Nano-edge placement of eFlexPWM on KV4x family

1. The feature of KV4x family.

The assumed application field of Kv4x family is motor control and switch mode power supply, most of the peripherals of KV4x family are the same as that of DSC, but the core changes to Cortex-M4, it is sub-family of Kinetis.

In switch mode power supply application, high speed ADC converter, advanced PWM module which can generate complicated PWM signal with high resolution, timers, comparator are prerequisite. The KV4x family has eFlexPWM, FTM, high speed cyclic ADC converter, crossbar module, on-chip comparator, all the peripheral modules make it suitable in switch mode power supply application and advanced level motor control.

it is desired to improve controlling voltage resolution, but the higher the PWM signal frequency, the lower the voltage resolution is, for example, the usual PWM frequency is 100KHz in switched mode power supply, if we set the PWM driving clock frequency is 100MHz, the Voltage resolution is about less than 10 bits.

The eFlexPWM module can generate complicated PWM waveform, the KV4x has only ONE eFlexPWM module, the eFlexPWM module has 4 sub-modules, from theory, it can generate 3*4=12 independent PWM signals, both the rising and falling edge of each PWM signal can be controlled by firmware.

In order to improve the PWM resolution, the eFlexPWM supports the nano-edge placement and duty cycle dithering features, the eFlexPWM module with internal PLL can support both the nano-edge placement feature and dithering duty cycle feature.

2. nano-edge placement feature

The Nano-edge placement:

The nano-edge placement means that the PWM duty cycle can extend a fractional tick cycle time in order to increase PWM resolution, in other words, the PWM module can generate integer plus fractional duty cycle. For example, the PWM driving clock for KV4x is 100MHz, the 100MHz clock is the main tick, the main tick cycle time is 10ns. There is an internal PLL inside the eFlexPWM module, the PLL can multiply the 100MHz main clock with 32 and can output 3.2GHz clock, the 3.2GHz clock is the fractional clock. User can set the duty cycle with a number of main clock plus a number of fractional clock.

Assume that the main clock is 100MHz, the fractional clock is 3.2GHz. If you set the PWM frequency in 1MHz, the duty cycle is 50%, the High logic covers 50 main tick cycles, the low logic convers 50 main tick cycles. For nano-edge placement, the High and low logic can cover a number of both main tick cycles and a number of fractional tick cycles, in other words, the duty cycle can be x+y/32, x demotes the main tick cycle, the y denote the number of fractional cycle, it means that the duty cycle can be 50.03125, 50.0625, 50.09325,50.125,…….50.96875 when y is equals to 1, 2 3,4…31.

3. Test result and conclusion

The code can demonstrate the fractional duty cycle(nano-edge placement), it runs on TWR-KV46F150M with OSBDM mode plus primary Elev board.

The yellow channel on the screenshot is PWM_B0 signal which is tested by A39 pin of primary Elev board

The blue channel is PWM_A0 signal, which is tested by A40 pin of primary Elev board.

The PWM_A0/PWM_B0 signals are set up in independent mode, the VAL2=VAL4=0xFFFC, VAL3=VAL5=0x04, but the FRACVAL3=0x00, FRACVAL5=15<<11. From above scope screenshot, the PWM_B0 signal duty cycle extends by half main tick cycle.

The code can be ported to the other tools easily, because the code writes peripheral register address directly without calling ksdk function.

Appendix:

