case 2:
      GPIOPinConfigure(GPIO_PB7_M0PWM1);
      GPIOPinTypePWM(GPIO_PORTB_BASE, GPIO_PIN_7);
      PWMOutputState(PWM0_BASE, PWM_OUT_1_BIT, true);
    case 1:
      GPIOPinConfigure(GPIO_PB6_M0PWM0);
      GPIOPinTypePWM(GPIO_PORTB_BASE, GPIO_PIN_6);
      PWMOutputState(PWM0_BASE, PWM_OUT_0_BIT, true);
  }
  return true;
}

bool PWM_TIVA_SetDuty( uint8_t dutyCycle, uint8_t channel)
{
  uint32_t updateVal;

  if (channel > MaxConfiguredChannel)// sanity check, reasonable channel number
    return false;
  if (dutyCycle > 100)               // sanity check, reasonable DC
    return false;
  // update local copy of DC used when changing freq or period
  LocalDuty[channel] = dutyCycle;
```

```
  if (0 == dutyCycle){ // don't try to calculate with 0 DC
    updateVal = 0;
  }else{ // reasonable duty cycle number, so calculate new pulse width
    updateVal = (ulPeriod[channel>>1]*dutyCycle)/100;
  }
  // 100% DC needs to be handled differently to work with the PWM hardware
  if (100 == dutyCycle)
  {
  // To program 100% DC, simply set the action on Zero to set the output to ONE
    HWREG( ChannelToPWM_MOD[channel]+ChannelToGENOffset[channel] ) =
                                      ChannelTo100DCMode(channel);

  }else{
    // if not 100%, then program normal DC actions
    HWREG( ChannelToPWM_MOD[channel]+ChannelToGENOffset[channel] ) =
                                      ChannelToNormDCMode(channel);
    // and set the new pulse width based on requested DC
    PWMPulseWidthSet(ChannelToPWM_MOD[channel],
                                        ChannelToPWMconst[channel],updateVal);
  }
  return true;
}

bool PWM_TIVA_SetPulseWidth( uint16_t NewPW, uint8_t channel)
{
  if (channel > MaxConfiguredChannel) // sanity check, reasonable channel number
    return false;
  // make sure that the requested PW is less than the period before updating
  if ( NewPW < ulPeriod[channel>>1]){
    PWMPulseWidthSet(ChannelToPWM MOD[channel],
ChannelToPWMconst[channel],NewPW);
    return true;
  }else{
    return false;
  }
}

/*****************************************************************************
  PWM_TIVA_SetPeriod( uint16_t reqPeriod, uint8_t group)
    sets the requested PWM group's period to the Requested Period
    group 0 = channels 0 & 1
    group 1 = channels 2 & 3
    group 2 = channels 4 & 5
    group 3 = channels 6 & 7
    group 4 = channels 8 & 9
    group 5 = channels 10 & 11
    group 6 = channels 12 & 13
    group 7 = channels 14 & 15
*****************************************************************************/

bool PWM_TIVA_SetPeriod( uint16_t reqPeriod, uint8_t group)
{
  if(group > (MaxConfiguredChannel>>1)){  // sanity check
    return false;
  }
  //Set the Period (expressed in clock ticks)
  ulPeriod[group] = reqPeriod;
  PWMGenPeriodSet(ChannelToPWM_MOD[group<<1], Group2GENconst[group], reqPeriod);
  // Set new Duty after period change
  PWM_TIVA_SetDuty( LocalDuty[group<<1], group<<1);
  PWM_TIVA_SetDuty( LocalDuty[(group<<1)+1], (group<<1)+1);
  return true;
}
```

```
 /*****************************************************************************
  PWM_TIVA_SetFreq( uint16_t reqPeriod, uint8_t group)
    sets the requested PWM group's frequency to the Requested Frequency, in Hz
    group 0 = channels 0 & 1
    group 1 = channels 2 & 3
    group 2 = channels 4 & 5
    group 3 = channels 6 & 7
    group 4 = channels 8 & 9
    group 5 = channels 10 & 11
    group 6 = channels 12 & 13
    group 7 = channels 14 & 15
*****************************************************************************/

bool PWM_TIVA_SetFreq( uint16_t reqFreq, uint8_t group)
{
  if(group > (MaxConfiguredChannel>>1)){ // sanity check
    return false;
  }
  //Use the Frequency (expressed in Hz) to calculate a new period
  ulPeriod[group] = SysCtlClockGet()/32 /reqFreq;
  // apply the new period
  PWM_TIVA_SetPeriod( ulPeriod[group], group);
  return true;
}
```