DMA triggering function Demo for LPC55xx From CTimer

1.Introduction

The document describes how to use DMA triggering mode to transfer data between memory and peripheral. In detail, the CTimer0 is configured to generate matching event with programmable period, the CTimer0 matching event triggers DMA, the DMA transfer data between a variable in memory and GPIO Toggling register(the GPIO is connected to a LED), so user can see the LED toggling. The DMA Ping-Pong modes are used.

2 DMA description

The DMA is an important module, it can transfer data from one location to another location without Core involved, in other words, it relieves the core workload, increases the efficiency. The LPC55xx has on-chip DMA module, which can access the peripheral and memory as master. It supports the transfer types: peripheral to memory, memory to peripheral, memory to memory.
The LPC55xx has two independent DMA modules, DMA0 and DMA1
The unique feature of the DMA is the request mode and triggering mode, as the following figure.

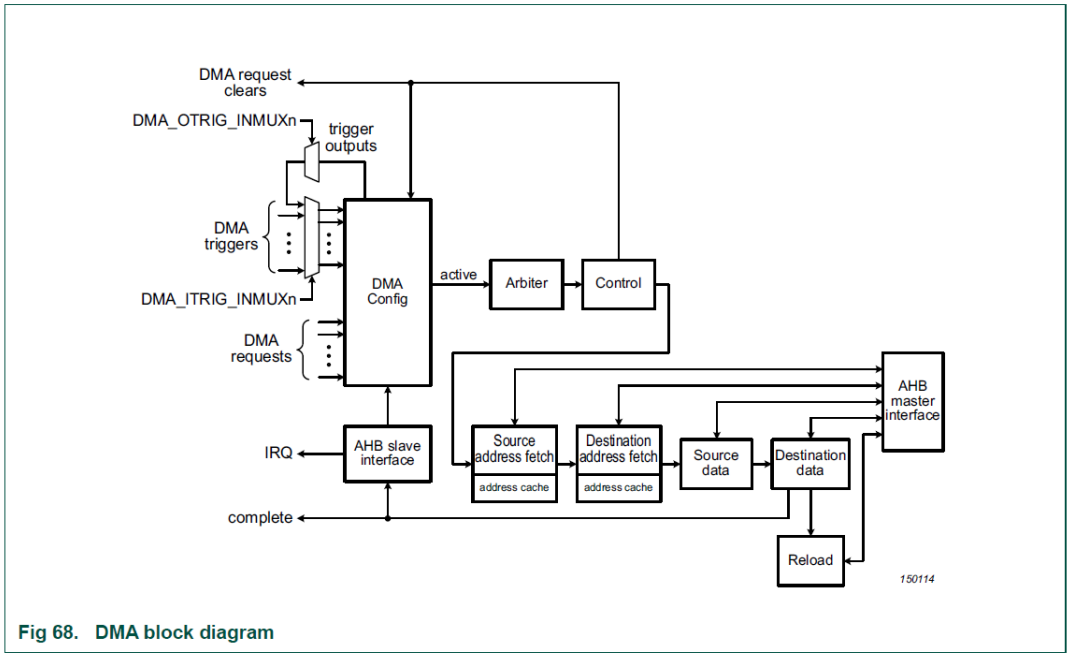


Fig 68. DMA block diagram

The request sources corresponds to fixed DMA channel as the following Fig:
For example, FlexComm0 receiver corresponds to DMA0 channel 4, the FlexComm0 transmitter corresponds to DMA0 channel 5, the relationship between request source and

DMA channel can not be changed. The DMA request mechanism can pace the transfer automatically.
For memory to memory transfer, because there is not request source, any DMA channel can be used with software trigger.

Another DMA mode is triggering mode, which is different with request mode, in general, the request mode can pace the transfer rate, for example, if you use the FlexComm0 module as uart, and you can use DMA0 channel 5 to transfer data between memory to uart transmitter data register, the request mode can pace the uart transfer.
In triggering mode, the triggering sources do not correspond to the DMA channel, in other words, the trigger source can use any DMA channel.