The code is developed with TWR-KV46F150M with OSBDM mode plus primary Elev board and MCUpresso IDE v10.2.0 tools by XiangJun Rong

```
/**
 * @file    MKV46F256xxx16_Project.c
 * @brief   Application entry point.
 */
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKV46F16.h"
#include "fsl_debug_console.h"
#define ALT6 6<<8
/* TODO: insert other include files here. */

/* TODO: insert other definitions and declarations here. */
void PLLset(void);
void pinsAssignment(void);
void nanoEdgePWM(void);
void PWMEnable(void);
/*
 * @brief   Application entry point.
 */
int main(void) {

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

    PRINTF("Hello World\n");
    PLLset();
    pinsAssignment();
    nanoEdgePWM();
    PWMEnable();
    __asm("nop");
    /* Force the counter to be placed into memory. */
    volatile static int i = 0 ;
    /* Enter an infinite loop, just incrementing a counter. */
    while(1) {
        i++ ;
    }
    return 0 ;
}

//there is an external 8MHz crystal Y1 on TWR-KV46F150M
```

```
//procedure to enter normal speed run mode, core/system/fast peripheral clock 100MHz,
bus/flash clock is 20MHz
//The MCGPLLCLK is 2xfast peripheral clock, it is 200MHz, in other words, have PLL
output 200mhz
//PTA18/Extal, PTA19/Xtal
//data sheet specification:
//PLL input clock frequency range is from 8~16MHz
//PLL VCO output clock frequency is from 180MHz to 360mHz
//MCG state machine: FEI->FBE->PBE->PEE

void PLLset(void)
{
	//The PTA18 is EXTAL0 pin in default, PTA19 as Xtal pin in default

	//MCG is set up in from FEI to FBE

//configure the core/system, fast peripheral and bus/flash clock
	//core/system=200MHz/2=100MHz, fast peripheral=200mhz/2=100Mhz,
bus/flash=200Mhz/10=20Mhz
	SIM->CLKDIV1=0x11090000;
	//From FEI to FBE
	OSC->CR|=0x80;
	MCG->C2|=0x04;  //set the EREFS bit to use external crystal
	MCG->C2|=0x28; //set RANGE as 2b10, in very high frequency range, HGO=1 in
high gain
	MCG->C1 = 0x98; //CLKS=10 to select external clock, FRDIV=3b011, by 256
divider to get 31.25KHz clock for the FLL input clock, IREFS=0 to select external
clock
	//polling the switching status
	while(!(MCG->S&0x02)) {} //polling OSCINIT0 bit
	while(MCG->S&0x10) {} //polling IREFST bit
	while((MCG->S&0x0C)!=0x08) {}

	//From FBE to PBE, PLL input reference frequency range is 8~16MHz, the
solution is (8M)*25=200MHz
	MCG->C5=0x00; //set PRDIV=0, divider is 1,
	MCG->C6=0x49;	//PLLS is set so that PLL works, VDIV=5b'01001, the
multiply is 25
	MCG->C5|=0x40;
	while(!(MCG->S&0x20)) {} //polling PLLST bit
	while(!(MCG->S&0x40)) {} //Polling LOCK0 bit

	//from PBE to PEE mode
	MCG->C1&=~(0xC0);
	while((MCG->S&0x0C)!=0x0C) {}
	__asm("nop"); //set a break point here for only test

}
//PTD0:PWMA_A0
//PTD1:PWMA_B0
//PTD2:PWMA_A1
//PTD3:PWMA_B1
void pinsAssignment(void)
{
```

```
	//enable PORTD gated clock
	SIM->SCGC5|=0x1000;
	//set PTE MUX  ALT6
	PORTD->PCR[0]&=~(0xF00);
	PORTD->PCR[0]|=(ALT6);

	PORTD->PCR[1]&=~(0xF00);
	PORTD->PCR[1]|=(ALT6);

	PORTD->PCR[2]&=~(0xF00);
	PORTD->PCR[2]|=(ALT6);

	PORTD->PCR[3]&=~(0xF00);
