```
/*
 * Copyright 2016-2018 NXP
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or
without modification,
 * are permitted provided that the following conditions are met:
 *
 * o Redistributions of source code must retain the above copyright
notice, this list
 *   of conditions and the following disclaimer.
 *
 * o Redistributions in binary form must reproduce the above
copyright notice, this
 *   list of conditions and the following disclaimer in the
documentation and/or
 *   other materials provided with the distribution.
 *
 * o Neither the name of NXP Semiconductor, Inc. nor the names of
its
 *   contributors may be used to endorse or promote products derived
from this
 *   software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
CONTRIBUTORS BE LIABLE FOR
 * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON
 * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/**
 * @file    LPC55S69_blinky.c
 * @brief   Application entry point.
 */
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
```

```
#include "LPC55S69_cm33_core0.h"
#include "fsl_debug_console.h"
/* TODO: insert other include files here. */
#include "fsl_mrt.h"
#include "fsl_pint.h"
#include "fsl_inputmux.h"
#include "fsl_ctimer.h"

/* TODO: insert other definitions and declarations here. */
#define MRT_CLK_FREQ CLOCK_GetFreq(kCLOCK_BusClk)
#define APP_LED_INIT LED_GREEN_INIT(LOGIC_LED_OFF);
#define APP_LED_ON (LED_GREEN_ON());
#define APP_LED_TOGGLE (LED_GREEN_TOGGLE());
#define BLINKY_PINT_PIN_INT0_USER kINPUTMUX_GpioPort1Pin9ToPintsel
#define CTIMER CTIMER2                  /* Timer 2 */
#define CTIMER_MAT_OUT kCTIMER_Match_2 /* Match output 1 */
#define CTIMER_CLK_FREQ CLOCK_GetFreq(kCLOCK_CTmier2)

#define COUNT_DOWN                0
#define COUNT_UP                  1
#define COUNT_PAUSE      2

uint32_t maskLED = 0;
volatile uint32_t g_pwmPeriod = 0U;
volatile uint32_t g_pulsePeriod = 0U;
uint32_t g_timerClock;

/
**************************************************************************
***********
 * Code

**************************************************************************
**********/
status_t CTIMER_GetPwmPeriodValue(uint32_t pwmFreqHz, uint8_t
dutyCyclePercent, uint32_t timerClock_Hz)
{
    /* Calculate PWM period match value */
    g_pwmPeriod = (timerClock_Hz / pwmFreqHz) - 1;

    /* Calculate pulse width match value */
    if (dutyCyclePercent == 0)
    {
        g_pulsePeriod = g_pwmPeriod + 1;
    }
    else
    {
        g_pulsePeriod = (g_pwmPeriod * (100 - dutyCyclePercent)) /
100;
    }
    return kStatus_Success;
}

void MRT0_IRQHandler(void)
```

```
{
        /* Will get here every 10ms */
        static uint32_t brightcount = 0;
        static uint32_t dir_count = COUNT_UP;
    /* Clear interrupt flag.*/
    MRT_ClearStatusFlags(MRT0, kMRT_Channel_0,
kMRT_TimerInterruptFlag);

    /* Update the PWM period in range 1%-99% every 10ms in sawtooth
brightness */
    if (dir_count==COUNT_UP) {
        brightcount++;
        if (brightcount == 100) {
                        dir_count = COUNT_DOWN;
        }
    }
    if (dir_count==COUNT_DOWN) {
        brightcount--;
        if (brightcount == 0) {
                dir_count = COUNT_UP;
        }
    }

    if (maskLED) {
        dir_count = COUNT_UP;
        brightcount = 99;
    }

    CTIMER_GetPwmPeriodValue(20000, brightcount, g_timerClock);
    CTIMER_SetupPwmPeriod(CTIMER, CTIMER_MAT_OUT, g_pwmPeriod,
g_pulsePeriod, false);

}


/*!
* @brief Call back for PINT Pin interrupt 0-7.
*/
void pint_intr_callback(pint_pin_int_t pintr, uint32_t
pmatch_status)
{
    PRINTF("\r\nPINT Pin Interrupt %d event detected.", pintr);
    maskLED ^= 1UL;
}


/*
 * @brief   Application entry point.
 */
int main(void) {

    /* Structure of initialize MRT */
    mrt_config_t mrtConfig;
    uint32_t mrt_clock;
```

```
    ctimer_config_t config;
    uint32_t srcClock_Hz;



        /* Init board hardware. */
    BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitBootPeripherals();
        /* Init FSL debug console. */
    BOARD_InitDebugConsole();

    PRINTF("\n\rNXP Blinky");

    /* enable clock for GPIO; used to toggle the LED's */
//    CLOCK_EnableClock(kCLOCK_Gpio1);


    /* Connect trigger sources to PINT */
    INPUTMUX_Init(INPUTMUX);
    INPUTMUX_AttachSignal(INPUTMUX, kPINT_PinInt0,
BLINKY_PINT_PIN_INT0_USER);
    /* Turnoff clock to inputmux to save power. Clock is only needed
to make changes */
    INPUTMUX_Deinit(INPUTMUX);
   /* Initialize PINT */
    PINT_Init(PINT);
    /* Setup Pin Interrupt 0 for rising edge */
    PINT_PinInterruptConfig(PINT, kPINT_PinInt0,
kPINT_PinIntEnableRiseEdge, pint_intr_callback);
    /* Enable callbacks for PINT0 by Index */
    PINT_EnableCallbackByIndex(PINT, kPINT_PinInt0);



    mrt_clock = MRT_CLK_FREQ;
    /* mrtConfig.enableMultiTask = false; */
    MRT_GetDefaultConfig(&mrtConfig);
    /* Init mrt module */
    MRT_Init(MRT0, &mrtConfig);
    /* Setup Channel 0 to be repeated */
    MRT_SetupChannelMode(MRT0, kMRT_Channel_0, kMRT_RepeatMode);
    /* Enable timer interrupts for channel 0 */
    MRT_EnableInterrupts(MRT0, kMRT_Channel_0,
kMRT_TimerInterruptEnable);
    /* Enable at the NVIC */
    EnableIRQ(MRT0_IRQn);
    /* Start channel 0 */
    PRINTF("\r\nStarting mrt timer0, channel No.0 for 10ms tick");
    MRT_StartTimer(MRT0, kMRT_Channel_0, USEC_TO_COUNT(10000U,
mrt_clock));


    /* CTimer0 counter uses the AHB clock, some CTimer1 modules use
```

```
the Aysnc clock */
    PRINTF("\r\nCTimer initialisation for PWM");
    srcClock_Hz = CTIMER_CLK_FREQ;
    CTIMER_GetDefaultConfig(&config);
    g_timerClock = srcClock_Hz / (config.prescale + 1);
    CTIMER_Init(CTIMER, &config);
    /* Get the PWM period match value and pulse width match value of
20Khz PWM signal with 99% dutycycle */
    CTIMER_GetPwmPeriodValue(20000, 1, g_timerClock);
    CTIMER_SetupPwmPeriod(CTIMER, CTIMER_MAT_OUT, g_pwmPeriod,
g_pulsePeriod, false);
    CTIMER_StartTimer(CTIMER);


    while(1)     {
    }
    return 0 ;
}
```