Table 407. DMA0 requests and trigger multiplexers

DMA channel	Request input	DMA trigger mux
0	Hash-Crypt DMA request	DMA0_ITRIG_INMUX0
1	Spare channel, no request connected	DMA0_ITRIG_INMUX1
2	High Speed SPI (Flexcomm 8) RX	DMA0_ITRIG_INMUX2
3	High Speed SPI (Flexcomm 8) TX	DMA0_ITRIG_INMUX3
4	Flexcomm Interface 0 RX / I2C Slave [1]	DMA0_ITRIG_INMUX4
5	Flexcomm Interface 0 TX / I2C Master [1]	DMA0_ITRIG_INMUX5
6	Flexcomm Interface 1 RX / I2C Slave [1]	DMA0_ITRIG_INMUX6
7	Flexcomm Interface 1 TX / I2C Master [1]	DMA0_ITRIG_INMUX7
8	Flexcomm Interface 3 RX / I2C Slave [1]	DMA0_ITRIG_INMUX8
9	Flexcomm Interface 3 TX / I2C Master [1]	DMA0_ITRIG_INMUX9
10	Flexcomm Interface 2 RX / I2C Slave [1]	DMA0_ITRIG_INMUX10
11	Flexcomm Interface 2 TX / I2C Master [1]	DMA0_ITRIG_INMUX11
12	Flexcomm Interface 4 RX / I2C Slave [1]	DMA0_ITRIG_INMUX12
13	Flexcomm Interface 4 TX / I2C Master [1]	DMA0_ITRIG_INMUX13
14	Flexcomm Interface 5 RX / I2C Slave [1]	DMA0_ITRIG_INMUX14
15	Flexcomm Interface 5 TX / I2C Master [1]	DMA0_ITRIG_INMUX15
16	Flexcomm Interface 6 RX / I2C Slave [1]	DMA0_ITRIG_INMUX16

This is the trigger sources for LPC556x family.

Table 410. DMA trigger sources

DMA trigger	DMA0 trigger input	DMA1 trigger input
0	Pin interrupt 0	Pin interrupt 0
1	Pin interrupt 1	Pin interrupt 1
2	Pin interrupt 2	Pin interrupt 2
3	Pin interrupt 3	Pin interrupt 3
4	Timer CTIMER0 Match 0	Timer CTIMER0 Match 0
5	Timer CTIMER0 Match 1	Timer CTIMER0 Match 1
6	Timer CTIMER1 Match 0	Timer CTIMER2 Match 0

3 DMA triggering source

The INPUTMUX module set up the DMA channel and DMA triggering source, this is the registers. Each **DMA0_ITRIG_INMUX[0:22] register** corresponds a DMA channel, for example DMA0_ITRG_INMUX[0] corresponds DMA0 channel0, DMA0_ITRG_INMUX[1] corresponds DMA0 channel1. If you set DMA0_ITRG_INMUX[0]=4, the CTimer0 Match0 event can trigger DMA0 channel 0 to transfer data.

Table 355. DMA0 trigger Input multiplexing registers (DMA0_ITRIG_INMUX[0:22], offsets [0x0E0:0x138])

Bit	Symbol	Value	Description	Reset value
4:0	INP		Trigger input number (decimal value) for DMA channel n (n = 0 to 22).	0x1F
		0	Pin interrupt 0.	
		1	Pin interrupt 1.	
		2	Pin interrupt 2.	
		3	Pin interrupt 3.	
		4	Timer CTIMER0 Match 0.	
		5	Timer CTIMER0 Match 1.	
		6	Timer CTIMER1 Match 0.	
		7	Timer CTIMER1 Match 1.	
		8	Timer CTIMER2 Match 0.	
		9	Timer CTIMER2 Match 1.	
		10	Timer CTIMER3 Match 0.	
		11	Timer CTIMER3 Match 1.	
		12	Timer CTIMER4 Match 0.	
		13	Timer CTIMER4 Match 1.	
		14	Comparator 0 output.	
		15	DMA output trigger 0.	
		16	DMA output trigger 1.	
		17	DMA output trigger 2.	
		18	DMA output trigger 3.	
		19	SCT0 DMA request 0.	
		20	SCT0 DMA request 1.	
		21	Hash-Crypt output DMA.	
31:5	-	-	Reserved.	-