	PORTD->PCR[3]|=(ALT6);
}

void nanoEdgePWM(void)
{
	//enable PWMA clock
	SIM->SCGC4|=0xF000000;
	//PWMA register setting

		PWMA->SM[0].INIT=0xFFF8;
		PWMA->SM[0].VAL0=0x0000;
		PWMA->SM[0].VAL1=0x0007;  //the modulo is 256
		PWMA->SM[0].VAL2=0xFFFC;  //75% duty cycle
		PWMA->SM[0].VAL3=0x0004;
		PWMA->SM[0].VAL4=0xFFFC;  //25% duty cycle
		PWMA->SM[0].VAL5=0x0004;
		PWMA->SM[0].CTRL2=0x2000; //independent mode for PWMA_A0 and PWMA_B0, IP
bus clock
		PWMA->SM[0].CTRL=0x3400; //4 PWM opportunity, PWM clock=Fclk
		PWMA->SM[0].OCTRL=0x0000; //PWM does not inverter, PWM forced to logic 0
in fault state
		PWMA->SM[0].TCTRL=0x0000;
		PWMA->SM[0].INTEN=0x0000; //disable all interrupt
		PWMA->SM[0].DISMAP[0]=0x0000; //Disable fault mask
		PWMA->SM[0].DTCNT0=0x0000; //dead time is set to 0
		PWMA->SM[0].DTCNT1=0x0000;
		PWMA->SM[0].CTRL|=0x04;

		//SM1 module initialization

		PWMA->SM[1].INIT=0xFFF8;
		PWMA->SM[1].VAL0=0x0000;
		PWMA->SM[1].VAL1=0x07;    //the modulo is 256
		PWMA->SM[1].VAL2=0xFFFC;  //75% duty cycle
		PWMA->SM[1].VAL3=0x0004;
		PWMA->SM[1].VAL4=0xFFFC;  //25% duty cycle
		PWMA->SM[1].VAL5=0x0004;
		PWMA->SM[1].CTRL2=0x0202; //independent mode, SM0 clock source, the
INIT->SEL should be 10, which means
		//that the SM0 synchronize thw SM1, PWMX->INIT is set as a test
		PWMA->SM[1].CTRL=0x3400; //4 PWM opportunity, PWM clock=Fclk
		PWMA->SM[1].OCTRL=0x0000; //PWM does not inverter, PWM forced to logic 0
```

```
            PWMA->SM[1].TCTRL=0x0000;
            PWMA->SM[1].INTEN=0x0000;
            PWMA->SM[1].DISMAP[0]=0x0000; //Disable fault mask
            PWMA->SM[1].DTCNT0=0x0000; //dead time is set to 0
            PWMA->SM[1].DTCNT1=0x0000;
            PWMA->SM[1].CAPTCTRLX|=0x40; //PWMA1->X output
            PWMA->OUTEN|=0x0FF0;  //enable PWM output
      //    PWMA->FCTRL)=0xF000;      //fault logic setting
            PWMA->SM[1].CTRL|=0x04;


            //set PLL begin
            PMC->REGSC|=0x01;//setting the BGBE bit
            PWMA->SM[0].FRCTRL=0x114;
            PWMA->SM[1].FRCTRL=0x114;
            SIM->PWRC=0x100;
            while(!(SIM->PWRC&0x10000)) {}
            //SIM->SOPT2&=0x30000;
            //SIM->SOPT2|=0x10000;
            PWMA->MCTRL2=0x03;
            while(PWMA->MCTRL2!=0x03) {}
            //PLL is okay now
            PWMA->SM[0].FRACVAL5=15<<11;
            //set PLL end
                  //PWMA global register setting
                  PWMA->OUTEN|=0x0FF0;  //enable PWM output
                  PWMA->MASK=0x0000;  //disable PWM      mask
                  PWMA->SWCOUT=0x0000;  //determine dead time logic
                  //    PWMA->FCTRL)=0xF000;      //fault logic setting
                  PWMA->MCTRL|=0x0007; //must use the instruction, otherwise, the
counter will disorder, IPOL is cleared, PWM23 manipulate the duty cycle

            return;
      }

void PWMEnable(void)
{
PWMA->MCTRL|=0x0100;  //enable the PWM module
PWMA->MCTRL|=0x0200;  //enable the PWM module
PWMA->MCTRL|=0x0400;  //enable the PWM module
PWMA->MCTRL|=0x0800;  //enable the PWM module

}
```