4 features of the project

The CTimer0 Match0 register is set up so that the CTimer0_Match0 can generate event. The CTimer0 generates event by Match0 with 0.5S period, the event triggers DMA via INPUTMUX module. For each CTimer0_match0 triggering, the DMA will fetch data from memory and send to GPIO Toggle data register, both the memory and the GPIO Toggle data register remain their original address without any update. After ten times data transferring, the current DMA will terminate, the DMA interrupt is triggered, the DMA **`DMA_Callback()`** `function is called. The project uses DMA Ping-Pong mode, when the PING DMA terminates, because the XFERCFG0[RELOAD] bit is set, the PONG descriptor will reload the DMA, another DMA transfer will launch once the CTimer0_match0 event happens.`

5 Result

After the application code is run, user can see that the RED LED toggles 10 times and BLUE LED toggles 10 times, then the RED LED toggles 10 times and BLUE LED toggles 10 times, the procedure repeats 10 times, then stop.

6 appendix

This is the CTimer0 code.

```
#define CTIMER_CLK_FREQ CLOCK_GetFreq(kCLOCK_FroHf)
void CtimerInit(void)
{
    ctimer_config_t config;
    ctimer_match_config_t matchConfig;

    CLOCK_AttachClk(kFRO_HF_to_CTIMER0);
    CTIMER_GetDefaultConfig(&config);
    CTIMER_Init(CTIMER0, &config);

    matchConfig.enableCounterReset = true;
    matchConfig.enableCounterStop  = false;
    matchConfig.matchValue         = CTIMER_CLK_FREQ /2;
    matchConfig.outControl         = kCTIMER_Output_Toggle;
    matchConfig.outPinInitState    = true;
    matchConfig.enableInterrupt    = true;
    CTIMER_SetupMatch(CTIMER0, kCTIMER_Match_0, &matchConfig);

    matchConfig.enableCounterReset = false;
    matchConfig.enableCounterStop  = false;
    matchConfig.matchValue         = (CTIMER_CLK_FREQ /2)-100;
    matchConfig.outControl         = kCTIMER_Output_Toggle;
    matchConfig.outPinInitState    = true;
    matchConfig.enableInterrupt    = false;
    CTIMER_SetupMatch(CTIMER0, kCTIMER_Match_1, &matchConfig);
    CTIMER_RegisterCallBack(CTIMER0, &ctimer_callback_table[0],
kCTIMER_MultipleCallback);

}

void CTimerStart(void)
```

```
{
    CTIMER_StartTimer(CTIMER0);
   //__NVIC_DisableIRQ(CTIMER0_IRQn);
}
```

Note: There is not dedicated bit to enable the CTimer matching event to trigger DMA, user has to enable the CTimer corresponding match interrupt, and enable the corresponding bit in NVIC->ISER[] bit, if the match event is used to trigger DMA channel, the interrupt is not triggered instead.
Because only CTimer0 match0 and match1 events can trigger DMA, if the match0 event is used to trigger DMA,the MR0I bit must be set, if the match1 event is used to trigger DMA,the MR1I bit must be set.

Table 538. Match Control Register (MCR, offset 0x014)

Bit	Symbol	Description	Reset Value
0	MR0I	Interrupt on MR0: an interrupt is generated when MR0 matches the value in the TC. 0 = disabled. 1 = enabled.	0
1	MR0R	Reset on MR0: the TC will be reset if MR0 matches it. 0 = disabled. 1 = enabled.	0
2	MR0S	Stop on MR0: the TC and PC will be stopped and TCR[0] will be set to 0 if MR0 matches the TC. 0 = disabled. 1 = enabled.	0
3	MR1I	Interrupt on MR1: an interrupt is generated when MR1 matches the value in the TC. 0 = disabled. 1 = enabled. 0 = disabled. 1 = enabled.	0
4	MR1R	Reset on MR1: the TC will be reset if MR1 matches it. 0 = disabled. 1 = enabled.	0
5	MR1S	Stop on MR1: the TC and PC will be stopped and TCR[0] will be set to 0 if MR1 matches the TC. 0 = disabled. 1 = enabled.	0
6	MR2I	Interrupt on MR2: an interrupt is generated when MR2 matches the value in the TC. 0 = disabled. 1 = enabled.	0
7	MR2R	Reset on MR2: the TC will be reset if MR2 matches it. 0 = disabled. 1 = enabled.	0
8	MR2S	Stop on MR2: the TC and PC will be stopped and TCR[0] will be set to 0 if MR2 matches the TC. 0 = disabled. 1 = enabled.	0
9	MR3I	Interrupt on MR3: an interrupt is generated when MR3 matches the value in the TC.	0

As the following figure, the ISE_CTIMER0 bit in the NVIC->ISER0 must be set if the CTimer0 match event is ued to trigger DMA.

Table 10. Interrupt set-enable register 0

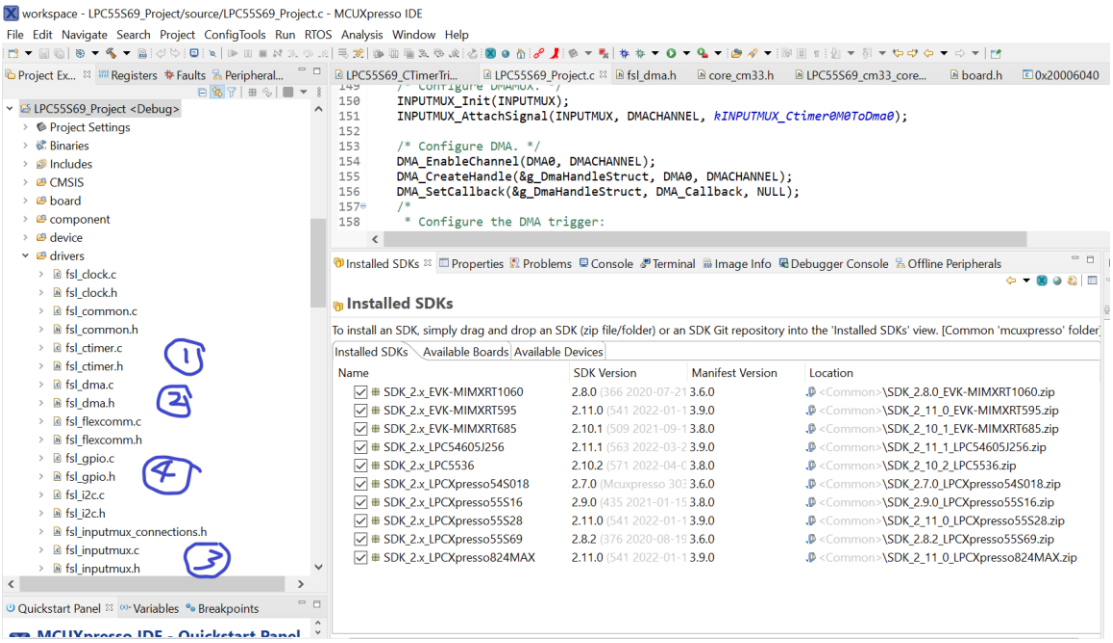
Bit	Name	Value	Function
0	ISE_WDTBOD	[1]	Watchdog Timer, BOD interrupt enable.
1	ISE_SDMA0	[1]	SDMA0 interrupt enable.
2	ISE_GINT0	[1]	GPIO group 0 interrupt enable.
3	ISE_GINT1	[1]	GPIO group 1 interrupt enable.
4	ISE_PINT0	[1]	Pin interrupt / pattern match engine slice 0 interrupt enable.
5	ISE_PINT1	[1]	Pin interrupt / pattern match engine slice 1 interrupt enable.
6	ISE_PINT2	[1]	Pin interrupt / pattern match engine slice 2 interrupt enable.
7	ISE_PINT3	[1]	Pin interrupt / pattern match engine slice 3 interrupt enable.
8	ISE_UTICK	[1]	Micro-Tick Timer interrupt enable.
9	ISE_MRT	[1]	Multi-Rate Timer interrupt enable.
10	ISE_CTIMER0	[1]	Standard counter/timer CTIMER0 interrupt enable.
11	ISE_CTIMER1	[1]	Standard counter/timer CTIMER1 interrupt enable.
12	ISE_SCT	[1]	SCT interrupt enable.
13	ISE_CTIMER3	[1]	Standard counter/timer CTIMER3 interrupt enable.
14	ISE_FC0	[1]	Flexcomm Interface 0 interrupt enable

For the DMA module configuration, the PERIPHREQEN **bit in Configuration registers** of corresponding DMA channel must be cleared, the HWTRIGEN must be set.

Table 432. Configuration registers for channel 0 to 22 ((CFG[0:22], offset 0x400 (CFG0) to 0x560 (CFG22))

Bit	Symbol	Value	Description	Reset value
0	PERIPHREQEN	-	Peripheral request Enable. If a DMA channel is used to perform a memory-to-memory move, any peripheral DMA request associated with that channel can be disabled to prevent any interaction between the peripheral and the DMA controller.	0
		0	Disabled. Peripheral DMA requests are disabled.	
		1	Enabled. Peripheral DMA requests are enabled.	
1	HWTRIGEN		Hardware triggering enable for this channel.	0
		0	Disabled. Hardware triggering is not used.	
		1	Enabled. Use hardware triggering.	
3:2	-		Reserved. Read value is undefined, only zero should be written.	NA

The following part is the project code. It is developed with MCUXpresso ver11.5.0 and SDK_2.8.2_LPCXpresso55S69.zip. It is run on LPCXpresso55S69 EVK board.
The project must include the driver such as GPIO, CTimer, DMA and INPUTMUX



The appendix is the code, any issue, contact xiangjun.rong@nxp.com:

```
/**
 * @file    LPC55S69_Project.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"
#include "fsl_ctimer.h"
#include "fsl_inputmux.h"
#include "fsl_dma.h"



/* TODO: insert other include files here. */
static void DMA_Configfuation(void);
void LEDInit(void);
void CtimerInit(void);
void ctimer_match0_callback(uint32_t flags);
void ctimer_match1_callback(uint32_t flags);
void DMA_Callback(dma_handle_t *handle, void *param, bool transferDone,
```

```
uint32_t tcds);
void CTimerStart(void);

/* TODO: insert other definitions and declarations here. */
#define DMA_DESCRIPTOR_NUM     4U
#define DMA_LINK_TRANSFER_COUNT 10
/*
 * @brief   Application entry point.
 */
/* Array of function pointers for callback for each channel */
ctimer_callback_t ctimer_callback_table[] = {
        ctimer_match0_callback, ctimer_match1_callback, NULL, NULL, NULL,
NULL, NULL, NULL};

void ctimer_match0_callback(uint32_t flags)
{
    //GPIO->NOT[1]|=1<<4;
    //   GPIO->NOT[1]|=1<<6;

}
void ctimer_match1_callback(uint32_t flags)
{
  //  GPIO->NOT[1]|=1<<4;

}
DMA_ALLOCATE_LINK_DESCRIPTORS(s_dma_table, DMA_DESCRIPTOR_NUM);
static volatile bool s_Transfer_Done = false;
uint32_t ledPattern,ledPattern1;
uint32_t testWord;
static volatile uint32_t s_transferCount;
int main(void) {

    /* Init board hardware. */
    BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitBootPeripherals();
#ifndef BOARD_INIT_DEBUG_CONSOLE_PERIPHERAL
    /* Init FSL debug console. */
    BOARD_InitDebugConsole();
#endif
    ledPattern=1<<4; //light LED_BLUE via DMA  transfer
    ledPattern1=1<<6; //light LED_RED via DMA  transfer
    PRINTF("Hello World\n");
    LEDInit();
```

```
    CtimerInit();
    DMA_Configfuation();
    CTimerStart();
    /* Force the counter to be placed into memory. */
    volatile static int i = 0 ;
    /* Enter an infinite loop, just incrementing a counter. */
    while(1) {
        i++ ;
        /* 'Dummy' NOP to allow source level single stepping of
            tight while() loop */
        __asm volatile ("nop");
    }
    return 0 ;
}

/* User callback function for DMA transfer. */
void DMA_Callback(dma_handle_t *handle, void *param, bool transferDone,
uint32_t tcds)
{
    __asm("nop");

    if (transferDone)
    {
        if (++s_transferCount > DMA_LINK_TRANSFER_COUNT)
        {
            s_Transfer_Done = true;
            DMA_DisableChannel(DMA0, 0);
        }
    }

}


#define DMACHANNEL 0
dma_handle_t g_DmaHandleStruct;      /* Handler structure for using DMA. */
const uint32_t g_XferConfig =
    DMA_CHANNEL_XFER(true,                        /* Reload link descriptor
after current exhaust, */
                     true,                       /* Clear trigger status. */
                     true,                       /* Enable interruptA. */
                     false,                      /* Not enable interruptB.
*/
                     sizeof(uint32_t),            /* Dma transfer width. */
                     kDMA_AddressInterleave0xWidth, /* Dma source address no
```

```
interleave  */
                    kDMA_AddressInterleave0xWidth, /* Dma destination
address no interleave  */
                    10*sizeof(uint32_t)             /* Dma transfer byte.
*/
    );
static void DMA_Configfuation(void)
{
    dma_channel_config_t dmaChannelConfigStruct;
    dma_channel_trigger_t dmaChannelTriggerStruct;

    /* Init DMA. This must be set before INPUTMUX_Init() for DMA peripheral
reset will clear the mux setting. */
    DMA_Init(DMA0);

    /* Configure DMAMUX. */
    INPUTMUX_Init(INPUTMUX);
    INPUTMUX_AttachSignal(INPUTMUX, DMACHANNEL, kINPUTMUX_Ctimer0M0ToDma0);

    /* Configure DMA. */
    DMA_EnableChannel(DMA0, DMACHANNEL);
    DMA_CreateHandle(&g_DmaHandleStruct, DMA0, DMACHANNEL);
    DMA_SetCallback(&g_DmaHandleStruct, DMA_Callback, NULL);
    /*
     * Configure the DMA trigger:
     * The DATAVALID of ADC will trigger the interrupt. This signal is also
for thie DMA triger, which is changed 0 ->
     * 1.
     */
    dmaChannelTriggerStruct.burst = kDMA_EdgeBurstTransfer1;
    dmaChannelTriggerStruct.type  = kDMA_RisingEdgeTrigger;
    dmaChannelTriggerStruct.wrap  = kDMA_NoWrap;

    /* Prepare and submit the transfer. */
    DMA_PrepareChannelTransfer(&dmaChannelConfigStruct,             /* DMA
channel transfer configurationstructure. */
                               (void *)&ledPattern,       /* DMA transfer
source address. */
                               (void *)&GPIO->NOT[1],             /* DMA
transfer destination address. */
                               g_XferConfig,                       /* Xfer
configuration */
                               kDMA_MemoryToMemory,                /* DMA
transfer type. */
```

```
                            &dmaChannelTriggerStruct,           /* DMA
channel trigger configurations. */
                            (dma_descriptor_t *)&(s_dma_table[0]) /*
Address of next descriptor. */
    );
    DMA_SubmitChannelTransfer(&g_DmaHandleStruct, &dmaChannelConfigStruct);

    /* Set two DMA descripters to use ping-pong mode.  */
    DMA_SetupDescriptor(&(s_dma_table[0]), g_XferConfig, (void
*)&ledPattern, (void *)&GPIO->NOT[1],
                        (dma_descriptor_t *)&(s_dma_table[1]));
    DMA_SetupDescriptor(&(s_dma_table[1]), g_XferConfig, (void
*)&ledPattern1, (void *)&GPIO->NOT[1],
                        (dma_descriptor_t *)&(s_dma_table[0]));
    //CTIMER0->IR
}


//LED is connected to PIO1_4 pin
#define IOCON_PIO_FUNC0 0x00
#define IOCON_PIO_MODE_PULLUP 0x20u
void LEDInit(void)
{
    GPIO_PortInit(GPIO, BOARD_LED_BLUE_GPIO_PORT); //GPIO port 1
    /* Enables the clock for the I/O controller.: Enable Clock. */
        CLOCK_EnableClock(kCLOCK_Iocon);
    const uint32_t LED_BULE = (/* Pin is configured as PIO1_4 */
                            IOCON_PIO_FUNC0 |
                            /* Selects pull-up function */
                            IOCON_PIO_MODE_PULLUP |
                            /* Standard mode, output slew rate control is
enabled */
                            IOCON_PIO_SLEW_STANDARD |
                            /* Input function is not inverted */
                            IOCON_PIO_INV_DI |
                            /* Enables digital function */
                            IOCON_PIO_DIGITAL_EN |
                            /* Open drain is disabled */
                            IOCON_PIO_OPENDRAIN_DI);
    /* PORT1 PIN4 (coords: 1) is configured as PIO1_4 */
    IOCON_PinMuxSet(IOCON, BOARD_LED_BLUE_GPIO_PORT,
BOARD_LED_BLUE_GPIO_PIN, LED_BULE);

    //set up GPIO
```

```
    /* Enables the clock for the GPIO1 module */
       CLOCK_EnableClock(kCLOCK_Gpio1);

       gpio_pin_config_t LED_BULE_config = {
           .pinDirection = kGPIO_DigitalOutput,
           .outputLogic = 0U
       };
       /* Initialize GPIO functionality on pin PIO1_4 (pin 1)  */
    GPIO_PinInit(GPIO, BOARD_LED_BLUE_GPIO_PORT, BOARD_LED_BLUE_GPIO_PIN,
&LED_BULE_config);

    //GPIO_PortToggle(GPIO, BOARD_LED_BLUE_GPIO_PORT, 1u <<
BOARD_LED_BLUE_GPIO_PIN);

//set up another LED PIO1_6
    const uint32_t LED_RED = (/* Pin is configured as PIO1_4 */
                              IOCON_PIO_FUNC0 |
                              /* Selects pull-up function */
                              IOCON_PIO_MODE_PULLUP |
                              /* Standard mode, output slew rate control is
enabled */
                              IOCON_PIO_SLEW_STANDARD |
                              /* Input function is not inverted */
                              IOCON_PIO_INV_DI |
                              /* Enables digital function */
                              IOCON_PIO_DIGITAL_EN |
                              /* Open drain is disabled */
                              IOCON_PIO_OPENDRAIN_DI);
    /* PORT1 PIN6 (coords: 1) is configured as PIO1_6 */
    IOCON_PinMuxSet(IOCON, BOARD_LED_BLUE_GPIO_PORT,
BOARD_LED_RED_GPIO_PIN, LED_RED);

    //set up GPIO
       gpio_pin_config_t LED_RED_config = {
           .pinDirection = kGPIO_DigitalOutput,
           .outputLogic = 0U
       };
       /* Initialize GPIO functionality on pin PIO1_4 (pin 1)  */
    GPIO_PinInit(GPIO, BOARD_LED_BLUE_GPIO_PORT, BOARD_LED_RED_GPIO_PIN,
&LED_RED_config);
#if 0 //for only test
    GPIO_PortToggle(GPIO, BOARD_LED_BLUE_GPIO_PORT, 1u <<
BOARD_LED_RED_GPIO_PIN);
    __asm("NOP");
```

```
   GPIO->NOT[1]|=1<<6;

//////////////////////


    GPIO_PortToggle(GPIO, BOARD_LED_BLUE_GPIO_PORT, 1u <<
BOARD_LED_BLUE_GPIO_PIN);
    GPIO->NOT[1]|=1<<4;
#endif
}
#define CTIMER_CLK_FREQ CLOCK_GetFreq(kCLOCK_FroHf)
void CtimerInit(void)
{
    ctimer_config_t config;
    ctimer_match_config_t matchConfig;

    CLOCK_AttachClk(kFRO_HF_to_CTIMER0);
    CTIMER_GetDefaultConfig(&config);
    CTIMER_Init(CTIMER0, &config);

    matchConfig.enableCounterReset = true;
    matchConfig.enableCounterStop  = false;
    matchConfig.matchValue         = CTIMER_CLK_FREQ /2;
    matchConfig.outControl         = kCTIMER_Output_Toggle;
    matchConfig.outPinInitState    = true;
    matchConfig.enableInterrupt    = true;
    CTIMER_SetupMatch(CTIMER0, kCTIMER_Match_0, &matchConfig);

    matchConfig.enableCounterReset = false;
    matchConfig.enableCounterStop  = false;
    matchConfig.matchValue         = (CTIMER_CLK_FREQ /2)-100;
    matchConfig.outControl         = kCTIMER_Output_Toggle;
    matchConfig.outPinInitState    = true;
    matchConfig.enableInterrupt    = false;
    CTIMER_SetupMatch(CTIMER0, kCTIMER_Match_1, &matchConfig);
    CTIMER_RegisterCallBack(CTIMER0, &ctimer_callback_table[0],
kCTIMER_MultipleCallback);

}

void CTimerStart(void)
{
    CTIMER_StartTimer(CTIMER0);
   //__NVIC_DisableIRQ(CTIMER0_IRQn);
```

```
}
```